



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 en torno 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

El proyecto en el que nos encontramos está dedicado a la representación de datos mediante dashboards, gracias a una herramienta de software libre llamada Grafana. La información utilizada nos permite analizar uno o varios proyectos concretos en profundidad, centrándonos en diferentes aspectos, como puede ser estudio de estadísticos (media, máximos/mínimos, desviaciones...), uso de tablas, listados o tops, o una representación más atractiva mediante la utilización de gráficos de barras, de dispersión, etc.

La información base necesaria es extraída de múltiples repositorios de Git y GitHub, y de diferentes listas de correo, gracias a la herramienta *opensource* GrimoireLab, creada precisamente para este propósito. Los datos son almacenados en índices de Elasticsearch, aplicación que nos permitirá tener acceso en Grafana a los diferentes índices creados durante la recolección.

Los diferentes dashboards creados, junto con la instalación y configuración de las diferentes aplicaciones utilizadas, se integran finalmente en un mismo *docker* o contenedor de imagen, gracias al uso de la herramienta Docker Hub. Esto permite un despliegue rápido, sencillo y unificado de todo el proyecto via web, ya que todo queda alojado en un servidor externo.

Summary

The project we are in is dedicated to the representation of data through dashboards, thanks to a free software tool called Grafana. The information obtained allows us to analyze one or more specific projects in depth, focusing on different aspects, such as the study of statistics (mean, maximum / minimum, deviations ...), use of tables, listings or tops, or maybe a more visual or attractive representation, through the use of bar graphs, scattering, etc.

The base information is extracted from multiple repositories of Git and GitHub, and from different mailing lists, thanks to the GrimoireLab opensource tool, precisely created for this purpose. The data is stored in Elasticsearch indexes, a tool that will allow us to access the different indexes created during the collection in Grafana.

The different dashboards created, the installation and configuration of the different applications used, are finally integrated into the same docker or image container, thanks to the use of the Docker Hub tool. This allows a quick and simple deployment of the entire project via web, since everything is hosted on an external server.

Índice general

1. Introducción	1
1.1. Contexto	1
1.2. Objetivo general	3
1.3. Objetivos secundarios	4
1.4. Estructura de la memoria	4
2. Estado del arte	7
2.1. Bases de datos	7
2.1.1. Elasticsearch	7
2.1.2. MariaDB	8
2.1.3. InfluxDB	9
2.2. Sistemas de visualización	9
2.2.1. Kibana	9
2.2.2. Grafana	10
2.2.3. Cauldron	11
2.3. Herramientas adicionales	12
2.3.1. Git	12
2.3.2. GitHub	13
2.3.3. GrimoireLab	14
2.3.4. Docker Hub	15
2.3.5. Overleaf	16
3. Diseño e implementación	17
3.1. Sprint 1 - Instalación de Docker Hub y GrimoireLab	18

3.1.1.	Preparando el contenedor	18
3.1.2.	Funcionamiento de GrimoireLab	19
3.1.3.	Exploración de dashboards con Kibiter/Kibana	20
3.2.	Sprint 2 - Grafana: instalación y utilización	21
3.2.1.	Instalación	21
3.2.2.	Enlazado y configuración de índices	23
3.2.3.	Exploración de índices (consola de ElasticSearch)	25
3.2.4.	Familiarización con la interfaz	27
3.3.	Sprint 3 - Primer panel y plugins de Grafana	30
3.3.1.	Configurando nuestro panel	30
3.3.2.	Plugins o complementos	32
3.4.	Sprint 4 - Dashboards funcionales y comparación	34
3.5.	Sprint 5 - Integración: construcción del contenedor de GrimoireLab/Grafana . .	35
3.5.1.	Acercamiento a la construcción de la imagen	35
3.5.2.	Creando un nuevo Dockerfile	36
3.5.3.	Creación de una nueva <i>docker image</i>	37
3.6.	Sprint 6 - Actualización del contenedor de Grafana	38
3.6.1.	Exportación/importación disjunta	39
3.6.2.	Exportación/importación abrupta	41
3.7.	Sprint 7 - Imagen final: alojamiento en la nube y contenedores múltiples	42
3.7.1.	Preparación del contenedor final: alojamiento en servidor externo	43
3.7.2.	Despliegue con distintos contenedores de imagen	44
3.8.	Arquitectura final del proyecto	46
4.	Resultados	49
4.1.	Evaluación y demo de Grafana	49
4.1.1.	Demostración de dashboards	51
4.2.	Plugins relevantes	54
4.3.	Comparación de herramientas: Kibana vs Grafana	57
5.	Conclusiones	61
5.1.	Consecución de objetivos	61

5.2. Aplicación de lo aprendido	62
5.3. Lecciones aprendidas	63
5.4. Trabajos futuros	64
5.5. Planificación temporal	64
A. Recopilación de capturas	67
Bibliografía	75

Índice de figuras

1.1.	Representación de dashboard usando Grafana	2
2.1.	Ejemplo de query utilizando la API de Elasticsearch	8
2.2.	Aspecto visual de dashboard en Kibana	10
2.3.	Ejemplo de dashboard	11
2.4.	Ejemplo de dashboard generado por Cauldron	12
2.5.	Arquitectura en árbol de Git	13
2.6.	Funcionamiento básico de GitHub	14
2.7.	Esquema de funcionamiento de Docker Hub	16
3.1.	Esquema del entorno de trabajo SCRUM	18
3.2.	Esquema de funcionamiento de GrimoireLab	20
3.3.	Logs mostrados tras el lanzamiento del contenedor de GrimoireLab	21
3.4.	Ejemplo de dashboard ofrecido en <i>Overview</i>)	22
3.5.	Ejemplo de configuración de índice	24
3.6.	Listado de los índices recopilados hasta el momento	25
3.7.	Ejemplo de query en Elasticsearch	26
3.8.	Interfaz de Grafana (menús superiores)	27
3.9.	Interfaz de Grafana (menús laterales)	29
3.10.	Interfaz de Grafana (central) y ejemplo de dashboard	30
3.11.	Ejemplo de query para panel de Grafana	32
3.12.	Ejemplo de configuración visual de panel tipo <i>Graph</i>	33
3.13.	Ejemplo final de <i>Graph Panel</i>	34
3.14.	Ejemplo del plugin <i>Radar Graph</i>	34

3.15. Funcionamiento de docker con Dockerfile	36
3.16. Extracto del código agregado al nuevo Dockerfile	37
3.17. Extracto del código agregado al nuevo Entrypoint	38
3.18. Logs del nuevo contenedor de Grafana	39
3.19. Ejemplo de fichero de provisionamiento para importación de dashboards	40
3.20. Extracto del nuevo Dockerfile (utilizando método de exp/imp abrupta)	42
3.21. Extracto del archivo <i>projects.json</i>	45
3.22. Diagrama de bloques del proyecto	47
4.1. Página <i>Home</i> final de Grafana	50
4.2. Git Dashboard - Mezcla de paneles nativos y plugins	52
4.3. Git Dashboard - Recreación de parte de Git Overview de Kibana	52
4.4. Mailing Lists - Recreación parcial del dashboard de Kibana	52
4.5. Extracto del dashboard <i>Plugins</i>	53
4.6. Extracto del dashboard <i>Plugins</i>	57
A.1. Git Dashboard - Mezcla de paneles nativos y plugins	67
A.2. Git Dashboard - Mezcla de paneles nativos y plugins (2)	68
A.3. Git Dashboard - Recreación de parte de Git Overview de Kibana	68
A.4. Mailing Lists - Recreación parcial del dashboard de Kibana	68
A.5. Extracto del dashboard <i>Plugins</i>	69
A.6. Extracto del dashboard <i>Plugins</i> (2)	70
A.7. Extracto del dashboard <i>Plugins</i> (3)	71
A.8. Extracto del dashboard <i>Plugins</i> (4)	72
A.9. Extracto del dashboard <i>Plugins</i> (5)	73
A.10. Extracto del dashboard <i>Plugins</i> (6)	74

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 infinidad de fuentes distintas. Esto es debido al fenómenos conocido como Big Data, que no es otra cosa que un gran volumen de datos, tanto estructurados como no estructurados, que inundan los negocios cada día. Es necesario para el usuario, por tanto, que la información siga un cierto orden, o esté dispuesta de tal forma que se pueda elegir que visualizar y que no, dependiendo de los intereses de cada uno. En este proyecto nos centramos precisamente en eso, en la visualización o representación de datos mediante el uso de dashboards, consiguiendo una estructuración total de los datos obtenidos.

Pero para organizar esa ingente cantidad de información, primero hay que obtenerla. Es por ello que entra en juego otro de las herramientas más relevantes de este proyecto, GrimoireLab¹. Gracias a ella, podemos recopilar todos los datos necesarios para después plasmarlos en forma de dashboards.

La manera de representar cada elemento, 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 elemento recolectado. Este proyecto se centra en abordar en profundidad una de ellas, Grafana², la cual goza de gran popularidad dentro del sector.

El uso de paneles dinámicos es una herramienta, que utilizada de la forma adecuada, permite

¹<https://chaoss.github.io/grimoirelab/>

²<https://grafana.com/>

al usuario explorar de un vistazo la parte de la información que pueda interesarle más o menos. 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, estadísticos... Todo ello nos permite observar la correlación existente entre los diferentes datos mostrados.

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.



Figura 1.1: Representación de dashboard usando Grafana

Pero no basta solo con eso. Hoy en día, al usuario final le gusta la unificación. Aplicaciones que lo tengan todo: instalación sencilla, fácil de usar, que se integre con otras herramientas, etc. Es por ello que en este proyecto se llevaron a cabo estas ideas, haciendo uso de dockers o

contenedores creados gracias a Docker Hub³, permitiendo un despliegue de nuestro trabajo de una manera rápida, cómoda y unificada. Es por eso que esta herramienta se alojaría finalmente en la nube, siendo además de tipo *open source*, para que cualquier usuario pudiera hacer uso de ella⁴.

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, solo que esta vez el foco era yo.

1.2. Objetivo general

Este proyecto en el que nos encontramos se compone de 4 objetivos principales:

- Representación de datos con Grafana gracias al uso de GrimoireLab, la herramienta que nos permite la recolección y almacenamiento de datos extraídos de Git⁵, GitHub⁶ y pi-permail⁷.
- Unificación de ambas en un mismo *docker* o contenedor gracias a Docker Hub, automatizando así el despliegue de ambas aplicaciones en una misma imagen.
- Uso de multicontenedores: despliegue de varios dockers de distintos proyectos, ahorrando tiempo en la extracción, enriquecimiento y visualización de datos.
- Comparación de las dos herramientas de representación utilizadas: Grafana y Kibana⁸. Para ello se exploran los límites de cada una de ellas, señalando sus semejanzas y diferencias.

³<https://hub.docker.com/>

⁴<https://hub.docker.com/repository/docker/onac8/grafana-grimoirelab>

⁵<https://git-scm.com/>

⁶<https://https://Github.com/>

⁷<https://lists.linuxfoundation.org/mailman/listinfo>

⁸<https://www.elastic.co/es/products/kibana>

1.3. Objetivos secundarios

Para conseguir los resultados propuestos en el objetivo principal, ha sido necesario manejar al completo distintas herramientas, como es el caso de Elasticsearch⁹, Kibana y Docker Hub.

Primeramente, se aborda 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, es necesaria la formación en este tipo de BBDD, hasta ahora nuevas para mí. 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 GrimoireLab. Una vez aprendido su total manejo, servirá de guía para obtener resultados similares a los proporcionados por Grafana, y realizar así su posterior comparación. Finalmente, es vital el estudio completo de la herramienta de creación de imágenes y contenedores, Docker Hub, ya que nos servirá para el despliegue de las herramientas unificadas.

1.4. 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. Se aborda el contexto actual y las tecnologías ofrecidas, así como la finalidad de este trabajo y sus objetivos generales y particulares. Además, se proporciona una estructura de la tesis, explicando el contenido de cada sección.
- **Capítulo 2: Estado del arte.** Aquí nos encontraremos con una descripción general de todas las herramientas utilizadas en este proyecto (historia, características, uso...).
- **Capítulo 3: 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 4: Resultados.** Análisis de los resultados finales obtenidos.

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

- **Capítulo 5: 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 6: Apéndice.** En este apartado se incluyen las diferentes capturas realizadas durante la implementación del trabajo, agrupándolas por las diferentes secciones en las que se ha estructurado.

Capítulo 2

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, así como las diferentes herramientas complementarias que han servido de ayuda o inspiración para afrontarlo. Las tecnologías estarán agrupadas por secciones, según la labor que desempeñan. En este caso contamos con tres secciones diferentes: visualizadores de datos, bases de datos y herramientas adicionales.

2.1. Bases de datos

2.1.1. Elasticsearch

Elasticsearch es un servidor de búsqueda y análisis basado en Lucene¹. Es de código abierto, basado en Java, y con licencia Apache². Trabaja conjuntamente con un motor de recopilación de datos llamado Logstash³, y una plataforma de análisis y visualización llamada Kibana. 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, permitiendo usar, por ejemplo, una

¹<https://lucene.apache.org/>

²<http://httpd.apache.org/>

³<https://www.elastic.co/es/products/logstash>

consola para realizar las diferentes consultas.

- Provee un motor de búsqueda de texto completo, distribuido y con capacidad de multi-tenencia con documentos JSON.
- 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).

En nuestro caso, Elasticsearch ha sido la base de datos utilizada durante todo el proyecto. En ella se almacenan los elementos recopilados por GrimoireLab, y sus diferentes índices nos sirven para representar los datos en Grafana más adelante.

```

CHAOS
Dev Tools
Console
GET _cat/indices
1 yellow open git_aoc_grimoirelab-enriched YnKnY3CPN6kjmlJztVhwm 5 1 35119 0 26.6mb 26.6mb
2 yellow open git_grimoirelab-raw uhTZV1nsbQuleabPhz9ag 5 1 14186 0 17.9mb 17.9mb
3 yellow open git_grimoirelab-zip XZPfDfjGOGC1H448p0Bw 5 1 14186 0 23.5mb 23.5mb
4 yellow open github_grimoirelab-raw 8nd8cn3RgcH1H448p0Bw 5 1 1519 0 15.3mb 15.3mb
5 yellow open grimoirelab_identities_cache a_V7Hcap5ja3IPV30_4_jg 5 1 445 4625 675.5kb 675.5kb
6 yellow open mail_grimoirelab-raw Q0ST1Mhs5WqqrCcabaVA 5 1 2716 10 7.4mb 7.4mb
7 yellow open mail_grimoirelab V8DV0g5Q0hNPsCS1xbiQ 5 1 2716 1995 13.4mb 13.4mb
8 yellow open git_grimoirelab SHMYTHPMSCW6D0TMLzxLkg 5 1 14186 1759 67.9mb 67.9mb
9

```

Figura 2.1: Ejemplo de query utilizando la API de Elasticsearch

2.1.2. MariaDB

MariaDB⁴ es uno de los más populares sistemas de gestión de bases de datos relacionales. Deriva directamente de MySQL⁵, aunque con licencia GPL (General Public License). Fue desarrollado por Michael Widenius (fundador de MySQL), la fundación MariaDB y la comunidad de desarrolladores de software libre. Tiene una alta compatibilidad con MySQL ya que posee las mismas órdenes, interfaces, API y bibliotecas, siendo su objetivo poder cambiar un servidor por otro directamente. Además, es una de las herramientas líderes en computación en la nube, y es la opción por defecto en la mayoría de las distribuciones Linux.

⁴<https://mariadb.org/>

⁵<https://www.mysql.com/>

En este proyecto es fundamental para el correcto funcionamiento de GrimoireLab. Las distintas tablas son generadas por SortingHat⁶, herramienta que permite la identificación de personas o usuarios, almacenando identidades para cada una de ellas. Es fundamental, por tanto, mantener un continuo *backup* o respaldo de estos datos si se requiere una futura exportación de todas las bases de datos del proyecto.

2.1.3. InfluxDB

InfluxDB⁷ es una herramienta *open-source* para la gestión de bases de datos, basadas en series de tiempo (TSDB). Es desarrollada y mantenida por InfluxData, está escrita en lenguaje Go, y está optimizada para obtener una respuesta rápida y un eficiente almacenamiento y recolección de datos de tipo temporal. Se usa mayormente para monitorización, aplicación de métricas, sensores para IoT y análisis de datos y estadísticos en tiempo real.

Esta herramienta es la utilizada por defecto en la mayoría de tutoriales y ejemplos de Grafana. Sirvió de guía para poder entender como se realizan los diferentes dashboards en Grafana, como se usan sus diferentes plugins, etc.

2.2. Sistemas de visualización

2.2.1. 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. Posee una intuitiva API con interfaz web RESTful, donde podemos encontrar, por ejemplo, la consola de Elasticsearch para consultas, paneles de configuración para las bases de datos, ventanas para la creación de dashboards y múltiples opciones adicionales de desarro-

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

⁷<https://www.influxdata.com/>

llador. 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.

La principal guía para una correcta visualización de datos, se consiguió observando los diferentes dashboards de Kibana, proporcionados por GrimoireLab. Esto sirvió para una futura comparación de las dos herramientas principales del proyecto: Grafana y Kibana.

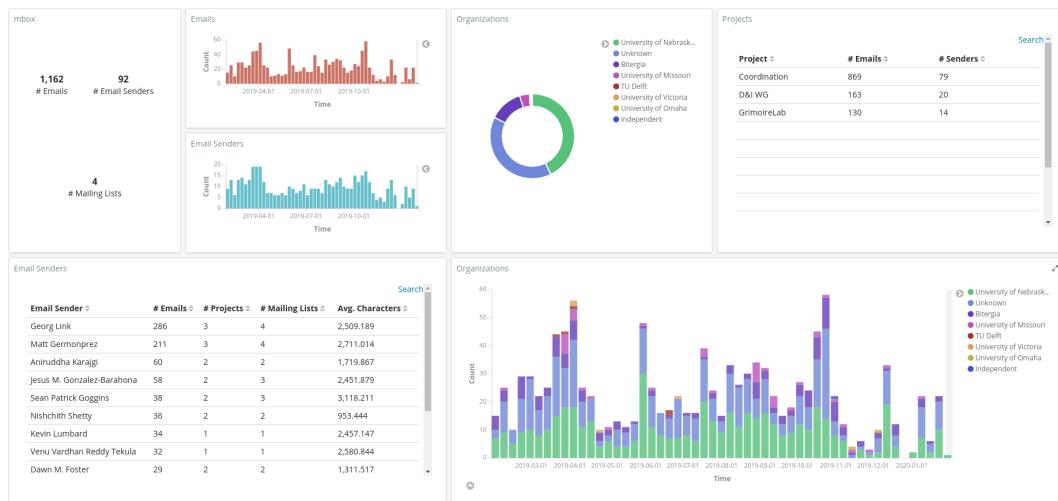


Figura 2.2: Aspecto visual de dashboard en Kibana

2.2.2. 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*⁸. 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.

⁸<https://golang.org/>

- 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, MariaDB⁹, MySQL¹⁰, PostgreSQL, CloudWatch...
- Gestión muy intuitiva de usuarios y *dashboards*. Esto permite que cada persona posea unos cuadros de mandos propios, y que pueda realizar *screenshots* (capturas estáticas) de estos, para poder compartirlos públicamente.

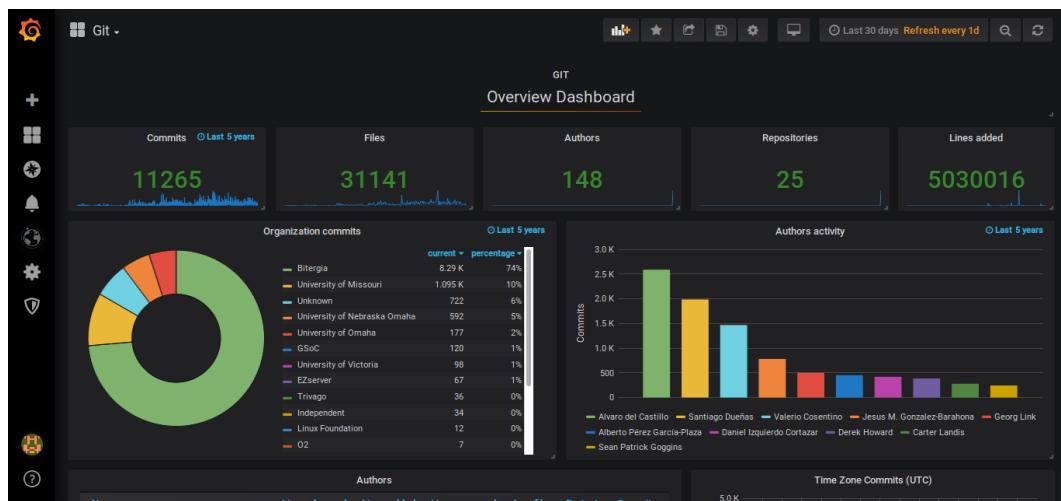


Figura 2.3: Ejemplo de dashboard

2.2.3. Cauldron

Cauldron es un conjunto de herramientas que permite el análisis vía web de proyectos alojados en servidores externos. Es una solución tipo Saas (Software as a Service), por lo que permite a los usuarios agregar información desde distintas plataformas de colaboración, y con diferentes fuentes de datos, ya que todo el soporte lógico de la aplicación se encuentra en la web. Esto permite a cada desarrollador definir qué datos incluir para su análisis y cuáles no,

⁹<https://mariadb.org/>

¹⁰<https://www.mysql.com/>

posibilitando el análisis de ciertas partes de un proyecto, y no en su totalidad (muy útil para trabajos de gran envergadura).

Es un claro ejemplo de integración de herramientas en una sola. Sirvió de inspiración para este proyecto, ya que en este también se unifican diferentes aplicaciones. Además, al ser un software alojado íntegramente en internet, se utilizó como idea para una proyección futura del proyecto en el que nos encontramos.

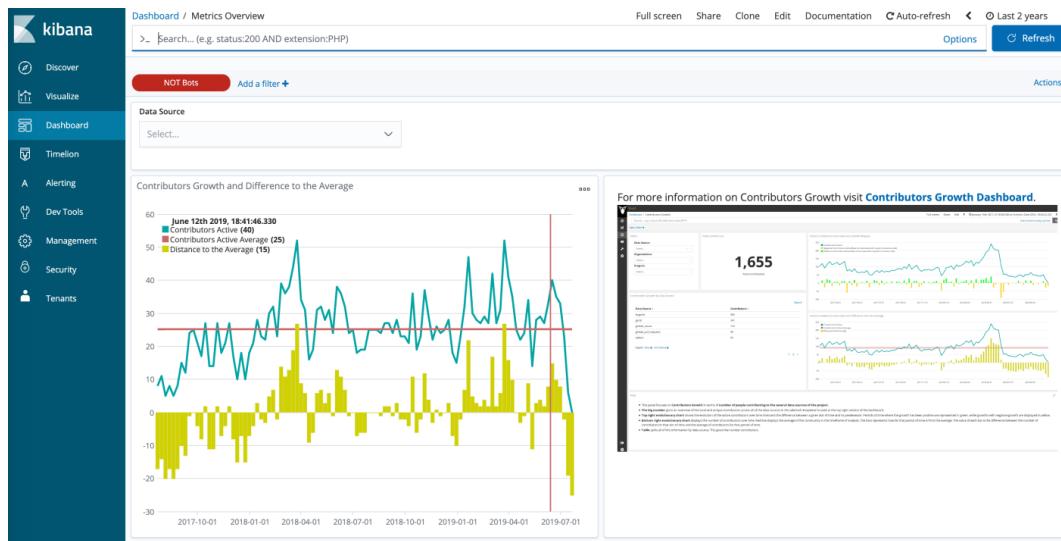


Figura 2.4: Ejemplo de dashboard generado por Cauldron

2.3. Herramientas adicionales

2.3.1. 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 CVS¹¹, SVN¹² o Mercurial¹³. Su funcionamiento es muy sencillo. Partimos

¹¹<https://www.gnu.org/software/trans-coord/manual/cvs/cvs.html>

¹²<https://subversion.apache.org/>

¹³<https://www.mercurial-scm.org/>

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 preserve a lo largo del tiempo. Una vez añadidos, basta con “comprometer” los cambios (hacer un *commit*) para que Git haga una captura o *screenshot* 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...

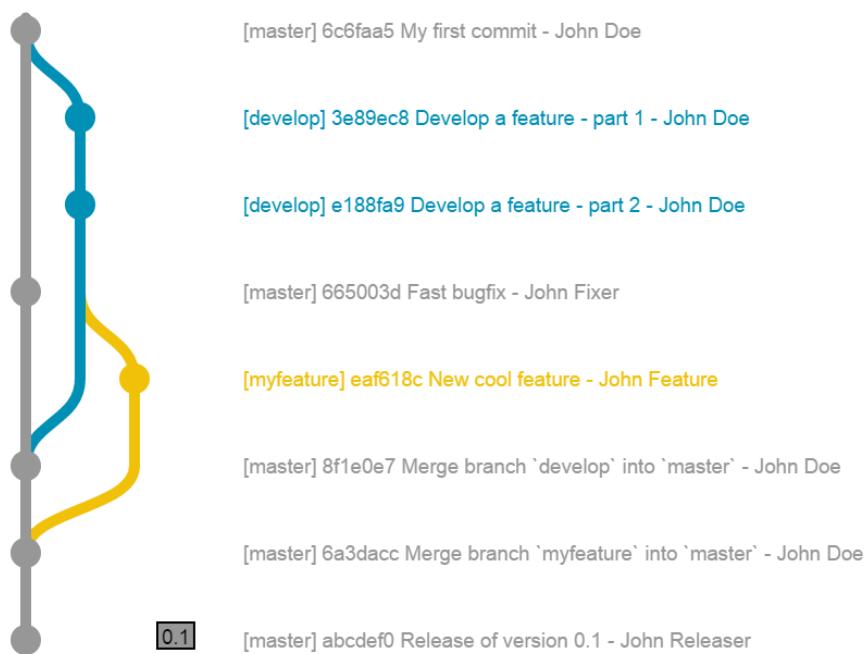


Figura 2.5: Arquitectura en árbol de Git

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.

2.3.2. 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 Buddy¹⁴ o Bitbucket¹⁵. 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.

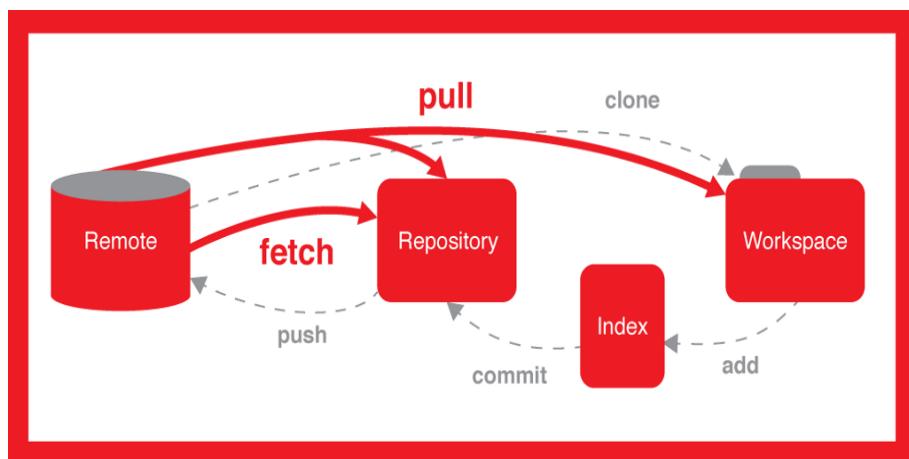


Figura 2.6: Funcionamiento básico de GitHub

2.3.3. GrimoireLab

GrimoireLab es un conjunto de herramientas de código libre, orientada a la investigación analítica de datos. Ha sido desarrollada por la empresa española Bitergia¹⁶. Permite extraer datos

¹⁴<https://buddy.works/>

¹⁵<https://bitbucket.org/>

¹⁶<https://bitergia.com/>

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 Perceval¹⁷, 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) como Jira o Bugzilla, *mailing lists* o listas de correo, diferentes fuentes de Stackoverflow¹⁸, etc. Actualmente, GrimoireLab forma parte del proyecto CHAOSS, perteneciendo también a la Fundación Linux.

2.3.4. 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 2.7 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 resuben las

¹⁷<https://chaoss.github.io/grimoirelab-tutorial/perceval/intro.html>

¹⁸<https://stackoverflow.com/>

imágenes en Docker Hub.

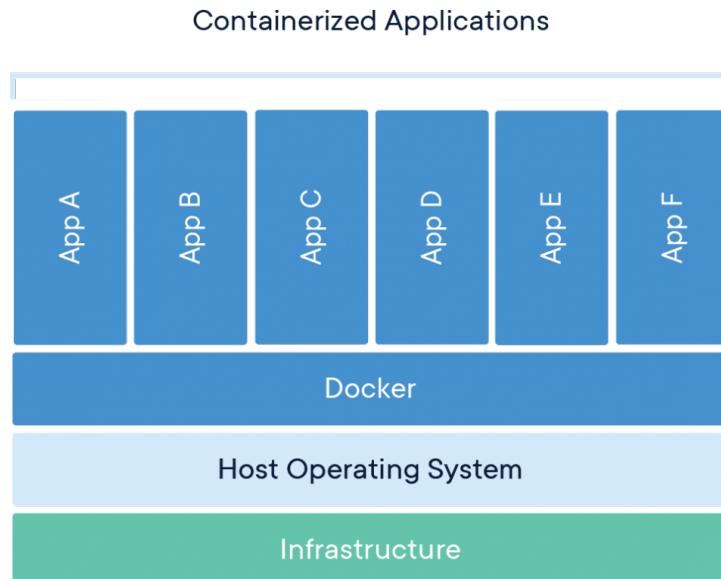


Figura 2.7: Esquema de funcionamiento de Docker Hub

2.3.5. Overleaf

Overleaf es un editor de LaTeX¹⁹ 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, ShareLatex. 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 TeXmaker²⁰, LyX²¹, 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.

¹⁹<https://www.latex-project.org/>

²⁰<https://www.xmlmath.net/texmaker/>

²¹<https://www.lyx.org/>

Capítulo 3

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 SCRUM¹.

SCRUM

Scrum es un 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>

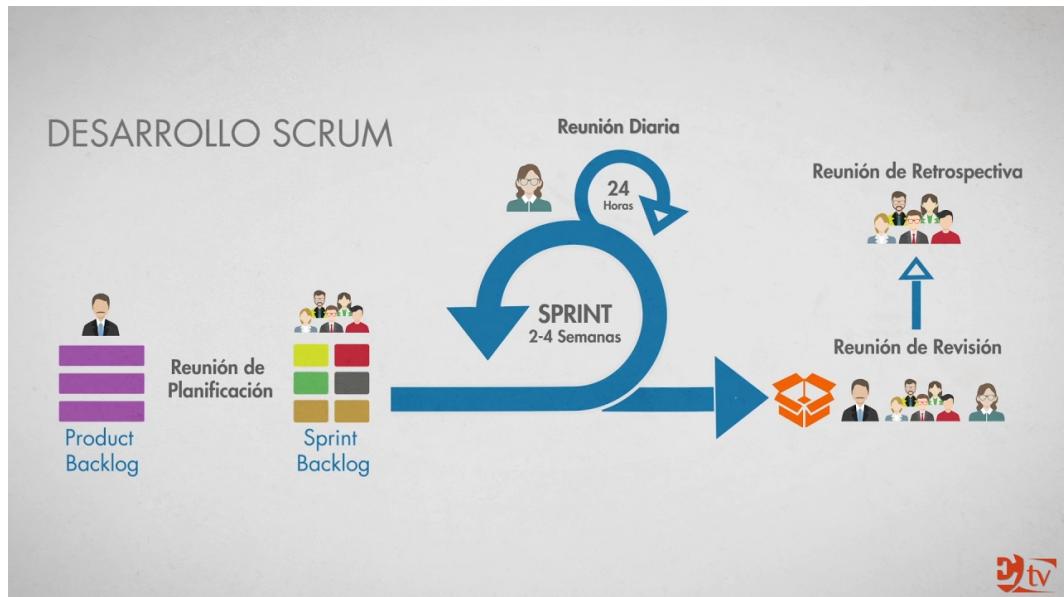


Figura 3.1: Esquema del entorno de trabajo SCRUM

3.1. Sprint 1 - Instalación de 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 2.3.4, que nos permitirá utilizar un contenedor de imagen con GrimoireLab, preparado para poder visualizar rápidamente diferentes paneles de control. Más adelante, en la Sección 3.5 (creación/uso de dockers propios), se profundiza en el funcionamiento de la herramienta.

3.1.1. 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).

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

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

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 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 3.4. Toda la información ampliada se encuentra disponible en el tutorial de la página del proyecto GrimoireLab⁶.

3.1.2. Funcionamiento de GrimoireLab

Antes de continuar explorando la herramienta de GrimoireLab, se decidió estudiar exhaustivamente su funcionamiento. Para ello, el esquema de la Figura 3.2 fue de gran ayuda:

En la Figura 3.2 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 Gri-

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/>

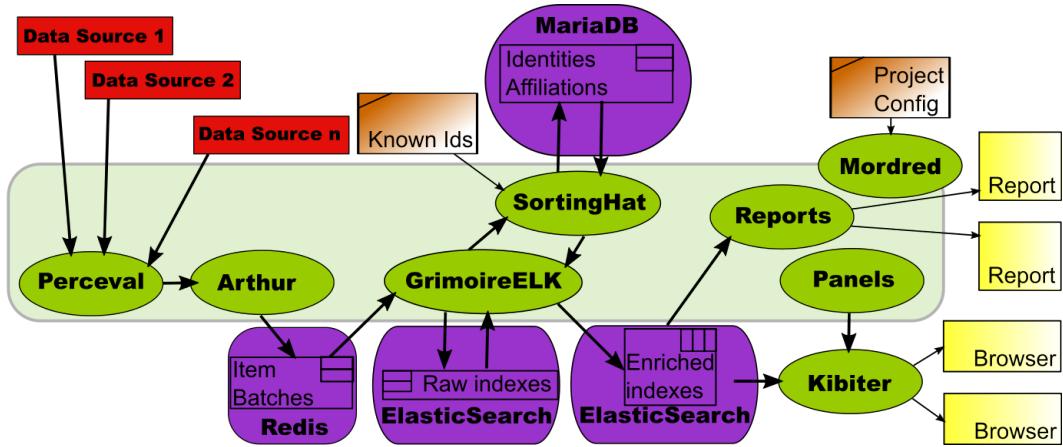


Figura 3.2: Esquema de funcionamiento de GrimoireLab

moireELK (que almacena los datos extraídos como índices crudos, y los utiliza para producir índices enriquecidos⁸, ambos en Elasticsearch), para finalmente llegar a Kibiter (herramienta *soft fork* de Kibana, para visualizar dashboards interactivos).

GrimoireELK (análogo al proyecto ELK visto en la Sección 2.1.1) 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.

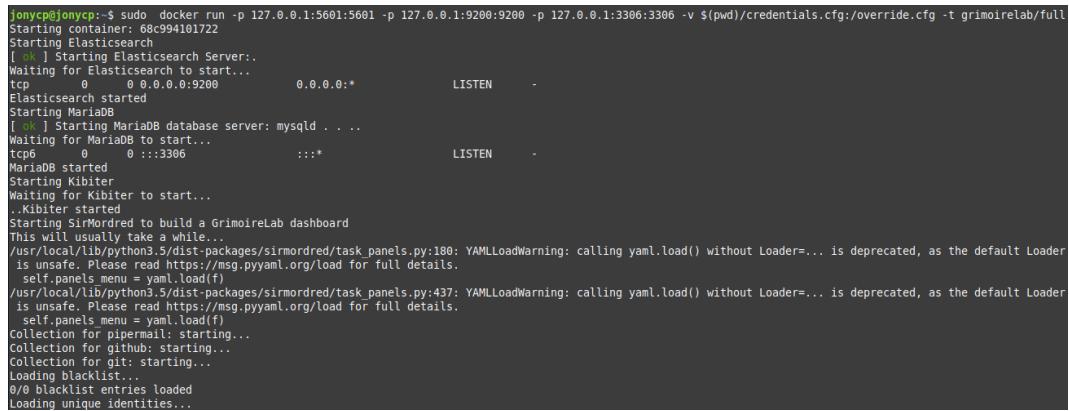
3.1.3. Exploración de dashboards con Kibiter/Kibana

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 3.1.1 (instalación de Docker Hub), tendremos inicializado el contenedor, el cual irá mostrando distintas líneas en el terminal a modo de *logs*, como se aprecia en la Figura 3.3. Una vez recolectada la información y enriquecidos los índices (puede demorar un tiempo), podremos acceder a la página desde el navegador (<http://localhost:5601>),

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

mostrándose así nuestro primer dashboard. Todos los aspectos relacionados con la base de datos se profundizan en la Sección 3.2.2 (configuración de índices de Elasticsearch).



```
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: 68c94101722
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 ::1:3306           ::*:*
MariaDB started
Starting Kibana
Waiting for Kibana to start...
Kibana 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 3.3: 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 3.4. 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 4.3 se ahonda más en el uso de Kibana, y se compara con Grafana, la herramienta principal de este proyecto.

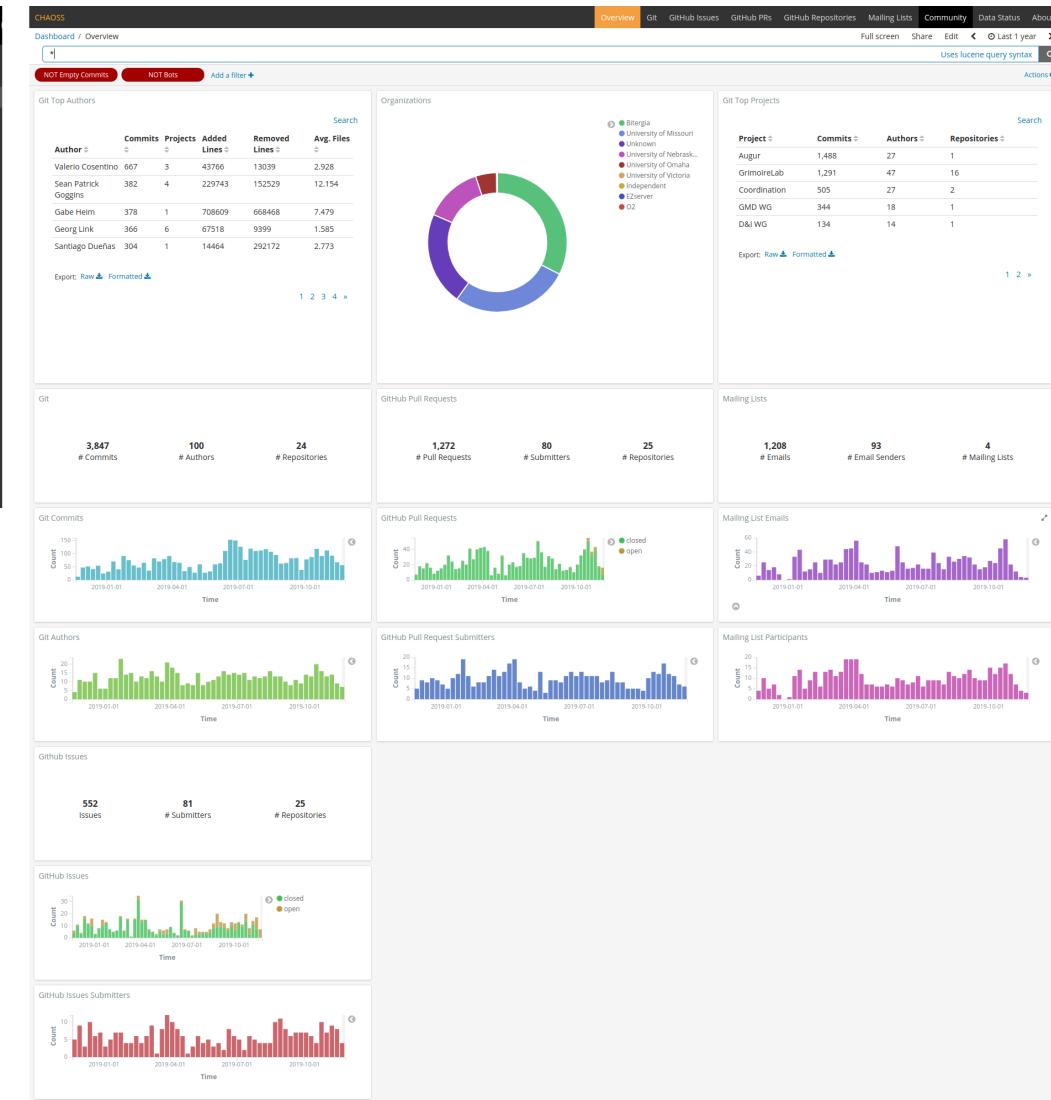
3.2. Sprint 2 - Grafana: instalación y utilización

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. Por tanto, los objetivos de este sprint fueron la instalación, familiarización con la herramienta y enlace de esta con nuestra base de datos.

3.2.1. Instalación

Para obtener todas las dependencias necesarias se hizo uso de la documentación⁹ de Grafana. La instalación local es tan simple como ejecutar las siguientes sentencias:

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

Figura 3.4: Ejemplo de dashboard ofrecido en *Overview*)

```

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 o limitaciones de diseño que presentaba la propia aplicación. Por suerte, es un software en continuo desarrollo y su mantenimiento y mejora están muy al día.

Ahora, para poder acceder a la herramienta, debemos iniciar el servicio con la siguiente instrucción:

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

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>	Para acceso remoto a una BD dada
<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 3.1: Parámetros de configuración de la BD

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 “es-
cuchando” 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.

3.2.2. Enlazado y configuración de índices

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/llenar 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 3.5. 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.

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

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

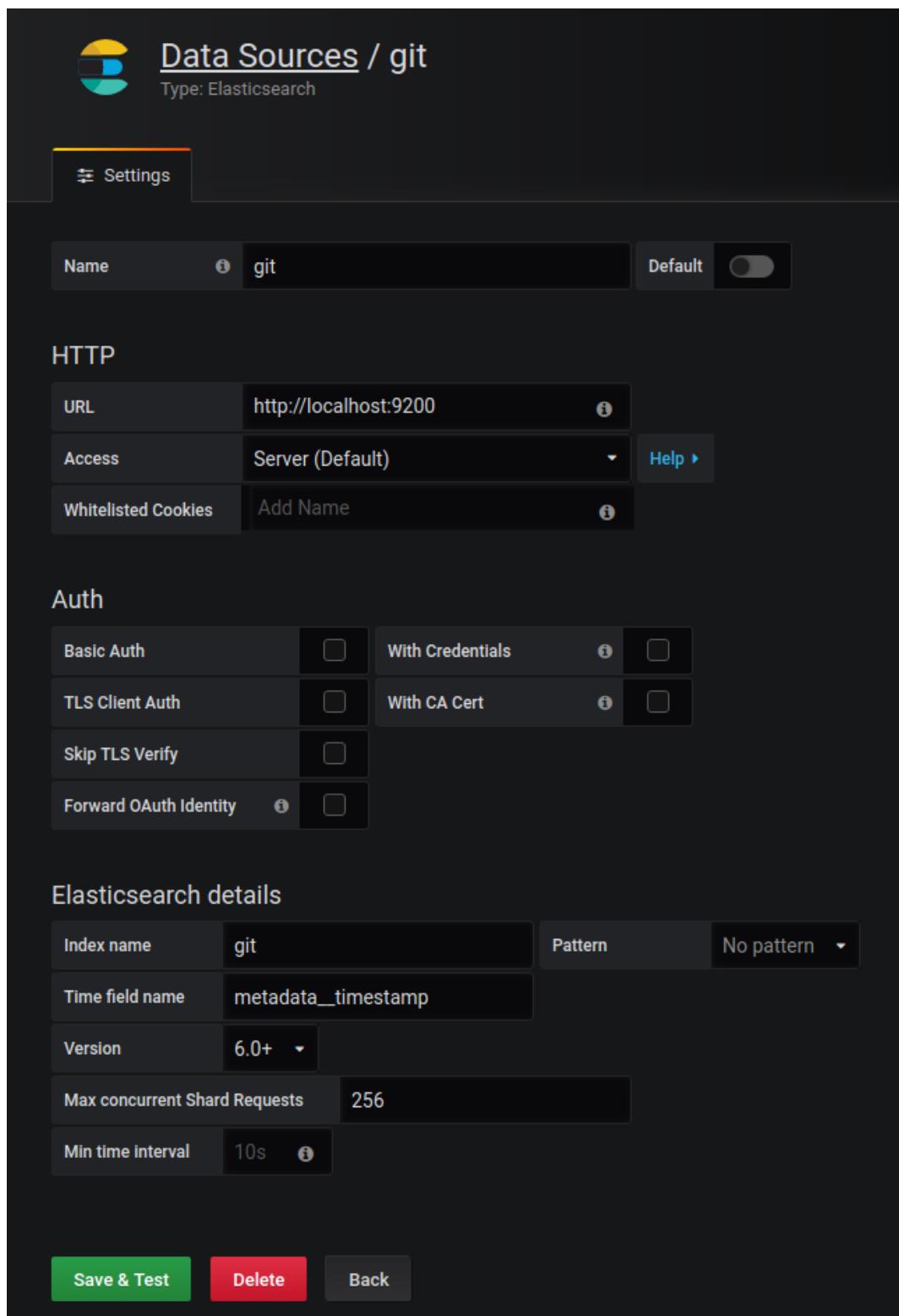


Figura 3.5: Ejemplo de configuración de índice

1	yellow open git_aco_grimoirelab-enriched	YnKnY3GPBw6kjmlJztVhnw	5	1	35119	0	26.6mb	26.6mb
2	yellow open git_grimoirelab-raw	uHTZv1pb5Q-oueabphz9ag	5	1	14186	0	17.9mb	17.9mb
3	yellow open _kibana	ZVUPs0785SGCI189d97MPg	1	1	198	4	234.8kb	234.8kb
4	yellow open github_grimoirelab-raw	BnqCnD3RgU1M1HAb0P0B	5	1	1519	0	15.3mb	15.3mb
5	yellow open grimoirelab_identities_cache	97V7KfLjJLJyJLJyJLJyJLJy	5	1	1	462	67.9mb	675.5kb
6	yellow open mail_grimoirelab-raw	Q0ST1Mhs5WqrCcabaA	5	1	2716	18	7.4mb	7.4mb
7	yellow open mail_grimoirelab	v8DV0qg5Q8hPcsCS1xbiQ	5	1	2716	1995	13.4mb	13.4mb
8	yellow open git_grimoirelab	5hmVTHPMScW60tMLzxLkg	5	1	14186	1759	67.9mb	67.9mb
9								

Figura 3.6: Listado de los índices recopilados hasta el momento

3.2.3. Exploración de índices (consola de ElasticSearch)

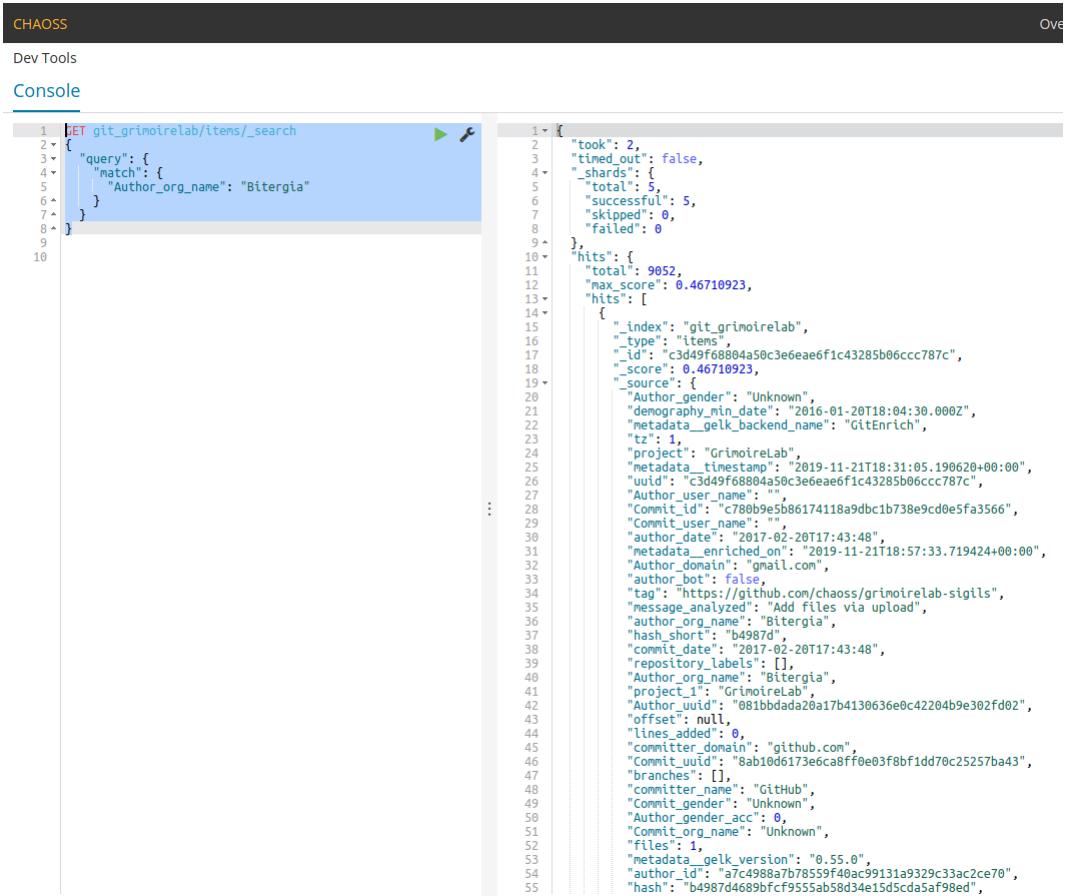
Para escoger correctamente los *Data Sources* de Grafana, hemos de estudiar detenidamente los índices de la base de datos. Para ello contamos con una consola proporcionada por Kibana, que se accede desde el ícono de la herramienta (DevTool>Console). Desde ahí, podemos realizar *queries* directamente a los diferentes índices. En la Figura 3.6 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 item 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 3.7, 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).



The screenshot shows the CHAOS Dev Tools interface with the 'Console' tab selected. On the left, a code editor displays a JSON search query:

```

1  GET git_grimoirelab/items/_search
2  {
3    "query": {
4      "match": {
5        "Author_org_name": "Bitergia"
6      }
7    }
8  }
9
10

```

On the right, the search results are displayed as a JSON object:

```

1  {
2    "took": 2,
3    "timed_out": false,
4    "_shards": {
5      "total": 5,
6      "successful": 5,
7      "skipped": 0,
8      "failed": 0
9    },
10   "hits": {
11     "total": 9052,
12     "max_score": 0.46710923,
13     "hits": [
14       {
15         "_index": "git_grimoirelab",
16         "_type": "items",
17         "_id": "c3d49f68804a50c3e6eae6f1c43285b06ccc787c",
18         "_score": 0.46710923,
19         "_source": {
20           "Author_gender": "Unknown",
21           "demography_min_date": "2016-01-20T18:04:30.000Z",
22           "metadata_gelk_backend_name": "GitEnrich",
23           "tz": "1",
24           "project": "GrimoireLab",
25           "metadata_timestamp": "2019-11-21T18:31:05.190620+00:00",
26           "uuid": "c3d49f68804a50c3e6eae6f1c43285b06ccc787c",
27           "Author_user_name": "",
28           "Commit_id": "c780b9e5b86174118a9dbc1b738e9cd0e5fa3566",
29           "Commit_user_name": "",
30           "author_date": "2017-02-20T17:43:48",
31           "metadata_enriched_on": "2019-11-21T18:57:33.719424+00:00",
32           "Author_domain": "gmail.com",
33           "author_bot": false,
34           "tag": "https://github.com/chaoss/grimoirelab-sigils",
35           "message_analyzed": "Add files via upload",
36           "author_org_name": "Bitergia",
37           "hash_short": "b4987d",
38           "commit_date": "2017-02-20T17:43:48",
39           "repository_labels": [],
40           "Author_org_name": "Bitergia",
41           "project_1": "GrimoireLab",
42           "Author_uid": "081bbddada20a17b4130636e0c42204b9e302fd02",
43           "offset": null,
44           "lines_added": 0,
45           "committer_domain": "github.com",
46           "Commit_uid": "8ab10d6173e6ca8ff0e03fb1dd70c25257ba43",
47           "branches": [],
48           "committer_name": "GitHub",
49           "Commit_gender": "Unknown",
50           "Author_gender_acc": 0,
51           "Commit_org_name": "Unknown",
52           "files": 1,
53           "metadata_gelk_version": "0.55.0",
54           "author_id": "a7c4988a7b78559f40ac99131a9329c33ac2ce70",
55           "hash": "b4987d4689bfcf9555ab58d34e15d5cda5af98ed"
}

```

Figura 3.7: Ejemplo de query en Elasticsearch

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



Figura 3.8: 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 *screenshot* o captura plana del mismo, o bien exportándolo como archivo JSON para poder importarlo en un futuro (más información en la Sección 3.6.1: Exportación de dashboards).
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.
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 playlists¹⁵ 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, 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*,

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

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

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

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

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

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* nativo de Grafana. Se profundiza en la Sección 3.3 (configuración de paneles primarios).
18. Herramienta de manejo temporal de un panel concreto. Se puede cambiar el rango de tiempo de cada panel de manera individual. Se analiza (de nuevo) en profundidad en la Sección 3.3.
19. Plugin de Grafana. En este caso es el complemento *Pie Chart*. Se profundiza en este aspecto en las Secciones 3.3.2 y 4.2 (instalación de plugins y complementos relevantes, respectivamente).
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 3.9: Interfaz de Grafana (menús laterales)

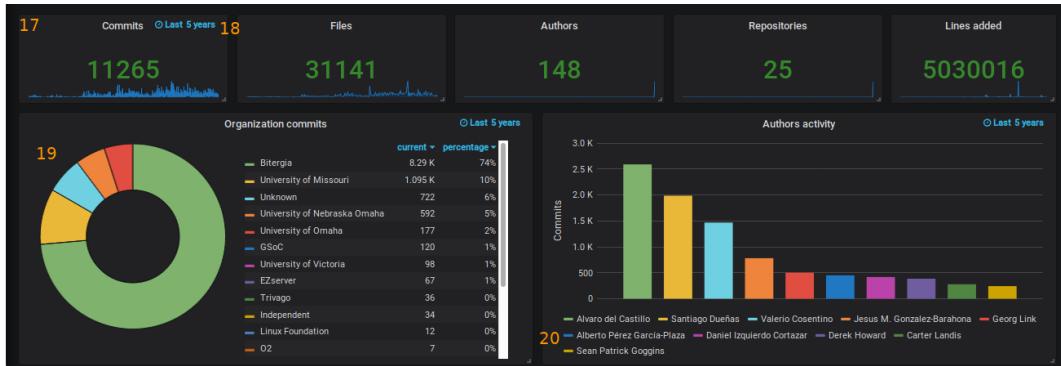


Figura 3.10: Interfaz de Grafana (central) y ejemplo de dashboard

3.3. Sprint 3 - Primer panel 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.

3.3.1. Configurando nuestro panel

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 (mencionado en la Sección 3.2.2: Configuración de índices de Elasticsearch).
- **Query:** Se puede utilizar una query de tipo Lucene¹⁷ para filtrar más concretamente lo

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

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 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 3.10). 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 3.10).

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.

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).

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

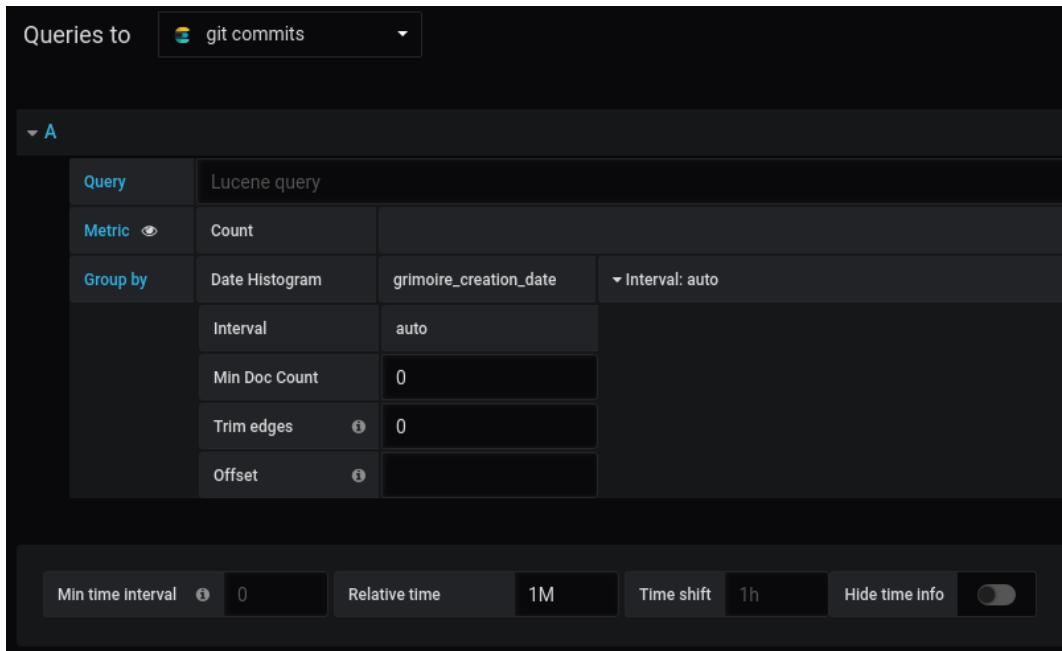


Figura 3.11: Ejemplo de query para panel de Grafana

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 3.13.

3.3.2. Plugins o complementos

Uno de los aspectos más distintivos de Grafana es el uso de los plugins¹⁹ creados y mantenidos por la comunidad. Un plugin, en este caso, es una simple extensión para la herramienta, a modo de nuevos paneles o aplicaciones integradas. Hay infinidad de complementos para ella: la mayoría son nuevos paneles, otros nos permiten la posibilidad de añadir nuevas fuentes de datos de BBDD no permitidas por defecto en la herramienta (Prometheus²⁰, Akumuli²¹, Stack-driver²²...) y finalmente tenemos las 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.

Instalar un plugin en Grafana es muy sencillo. Basta con elegir uno de entre su catálogo y

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

²⁰<https://prometheus.io/>

²¹<https://akumuli.org/>

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

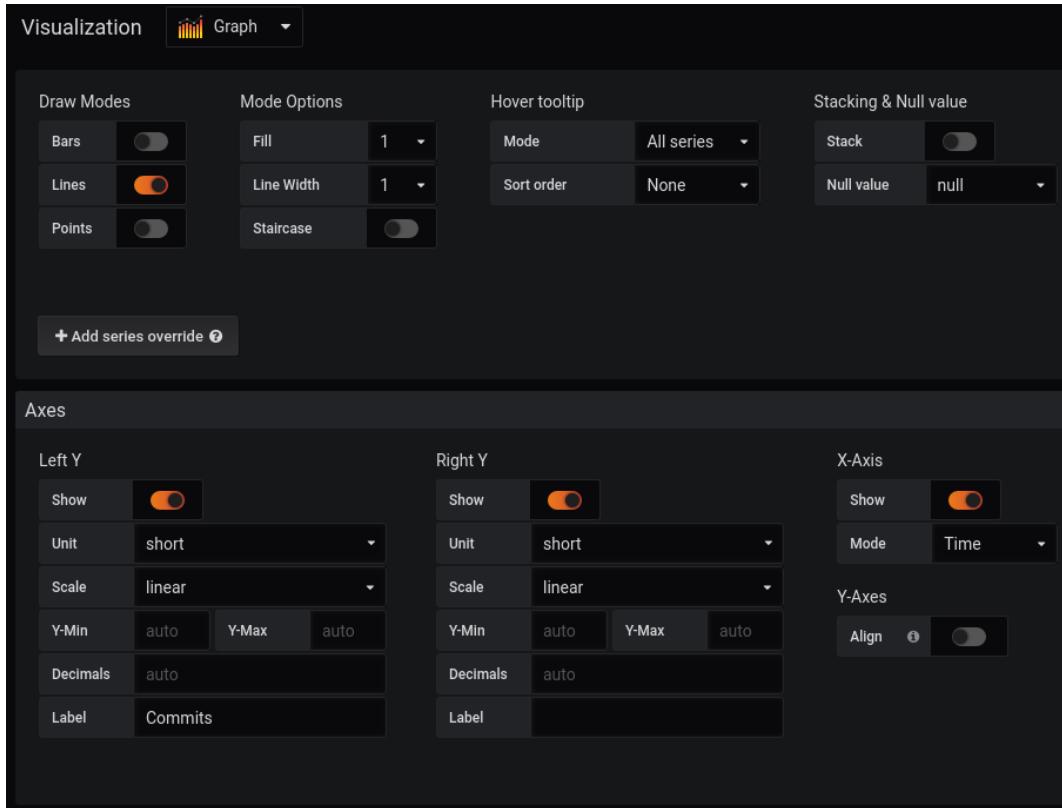


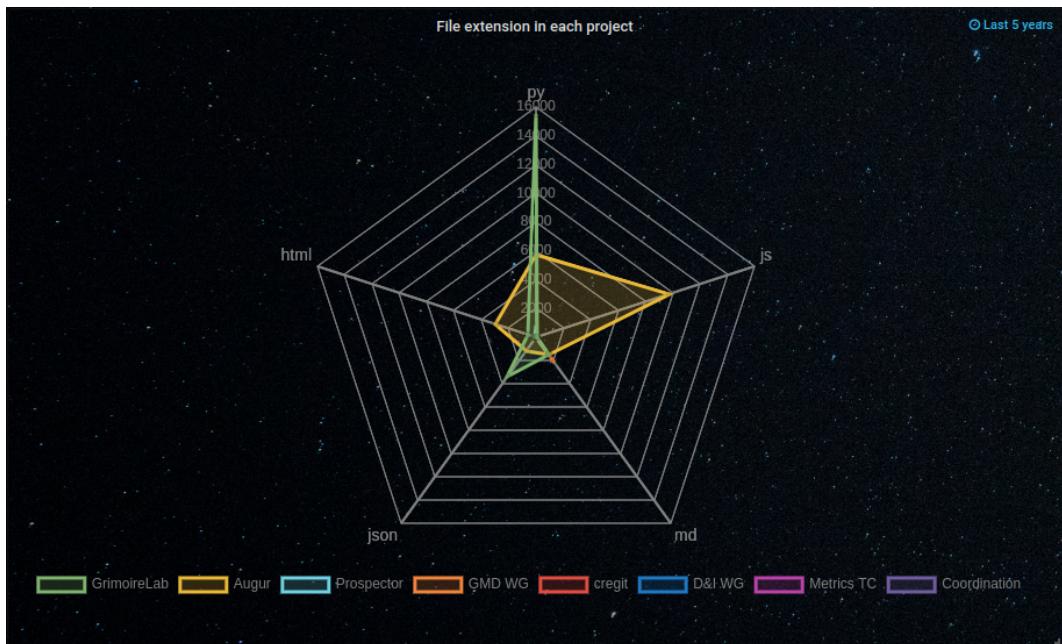
Figura 3.12: Ejemplo de configuración visual de panel tipo *Graph*

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 *Radar Graph Panel*, basta con escribir en un terminal lo siguiente (con el servidor de Grafana ya lanzado):

```
grafana-cli plugins install snuids-radar-panel
```

Desde este sprint en adelante se han instalado y analizado diversos complementos, para finalmente quedarnos con un total de 20 plugins, que son los que nos han parecido más relevantes/útiles. En la Sección 4.2 (Plugins relevantes) se aborda este tema mucho más en profundidad, describiendo cada uno de los complementos probados.

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

Figura 3.13: Ejemplo final de *Graph Panel*Figura 3.14: Ejemplo del plugin *Radar Graph*

3.4. 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. Este *sprint* acabó convirtiéndose finalmente en parte de la demo del producto final, Sección 4.1 (Evaluación y demo de Grafana). Además, sirvió para poder realizar una comparación entre ambas herramientas (virtudes y defectos), la cual se aborda en la Sección 4.3.

3.5. SPRINT 5 - INTEGRACIÓN: CONSTRUCCIÓN DEL CONTENEDOR DE GRIMOIRELAB/GRAFANA

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 3.8).

3.5. Sprint 5 - Integración: construcción del 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).

3.5.1. Acercamiento a la construcción de la imagen

Pero para poder realizar todo esto, hubo que recurrir primeramente al manual de Docker Hub, para familiarizarnos con los comandos y archivos necesarios para construir nuestra imagen. Se utilizaron principalmente dos fuentes: la proporcionada por la documentación oficial²⁴, y un pequeño manual en GitHub²⁵. Leyendo ambos manuales queda claro que sí o sí necesitábamos usar el comando `docker built` 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...).

²⁴<https://docs.docker.com/engine/reference/commandline/build/>

²⁵<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 GitHub²⁶. 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).

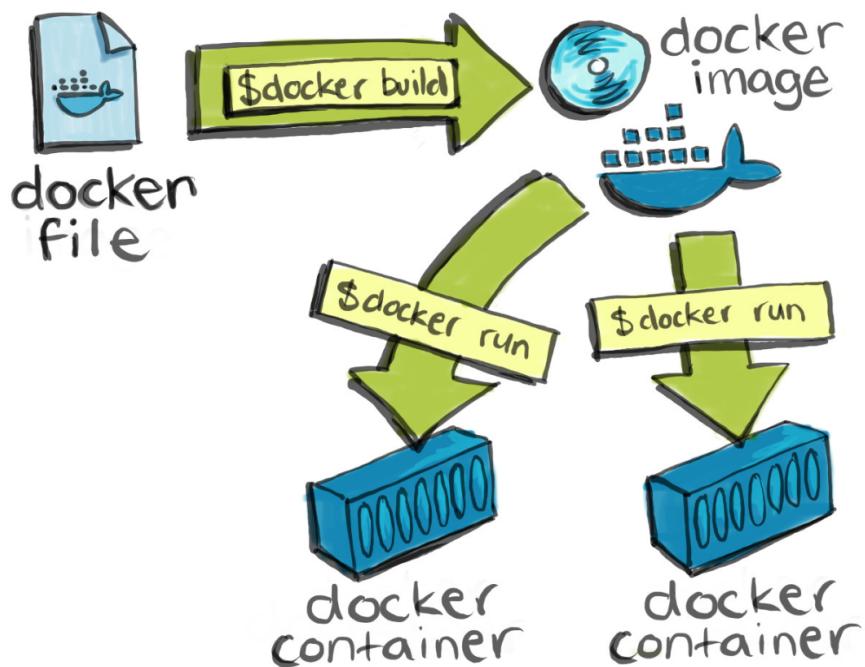


Figura 3.15: Funcionamiento de docker con Dockerfile

3.5.2. Creando un nuevo Dockerfile

Al abrir el fichero mencionado anteriormente, Dockerfile-full, nos encontramos con un conjunto de instrucciones y comandos, la mayor parte desconocidos hasta ahora. De todos los que se muestran, 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).

²⁶<https://github.com/chaoss/grimoirelab>

3.5. SPRINT 5 - INTEGRACIÓN: CONSTRUCCIÓN DEL CONTENEDOR DE GRIMOIRELAB/GRAFANA

- **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 desplegase 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 3.16.

```
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 3.16: Extracto del 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 3.3 (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 3.17.

3.5.3. 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 manual²⁷, pode-

²⁷<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 3.17: Extracto del 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 correctamente²⁸:

```

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 3.18. Tendremos entonces disponible la herramienta donde siempre, en `localhost:3000`.

3.6. 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 tan-

²⁸Si se lanza el servidor de Grafana localmente (`service grafana-server start`) antes que el contenedor, y no cerramos antes el puerto 3000, este no podrá ser “alcanzado” por el contenedor. Es necesario, por tanto, matar el proceso de Grafana para liberar el puerto, gracias a los comandos `ps -aux` y `sudo kill PID`.



```
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: 0aba9df5a1f0
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 ::1:3306           ::*:*
MariaDB started
Starting Kibiter
Waiting for Kibiter to start...
tcp        0      0 0.0.0.0:5601          0.0.0.0:*          LISTEN
Kibiter started
Starting Grafana
[ ok ] Starting Grafana Server...
Grafana started
```

Figura 3.18: Logs del nuevo contenedor de Grafana

to, 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.

3.6.1. 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 3.8). 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 aprovisionamiento²⁹ 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 caso, esta lista se corresponde al directorio local donde exportamos previamente los dashboards, del cual daremos su *path* (ver Figura 3.19). 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 3.20).

```

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 3.19: Ejemplo de fichero de provisionamiento para importación de dashboards

²⁹<https://grafana.com/docs/administration/provisioning>

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 script³⁰, el cual debemos añadir a nuestro *Entry point* 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 3.20).

3.6.2. 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

³⁰<https://rmoff.net/2017/08/08/simple-export/import-of-data-sources-in-grafana/>

imagen (ver Figura 3.20), al iniciar la herramienta en el contenedor todo estará tal cual lo dejamos. Es un método toscos 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).

```

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 michaeldmoore-announcer-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-datable-panel
96 RUN sudo grafana-cli plugins install jdbranham-diagram-panel
97 RUN sudo grafana-cli plugins install laronna-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 michaeldmoore-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 3.20: Extracto del nuevo Dockerfile (utilizando método de exp/imp abrupta)

3.7. Sprint 7 - Imagen final: alojamiento en la nube y contenedores múltiples

Tras lo explicado en la sección anterior, el proyecto entraba en su fase final de desarrollo. Quedaban aún en el aire dos ideas, que son las que se llevaron a cabo durante este sprint:

- Pasar de tener un contenedor ubicado localmente a tenerlo accesible de manera remota, desde la plataforma de Docker Hub.
- Posibilidad de analizar distintos proyectos de manera rápida y sencilla, ya sea con uno o con varios contenedores.

3.7.1. Preparación del contenedor final: alojamiento en servidor externo

Gracias a lo mostrado en la Sección 3.6 (Actualización del contenedor de Grafana), tenemos ya un contenedor de imagen completamente funcional. Lo único que falta es que nuestro docker pase de estar alojado localmente a estar en la nube. Para ello, debemos hacer nuevamente uso de la plataforma Docker Hub, y su respectiva documentación³¹.

El proceso es muy sencillo, y consta de 3 partes:

1. Hacer *login* en la plataforma Docker Hub gracias al comando³²:

```
1 docker login --u=<username> --p=<password>
2
```

2. Dar un nombre apropiado al docker. Para ello, podemos ver las imágenes creadas con `docker images`. Identificamos el identificador (ID) de la imagen que queremos renombrar adecuadamente, y lo hacemos con el comando `docker tag`. En mi caso concreto, quedaría:

```
1 docker tag TAG_ID onac8/grafana-grimoirelab:latest
2
```

3. Subir la imagen renombrada a la plataforma con:

```
1 docker push onac8/grafana-grimoirelab:latest
2
```

Si todo funcionó correctamente, deberían mostrarse diferentes logs mostrando como se sube la imagen poco a poco, y un mensaje final mostrando el identificador del nuevo docker. A partir de aquí, el contenedor quedará accesible desde cualquier dispositivo que permita instalar la

³¹<https://docs.docker.com/engine/reference/commandline/push/>

³²Es posible que la primera vez se obtenga un mensaje similar a “WARNING: login credentials saved in...”. Esto solo es una advertencia para que sepamos que nuestras credenciales se guardan localmente

herramienta de Docker Hub. Además, los comandos necesarios para su despliegue serán los mismos utilizados hasta ahora, aunque con un pequeño matiz: la primera vez que se ejecute el contenedor, Docker Hub lo descargará de su servidor y alojará localmente en nuestro dispositivo, por lo que pueden aparecer logs diferentes a los mostrados en las ejecuciones posteriores.

3.7.2. Despliegue con distintos contenedores de imagen

Es en esta subsección se aborda la implementación pensada para posibilitar el análisis rápido y sencillo de varios proyectos. Para ello, debemos cambiar el proyecto que debe analizar GrimoireLab. Esto se consigue modificando un fichero llamado `projects.json`³³.

El fichero en cuestión (ubicado en `grimoirelab/docker/`) otorga a GrimoireLab las direcciones web desde las que recolectar los diferentes datos, esto es, los enlaces a los distintos repositorios y listas de correo de un proyecto. Por tanto, si queremos trabajar con dos o más proyectos distintos, debemos “mapear” este archivo `.json` para conseguir que, al desplegar el contenedor, la herramienta de Docker Hub escoga el fichero que nosotros queramos (que será un nuevo documento con diferentes enlaces a repositorios). Nuestras sentencias para el despliegue quedarían así:

```

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 -v $(pwd)/myprojects.json:/projects.json \
5 -t grimoirelab/grafana

```

Con el comando `[...]/myprojects.json:/projects.json` utilizamos nuestro fichero particular (llamado en este ejemplo `myprojects.json`) para un proyecto concreto, como pueden ser los repositorios oficiales del proyecto Grafana³⁴.

³³Sección `grimoirelab/installed` del manual: <https://github.com/chaoss/grimoirelab/tree/master/docker>

³⁴<https://github.com/grafana/grafana>

```

1   {
2     "GrimoireLab": {
3       "git": [
4         "https://github.com/chaoss/grimoirelab",
5         "https://github.com/chaoss/grimoirelab-bestiary",
6         "https://github.com/chaoss/grimoirelab-ceres",
7         "https://github.com/chaoss/grimoirelab-elk",
8         "https://github.com/chaoss/grimoirelab-hatstall",
9         "https://github.com/chaoss/grimoirelab-kidash",
10        "https://github.com/chaoss/grimoirelab-kingarthur",
11        "https://github.com/chaoss/grimoirelab-manuscripts",
12        "https://github.com/chaoss/grimoirelab-sirmordred",
13        "https://github.com/chaoss/grimoirelab-perceval",
14        "https://github.com/chaoss/grimoirelab-perceval-mozilla",
15        "https://github.com/chaoss/grimoirelab-perceval-opnfv",
16        "https://github.com/chaoss/grimoirelab-perceval-puppet",
17        "https://github.com/chaoss/grimoirelab-sortinghat",
18        "https://github.com/chaoss/grimoirelab-sigils",
19        "https://github.com/chaoss/grimoirelab-tutorial"
20      ],
21      "github": [
22        "https://github.com/chaoss/grimoirelab",
23        "https://github.com/chaoss/grimoirelab-bestiary",
24        "https://github.com/chaoss/grimoirelab-ceres",

```

Figura 3.21: Extracto del archivo *projects.json*

Despliegue con distintas bases de datos

Una pequeña mejora a la implementación explicada anteriormente, es la posibilidad de tener bases de datos persistentes³⁵. Para ello, podemos guardar localmente todos y cada uno de los índices enriquecidos de Elasticsearch:

```

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 -v $(pwd)/es-data:/var/lib/elasticsearch \
5 -v $(pwd)/myprojects.json:/projects.json \
6 -t grimoirelab/grafana

```

Con el comando [...] (pwd) /es-data:/var/lib/elasticsearch, se consigue alojar la base de datos en un directorio cualquiera (en este caso *es-data*). Esta implementación permite desplegar un contenedor de la manera más rápida posible sin perder datos.

³⁵Sección *grimoirelab/full* del manual: <https://github.com/chaoss/grimoirelab/tree/master/docker>

3.8. Arquitectura final del proyecto

A continuación, se presenta la estructura general que obtuvo finalmente este Trabajo de Fin de Grado. Tal y como se ha comentado anteriormente (ver Sección 3: Metodología Scrum), el proyecto está dividido en los diferentes *scrums* realizados. Por esa razón, no se tuvo un diseño general hasta que el proyecto entró en su fase final de desarrollo, ya que se fueron añadiendo diferentes aspectos a lo largo del tiempo. En la Figura 3.22 se puede observar de un vistazo el diagrama final del trabajo. Se estructura en cuatro partes:

- Bloques azules: Docker Hub, 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.

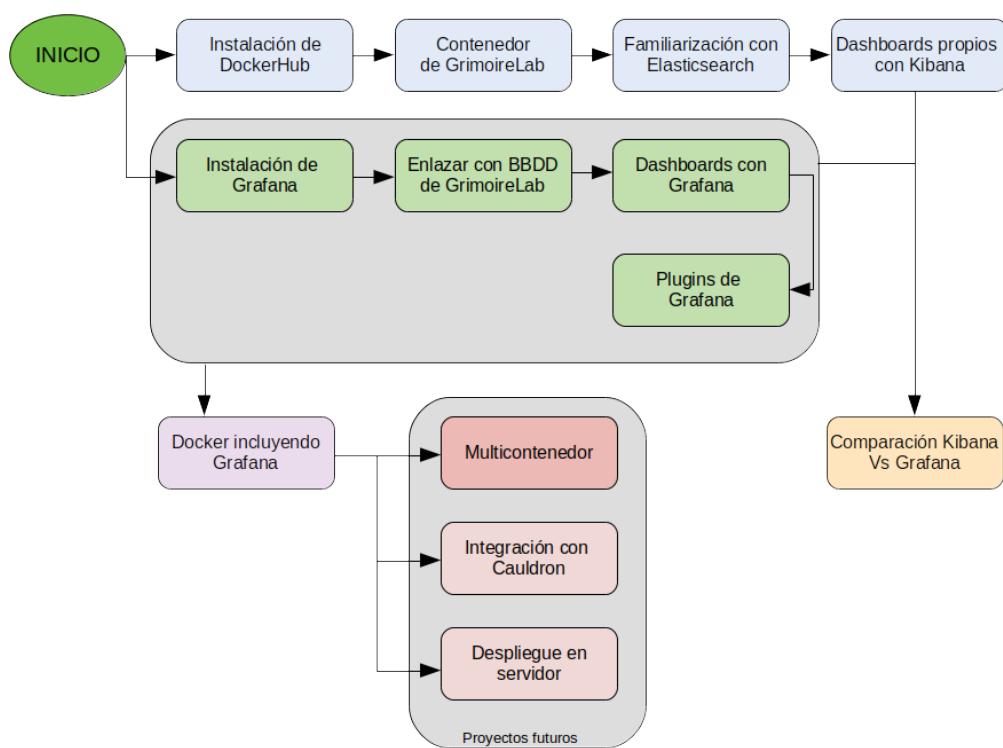


Figura 3.22: Diagrama de bloques del proyecto

Capítulo 4

Resultados

En esta sección se abordarán los resultados obtenidos tras haber realizado los diferentes sprints mencionados en el capítulo anterior. Además, se presentará el producto final, con diversos ejemplos visuales, plugins añadidos, etc. También se hará una comparación entre las dos herramientas predominantes del proyecto: Grafana y Kibana. Por último, se plantearán ideas futuras para una eventual expansión del proyecto, mencionando la versatilidad en el uso de los multicontenedores, o la integración de esta herramienta en otras de mayor envergadura.

4.1. Evaluación y demo de Grafana

Para una correcta evaluación de nuestra herramienta, hemos de comenzar por el principio, por su despliegue. Para este ejemplo, la demostración se hará sobre el proyecto CHAOSS, sin bases de datos persistentes. Por lo tanto:

```
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
```

En esta demostración, el contenedor `grimoirelab/grafana` (creado expresamente para este proyecto) lo tenemos instalado localmente. De no ser así, tal y como se explicó en la Sección 3.7.1 (alojamiento en la nube), la imagen utilizada se encuentra en la plataforma de Docker Hub a disposición de cualquier usuario. Por lo tanto, al ejecutarse por primera vez, la imagen correspondiente se descargará del servidor antes de proceder a su lanzamiento.

Cada argumento utilizado en la sentencia tiene un propósito: en las primeras dos líneas ordenamos “lanzar” el contenedor, indicando la IP y el puerto ((*flag -p*) de las cuatro herramientas principales (GrimoireLab/Kibana, API de Elasticsearch, MariaDB y Grafana). Además, se indica dos veces cada puerto, debido a que uno es que se utiliza en la máquina local y otro el que se utilizará dentro del contenedor (suele ser el mismo). El argumento *-v* nos indica que unidad o volumen se “montará” en el contenedor, en este caso la API Key de GitHub alojada en el fichero `credentials.cfg`. Por último indicamos el nombre o *tag* de la imagen a ejecutar (*flag -t*). Todo esto queda explicado también en la Sección 3.1.1.

El proyecto por defecto utilizado es el que ofrece GrimoireLab (analizando los repositorios y listas del proyecto en sí), por lo que no es necesario “mapear” el fichero `projects.json`, documento utilizado para decidir qué proyecto estudiar (todo esto queda reflejado y explicado en la Sección 3.7.2 de multicontenedores). Una muestra (visual) del despliegue utilizando otros proyectos y bases de datos persistentes puede encontrarse en el Apéndice (Capítulo A), al final de esta memoria.

Al desplegar Grafana en un navegador y autenticarnos, veremos un *Home* similar al mostrado en la Figura 4.1.

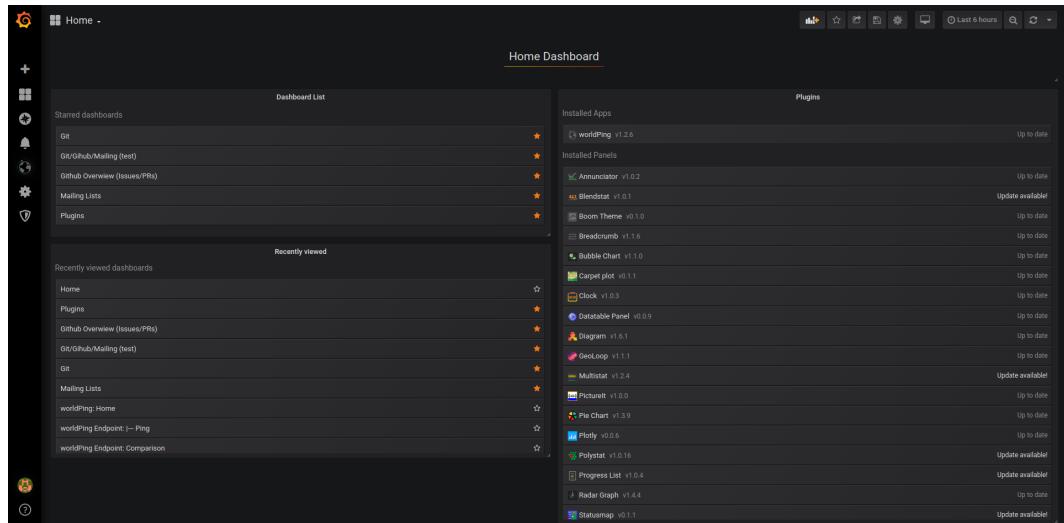


Figura 4.1: Página *Home* final de Grafana

Aquí se nos presentan dos paneles centrales principales: a la izquierda, los dashboards creados, llamados *Starred Dashboards*, y a la derecha los plugins/apps instalados, de los cuales se habla posteriormente en la Sección 4.2. Cada uno de los dashboards propios cumple un come-

tido:

- **Git:** Visión general del índice GIT de la base de datos. En este dashboard se muestran tres tipos de paneles: varios de ellos intentan imitar a los vistos en Kibana (*Git Overview*), a modo de comparación de herramientas, mezclándose además con paneles propios de la aplicación, dándole un aspecto más fresco. Por último se añade algún plugin, haciendo este dashboard uno de los más completos.
- **Mailing Lists:** Similar al anterior, tenemos aquí una recreación casi idéntica del dashboard visto en Kibana (con el mismo nombre), solo que añadiendo dos aspectos diferenciadores: uso de diferentes escalas/rangos de tiempo por cada panel y uso de plugins.
- **Plugins:** En este dashboard se muestran las diferentes y numerosas extensiones de Grafana, mostrando las que se han considerado como más útiles o importantes, con paneles que utilizan diferentes índices de la base de datos.
- **Git/Gitgub/Mailing (pruebas):** Utilizado a modo de experimentación. Si los paneles creados aquí cumplen las expectativas, se realiza una copia que se incluye en los dashboards principales.

4.1.1. Demostración de dashboards

A continuación se recopilan diferentes capturas de cada dashboard mencionado anteriormente. Se han combinado diferentes conjuntos de paneles nativos y plugins, para darle más diversidad al resultado final.

Ejemplos de paneles nativos son los mostrados por la Figura 4.2, donde se pueden observar:

- Gráficos de líneas: se encuentran ubicados en la parte superior izquierda, y en este caso describen la cantidad de *commits* a lo largo del tiempo (cada gráfica correspondiendo a un lapso temporal diferente).
- Mapas de calor: se puede observar en la parte derecha de la imagen. Este panel corresponde nuevamente al número de *commits* a lo largo del tiempo, pero permite observar la frecuencia a la que se realizaban dependiendo de la fecha.

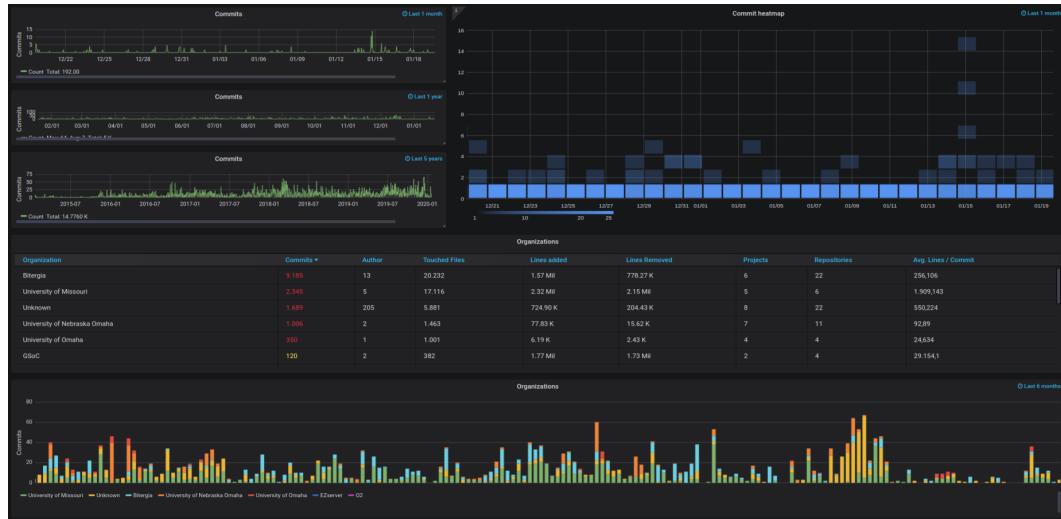


Figura 4.2: Git Dashboard - Mezcla de paneles nativos y plugins

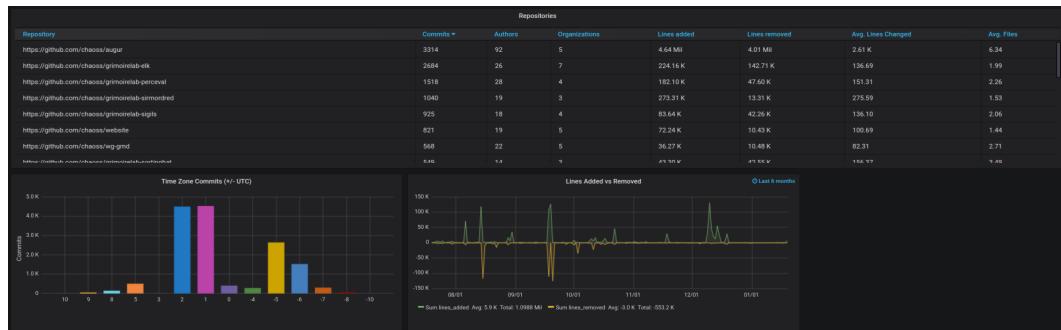


Figura 4.3: Git Dashboard - Recreación de parte de Git Overview de Kibana

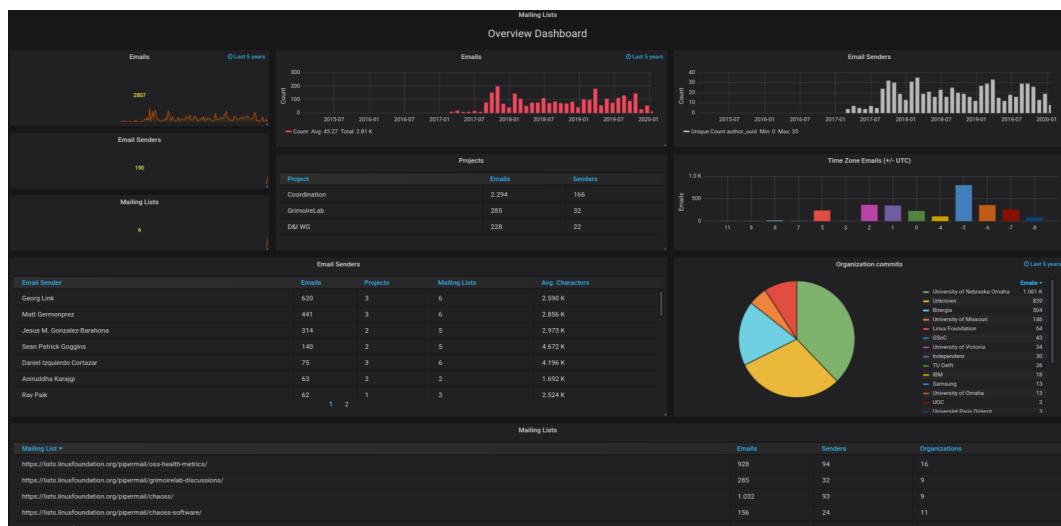
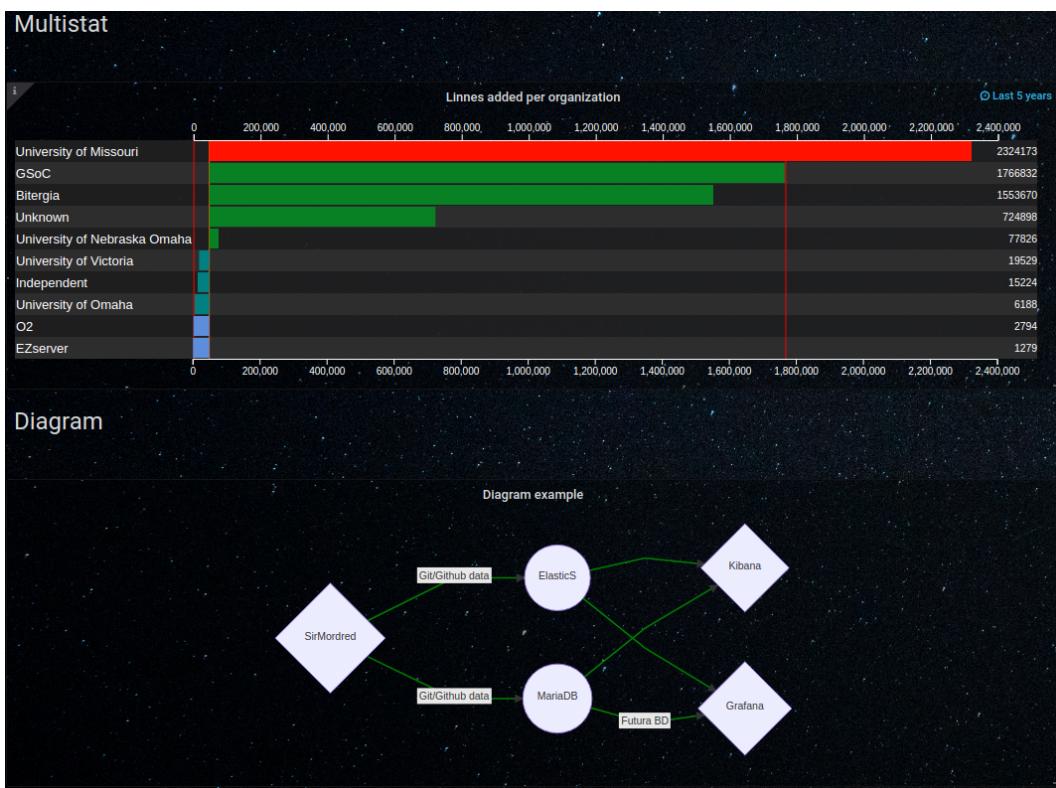


Figura 4.4: Mailing Lists - Recreación parcial del dashboard de Kibana

Figura 4.5: Extracto del dashboard *Plugins*

- Tablas: en la parte central de la figura se encuentra una pequeña tabla, la cual muestra el número de *commits* de toda la base de datos agrupados por *Organization*, de manera descendente. Además, posee un *threshold* para indicar las organizaciones que superen el centenar de contribuciones. Gráficos de barras agrupados: tal y como su nombre indica, permite agrupar en una misma barra varios parámetros. En este caso agrupa varias organizaciones, siempre y cuando en ese instante temporal haya contribuido con algún *commit* al proyecto específico.

Por otro lado tenemos también los diferentes plugins, encontrándonos por ejemplo el típico panel *Pie Chart* en la Figura 4.4. Aún así, en relación a este aspecto y también al dashboard *Plugins*, comentar que se aborda en profundidad en la sección siguiente. Además, la colección total de las capturas de este capítulo se encuentra en el Apéndice (Capítulo A) de esta memoria.

4.2. Plugins relevantes

Tal y como se abordaba en la Sección 3.3.2, el uso de plugins¹ es uno de los aspectos más distintivos de Grafana. Un plugin, en este caso, es una simple extensión para la herramienta, a modo de nuevos paneles o aplicaciones integradas. Hay infinidad de complementos para ella: la mayoría son nuevos paneles, otros nos permiten la posibilidad de añadir nuevas fuentes de datos de BBDD no permitidas por defecto en la herramienta (Prometheus², Akumuli³, Stackdriver⁴...) y finalmente tenemos las apps complementarias a Grafana, que añaden funcionalidad adicional.

En la Figura 4.1, se puede observar que desde la página principal de Grafana podemos ver los diferentes plugins/apps utilizados (parte derecha del menú *Home*). Al seleccionar cada uno de ellos podemos ver su manual de uso/instalación, así como diferentes ejemplos ideados por su creador. Para su uso/instalación, nuevamente en la Sección 3.3.2 se encuentra un pequeño manual genérico para el uso de dichas extensiones.

Para este proyecto se han instalado, analizado y probado multitud de complementos a lo largo de los diferentes sprints, para finalmente quedarnos con un total de **20 plugins**, que son los que nos han parecido más relevantes/útiles. La mayoría de ellos se encuentran disponibles en el dashboard *Plugins*. Y a continuación, se comenta para qué sirven cada uno de ellos:

- **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* (coordenadas) 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.

¹<https://grafana.com/grafana/plugins?orderBy=weight&direction=asc>

²<https://prometheus.io/>

³<https://akumuli.org/>

⁴<https://cloud.google.com/stackdriver/>

- **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.
- **Datable 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*⁵, mediante el uso de la librería `mermaid.js`⁶.
- **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...).
- **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*⁷ o hilo de Ariadna tan utilizado en páginas webs o exploradores de archivos. Es una

⁵https://es.wikipedia.org/wiki/Diagrama_de_Gantt

⁶<https://mermaidjs.github.io/>

⁷https://en.wikipedia.org/wiki/Breadcrumb_navigation

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.js⁸. 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.ly⁹ de Javascript¹⁰.
- **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 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...).

⁸<https://www.chartjs.org/>

⁹<https://plot.ly/>

¹⁰<https://developer.mozilla.org/es/docs/Web/JavaScript>

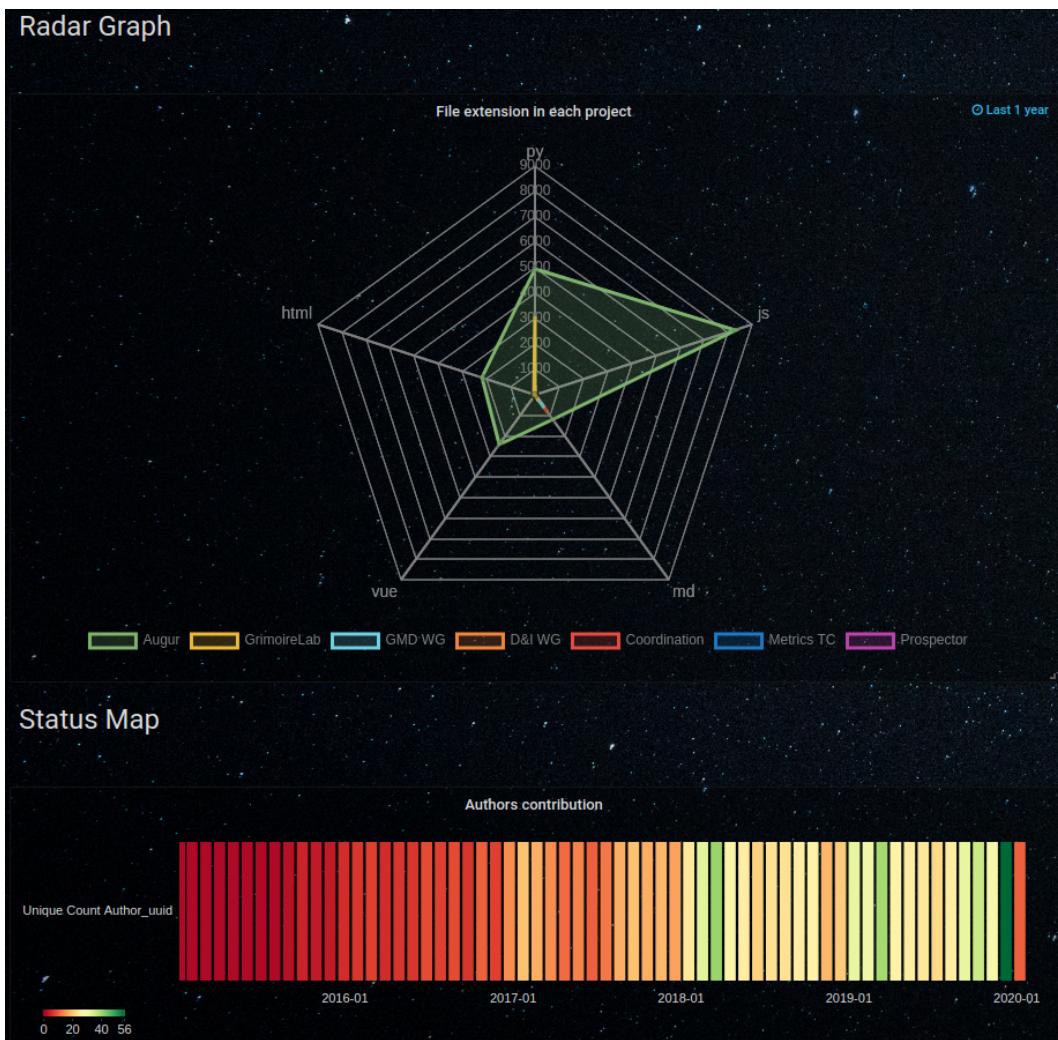


Figura 4.6: Extracto del dashboard *Plugins*

A lo largo de los diferentes dashboards se han introducido diferentes plugins, mezclándose con los paneles nativos de la aplicación, y aportando datos nuevos en comparación con lo visto en Kibana. La recopilación total de los diferentes plugins explicados, se encuentra nuevamente en el Apéndice (Capítulo A), al final del proyecto.

4.3. Comparación de herramientas: Kibana vs Grafana

Una de las principales motivaciones de este proyecto es la de utilizar una nueva herramienta de representación de datos y compararla con otra ya existente. En esta sección abordamos precisamente esto, comentando en profundidad las ventajas e inconvenientes que tiene cada una de ellas. Para ello, lo más conveniente es comentar sus diferencias y similitudes, agrupadas por

secciones:

Ejes de paneles

- Kibana: existe una completa manipulación de los ejes.
- Grafana: no podemos cambiar el nombre del eje X (de momento).

Manejo de documentos tipo *string*

- Kibana: trata de manera sencilla el conteo/recuento de los documentos tipo *string* usando simplemente la métrica `unique count`.
- Grafana: es un poco más tosco, ya que hay que aplicar un filtro a la búsqueda, permitiendo después realizar ese mismo `unique count`.

Intervalos de tiempo

- Kibana: únicamente se puede filtrar temporalmente por cada dashboard (commits de la última semana, issues del último mes, etc.)
- Grafana: filtrado general por cada dashboard. Además, en cada panel podemos indicar específicamente cual es su intervalo de análisis (último año, desde el inicio de los tiempos, último minuto...)

Selección de rangos temporales (gráficos)

- Kibana: si en gráficos temporales seleccionamos un rango concreto (efecto *zoom*), los cambia todo el dashboard al nuevo intervalo.
- Grafana: al seleccionar un nuevo rango, el cambio solo se aplica al panel en concreto, no a todo el dashboard.

Bases de datos de los dashboards

- Kibana: se puede utilizar una sola DB por cada dashboard.

- Grafana: dashboards con múltiples DB. En cada panel podemos elegir el índice sobre el que actuar, pudiendo tener diferentes paneles con índices de bases de datos distintas.

Aspecto visual

- Los datos básicos (nº total de proyectos, colaboradores, commits...) se representan como un único número plano.
- Paneles como Singlestat o Blendstat (plugin) permiten obtener un dato básico (o varios) con su respectiva gráfica temporal evolutiva de fondo.

Agrupación de paneles

- Kibana: cada cuadro de control es independiente de los demás y no es posible estructurarlos, solo redimensionarlos para obtener algo de organización
- Grafana: además de lo que ofrece Kibana, es posible agrupar diferentes paneles por *rows* o filas, y clasificarlos según lo que muestren.

Playlists

- Kibana: no hay manera de crear dashboards fluidos, es decir, que vayan cambiando a modo de “diapositivas”.
- Grafana: permite crear playlists¹¹ de dashboards y reproducirlos de manera cíclica. Muy útil si se van a representar datos de manera continua a modo de demo (TVs, reproductores...).

Plugins

- Kibana: no hay más paneles más allá de los nativos que proporciona la herramienta.
- Grafana: mayor cantidad de paneles por defecto y además infinidad de plugins creados y mantenidos por la comunidad (paneles, BBDD soportadas, apps...), aumentando enormemente las posibilidades de representación de datos.

¹¹<https://grafana.com/docs/reference/playlist/>

Alertas

- Kibana: sistema sencillo de refresco de dashboards, pero no cuenta con tanta alertas y monitorización como Grafana.
- Grafana: Sencillo sistema de alertas¹² para dashboards en tiempo real.

¹²<https://grafana.com/docs/alerting/rules/> <https://grafana.com/docs/alerting/notifications/>

Capítulo 5

Conclusiones

Tal y como se comentó en la Sección 1.2, en este proyecto se abordan 4 objetivos concretos: representación de datos, integración, multi-docker y comparación. Este capítulo será una aproximación para confirmar si finalmente se han conseguido estas metas y los objetivos adyacentes a estas. Además, se citarán los conceptos aprendidos y se propondrán ideas futuras para la ampliación de este proyecto.

5.1. Consecución de objetivos

Si echamos la vista atrás y profundizamos en las metas propuestas al principio de este proyecto, se puede comprobar que se han completado con éxito todos ellos:

- Se ha logrado la representación de datos con Grafana, tal y como se muestra en la Sección 3.4 (Dashboards funcionales).
- Se han usado un total de 20 plugins externos, que, juntados a los nativos de la aplicación, nos han permitido representar un total de 3 dashboards completos y completamente funcionales.
- Hemos conseguido unificar en un solo docker las diferentes herramientas utilizadas (ver Sección 3.5: Integración en un sólo contenedor). Esta *docker image*¹ se encuentra disponible en la plataforma de Docker Hub.

¹<https://hub.docker.com/r/onac8/grafana-grimoirelab>

- Además de unificar, se crearon contenedores adicionales para el despliegue de múltiples proyectos, explicado en el último Sprint (3.7) del proyecto.
- Se citan numerosas ventajas/desventajas y similitudes/diferencias entre las dos herramientas analizadas en la Sección 4.3 (Comparación de aplicaciones).

Por otra parte tenemos los objetivos secundarios. De nuevo, se han cumplido todas las metas propuestas, presentadas a continuación:

- Familiarización con BBDD no relacionales, y en particular con Elasticsearch (*queries*, introducción/extracción de documentos, etc.).
- Completo manejo de Kibana y Grafana (creación/manipulación de dashboards, sincronización con BBDD, etc.).
- Posibilidad de crear contenedores de imágenes simples gracias al uso de Docker Hub.

5.2. Aplicación de lo aprendido

Durante mi Grado he aprendido diferentes conceptos que me han servido directamente para poder realizar diversas tareas de este proyecto. En particular, se han aplicado conocimientos de las siguientes asignaturas:

1. **Sistemas Operativos**: Asignatura que permitió profundizar en el ámbito de la programación y en el manejo avanzado en sistemas operativos con kernel Unix². Resultó de mucha utilidad para el uso de scripts en dockers, comandos para mover/copiar archivos con autorizaciones especiales, cambiar permisos a directorios y/o ficheros ejecutables, etc.
2. **Servicios y Aplicaciones Telemáticas (SAT)**: Fue en esta asignatura donde se abordó por primera vez el uso de BBDD relaciones (SQLite³), ya que en las prácticas se trabajaban con páginas web dinámicas y programación orientada al lado del servidor.

²<https://es.wikipedia.org/wiki/Unix>

³<https://www.sqlite.org/index.html>

3. **Ingeniería de Sistemas de Información (ISI):** En esta materia se abordaron las bases de datos relacionales en profundidad (partiendo desde cero). Sirvió de ayuda para entender como se estructuran los diferentes proyectos usados para recopilar la información. Fue de gran ayuda, ya que es mucho más sencillo aprender BBDD NoSQL si se cuenta con una buena base en BBDD SQL.
4. **Desarrollo de Aplicaciones Telemáticas (DAT):** Conocimiento en profundidad esquematización y organización de páginas web, y de programación del lado del cliente. Dio una buena base para conseguir realizar unos dashboards más consistentes, mejor estructurados y vistosos para el usuario.

5.3. Lecciones aprendidas

En este apartado se expondrán los principales conocimientos aprendidos a lo largo de este Trabajo de Fin de Grado. Algunos de ellos cuelgan directamente de los objetivos principales y secundarios del proyecto, mencionados en la Sección 5.1:

1. Uso de sistemas de visualización de datos. Concretamente dos herramientas: Grafana y Kibana.
2. Análisis, recopilación, estudio y representación de datos de múltiples formas (centrándose en un aspecto más visual, orientarlo a un público concreto, etc.)
3. Manipulación de bases de datos (queries, inserción, reestructuración...). En particular las de tipo NoSQL, centrándose en Elasticsearch.
4. Importación/exportación de BBDD, en particular las de la herramienta Grafana, haciendo uso de su peculiar API.
5. Creación y manipulación de contenedores de imágenes complejos y completamente funcionales, gracias a Docker Hub, la herramienta líder en su sector.
6. Integración de aplicaciones: permitir que dos o más herramientas trabajen conjuntamente, comunicándose sin fallos.

7. Permitir el despliegue web de este conjunto de herramientas integradas, gracias nuevamente al servicio Docker Hub.
8. Posibilidad de trabajar en un proyecto de un tamaño considerable (CHAOSS), y el más grande hasta el momento para mi.
9. Una buena perspectiva en cuanto a trabajar en grupo: comunicación con mi tutor, feedback mutuo, discusión de ideas futuras...
10. Organización y estructuración temporal de un proyecto de gran envergadura para mi, cumpliendo con los plazos establecidos.
11. Aprendizaje de LaTeX, uno de los compositores de textos profesionales más utilizados.

5.4. Trabajos futuros

Uno de los puntos fuertes de este proyecto es la gran cantidad de posibilidades de ampliación que ofrece. Nos centraremos en este caso en tres de ellas:

- Desplegar esta herramienta en la nube, en algún servidor de internet. Para ello, el mejor ejemplo que se nos ocurrió a mi tutor y a mi, fue el de la posibilidad de integrar esta aplicación en un proyecto de mayor envergadura, que ya estuviese alojado en la web. Este proyecto es CAULDRON.
- Modificar los índices enriquecidos de la base de datos para que cuenten con documentos que hagan funcionar las diferentes Apps de Grafana (Worldmap, Worldping...).
- Creación de un sistema que nos permita exportar/importar las diferentes configuraciones de Grafana (usuarios, dashboards...) de una manera más sencilla (ver Secciones 3.6.1 y 3.6.2. Para ello se puede hacer uso de la API de la herramienta y crear diferentes scripts dependiendo de lo que se quiera exportar/importar en un nuevo contenedor o localmente.

5.5. Planificación temporal

Debido a que este proyecto sigue un modelo adaptado de la metodología Scrum, se han ido añadiendo nuevas ideas a lo largo del tiempo. La mayor parte del proyecto coincidió durante

mi último curso académico, mientras que los últimos meses se solapó con mi trabajo actual. En total, el proyecto ha durado unos 12 meses, con períodos de mayor contribución y con etapas más pausadas (teniendo en cuenta que yo estudiaba y trabajaba a la vez). 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: un trabajo de integración y otro de realidad virtual. Finalmente me decanté por la actual, 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 iniciales propuestos, así como los objetivos secundarios adyacentes.
- 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.

Apéndice A

Recopilación de capturas

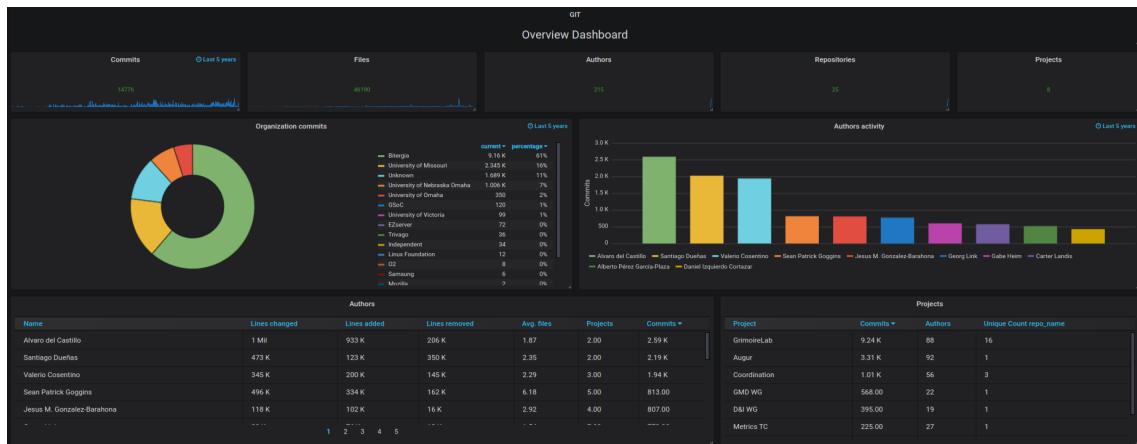


Figura A.1: Git Dashboard - Mezcla de paneles nativos y plugins

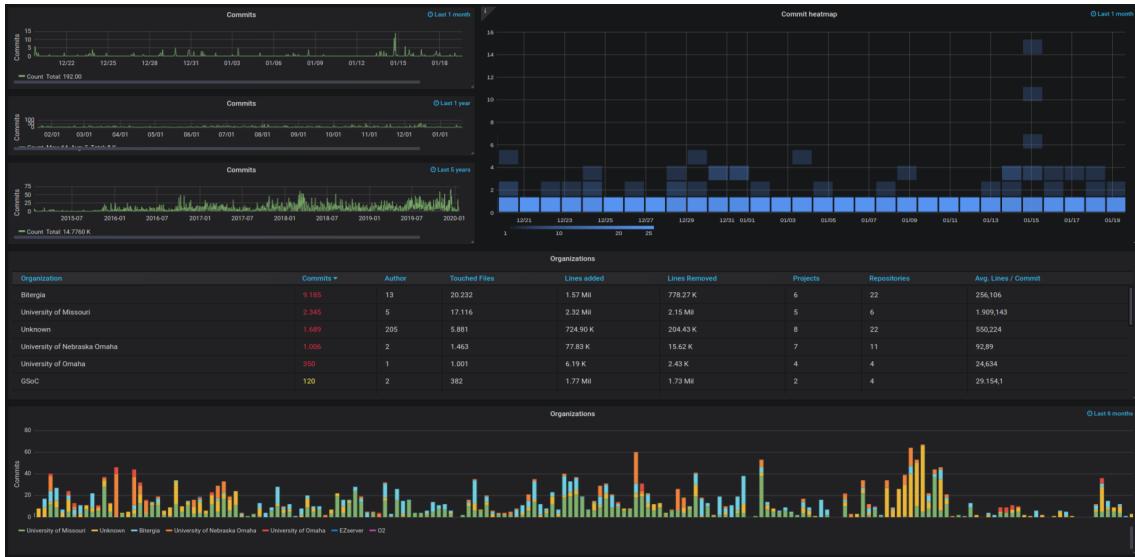


Figura A.2: Git Dashboard - Mezcla de paneles nativos y plugins (2)

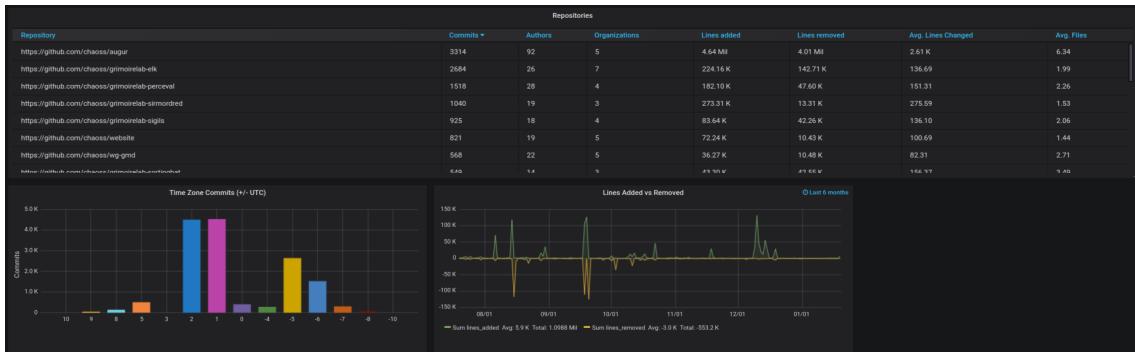


Figura A.3: Git Dashboard - Recreación de parte de Git Overview de Kibana

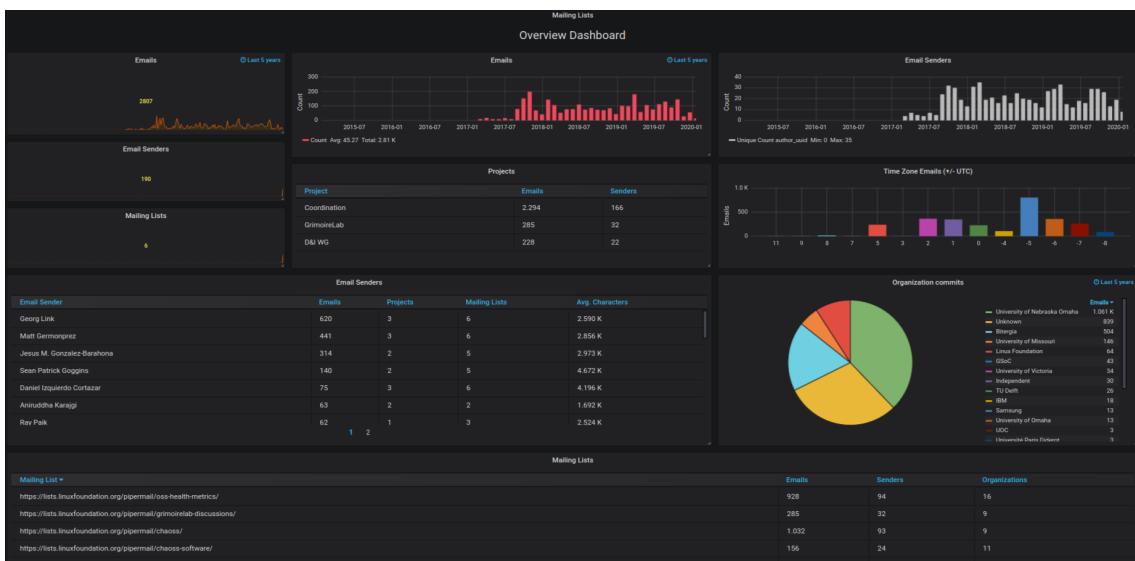


Figura A.4: Mailing Lists - Recreación parcial del dashboard de Kibana

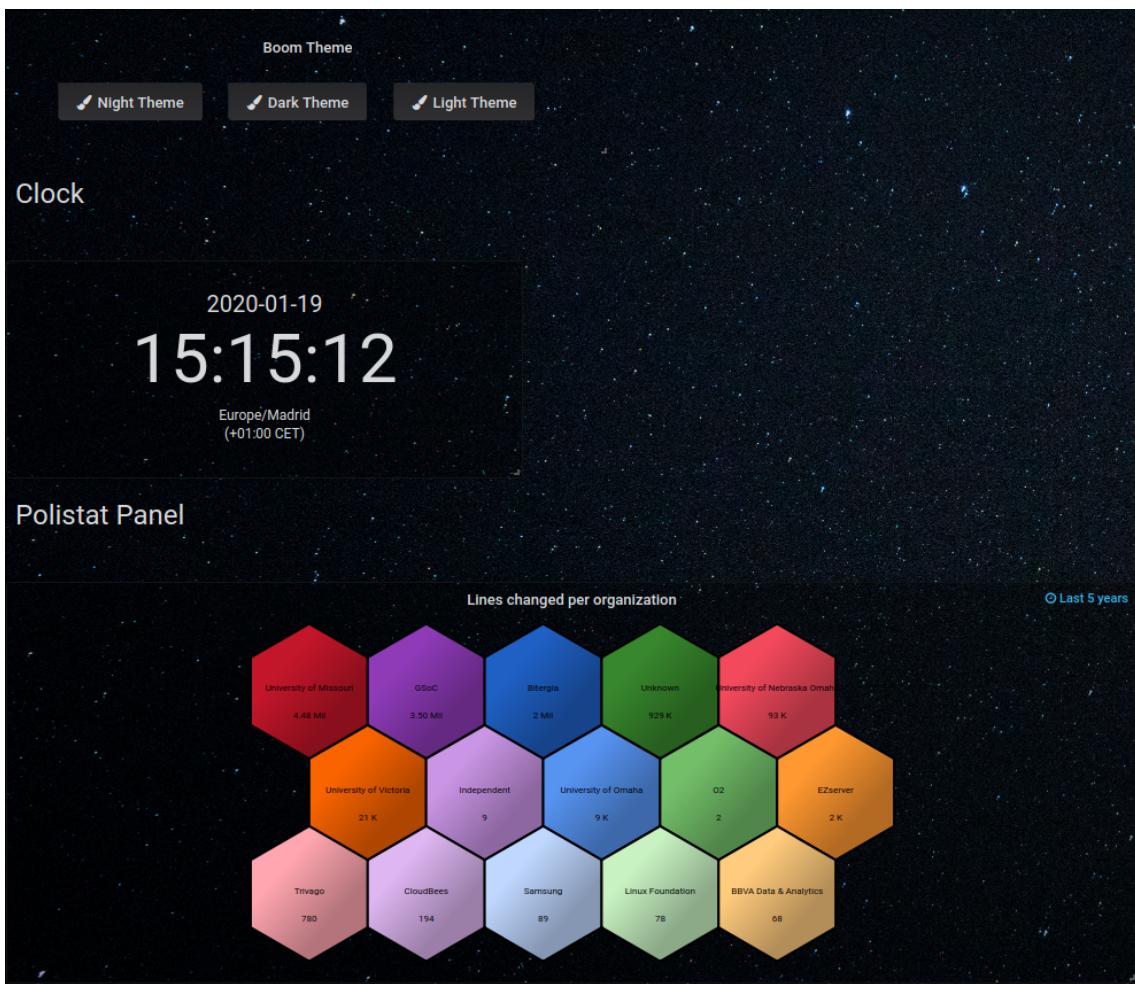
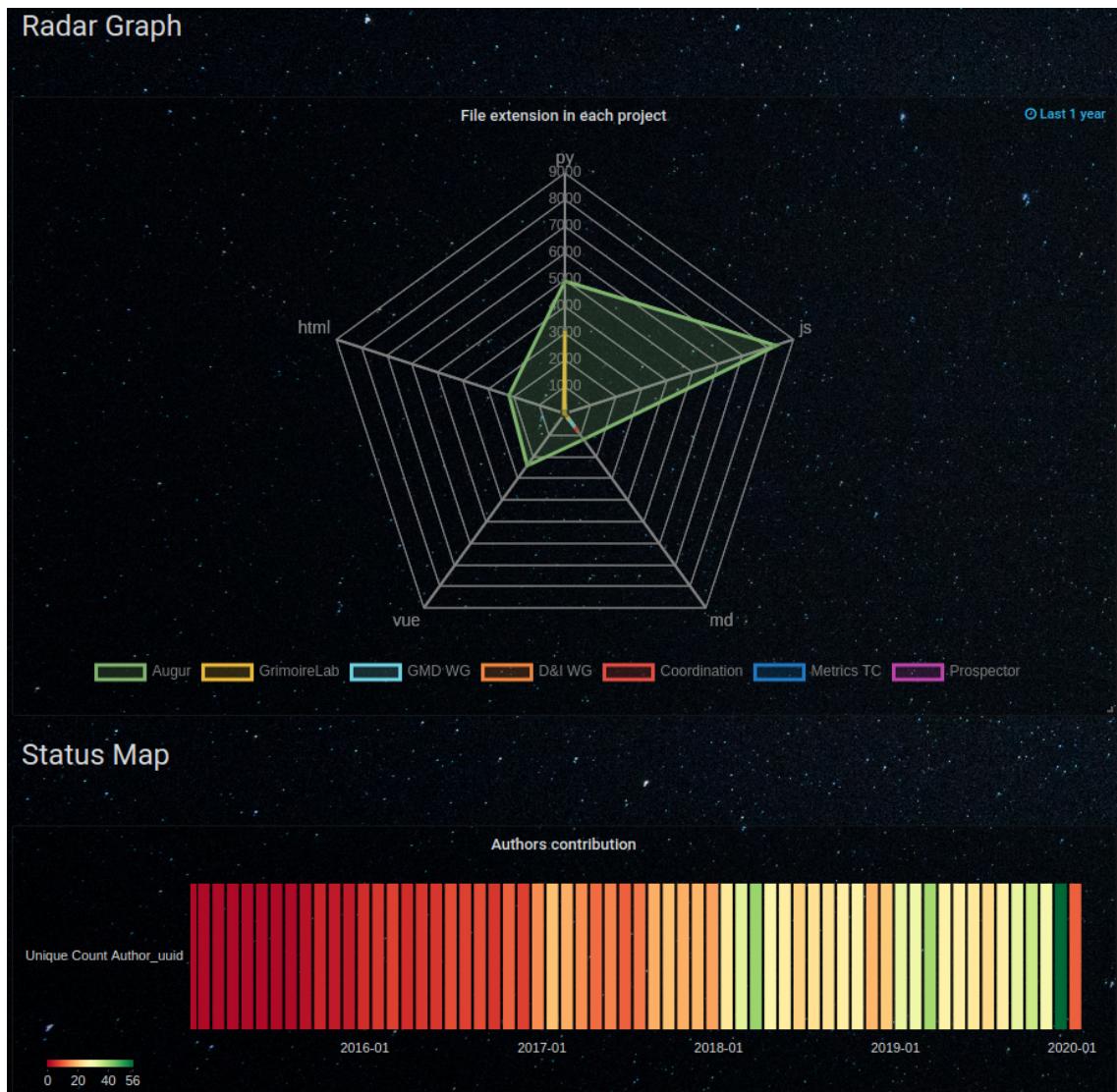


Figura A.5: Extracto del dashboard *Plugins*

Figura A.6: Extracto del dashboard *Plugins* (2)

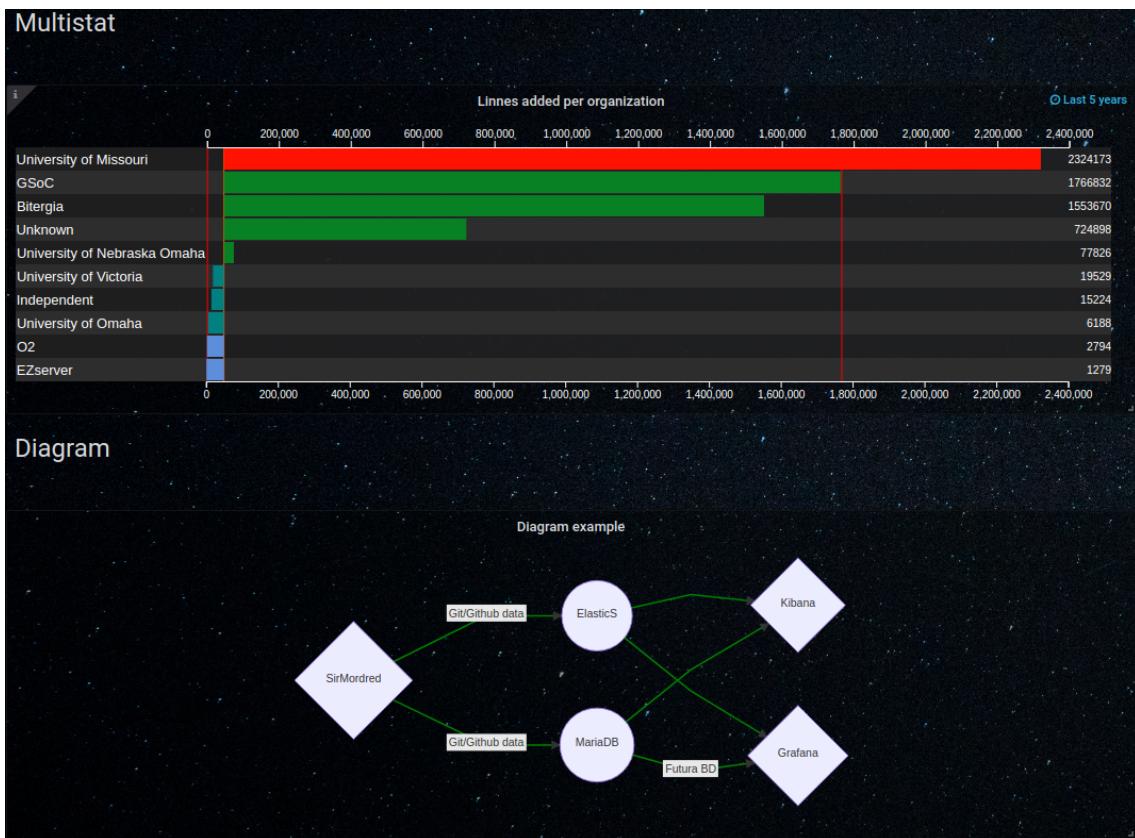


Figura A.7: Extracto del dashboard *Plugins* (3)

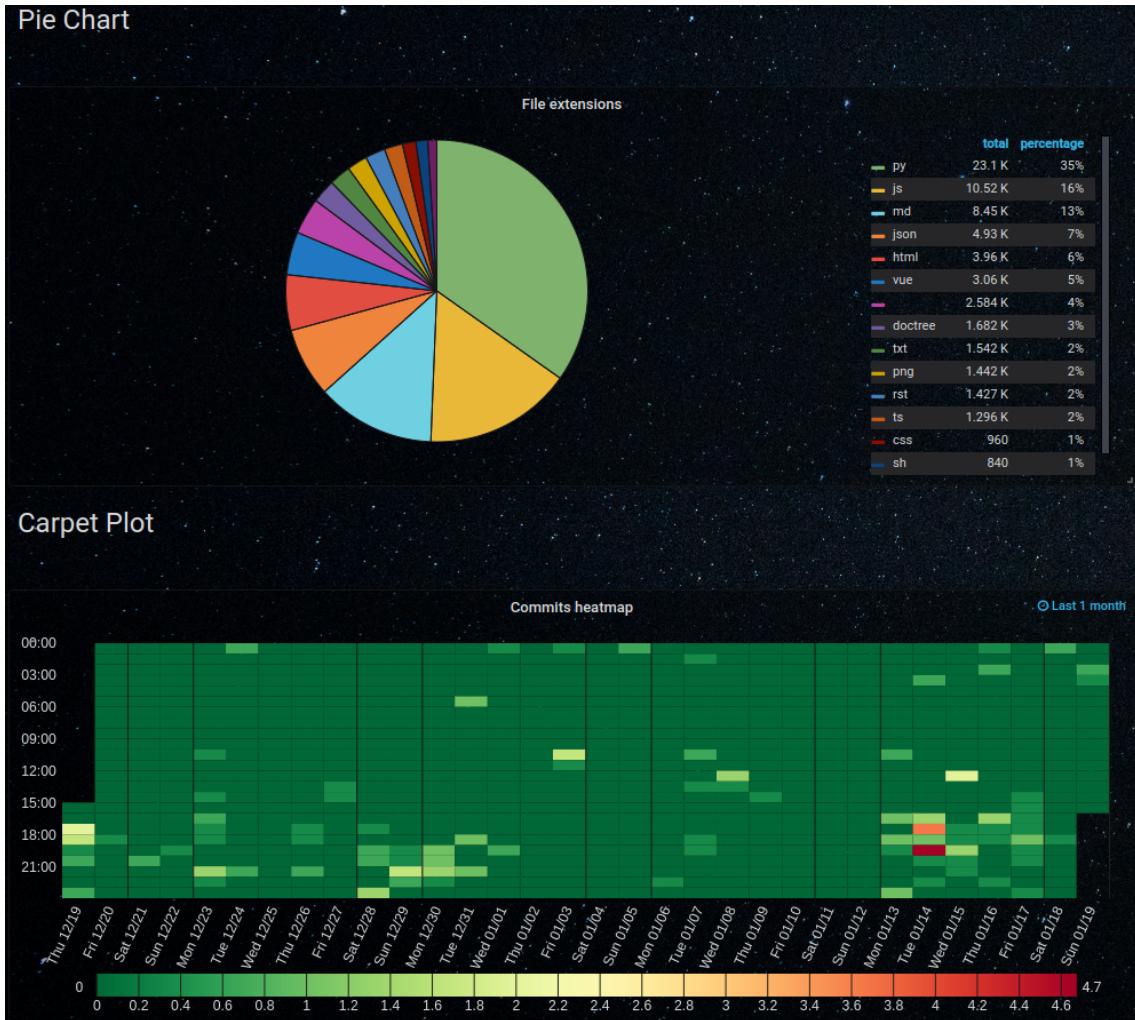


Figura A.8: Extracto del dashboard *Plugins* (4)

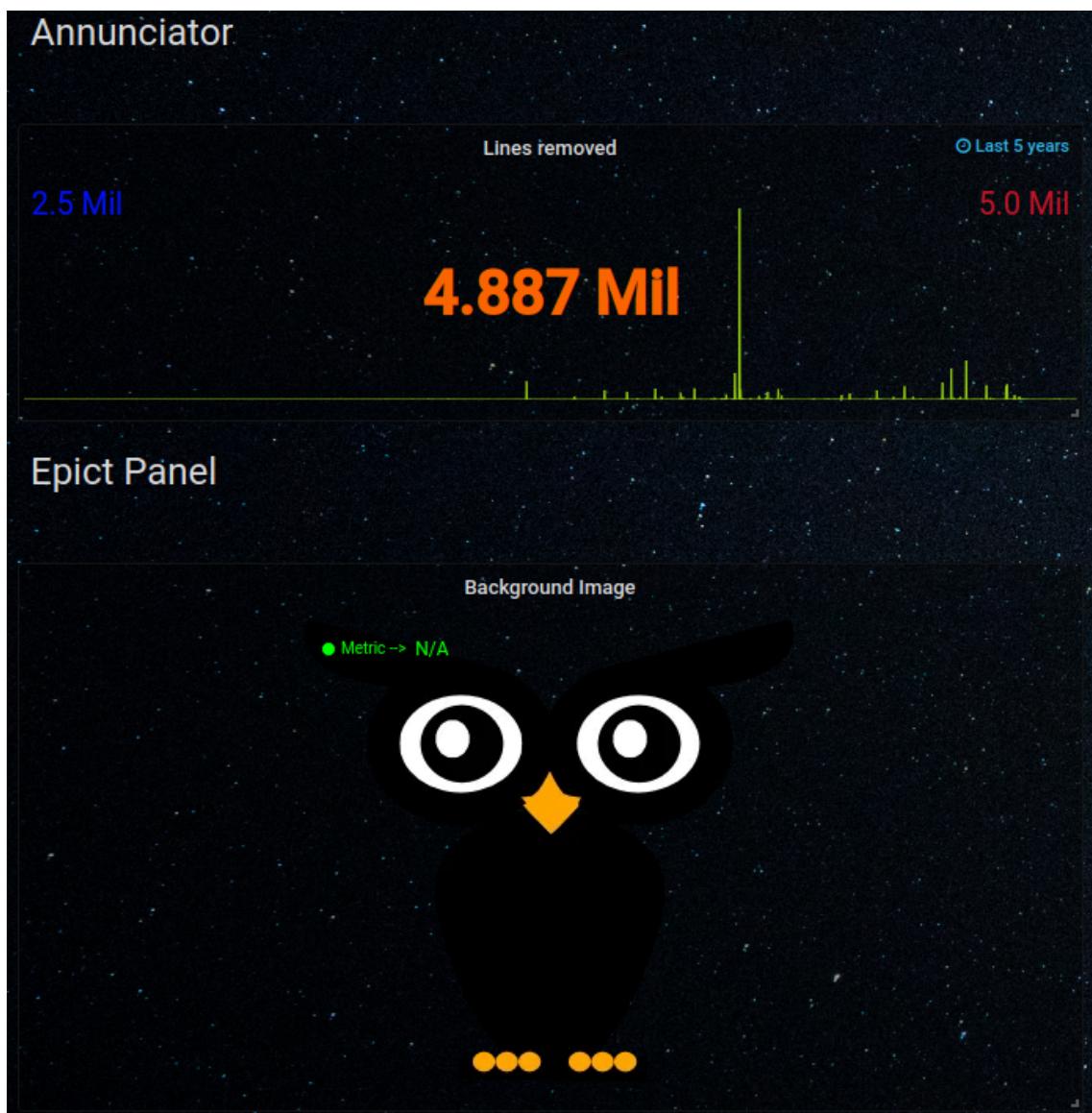
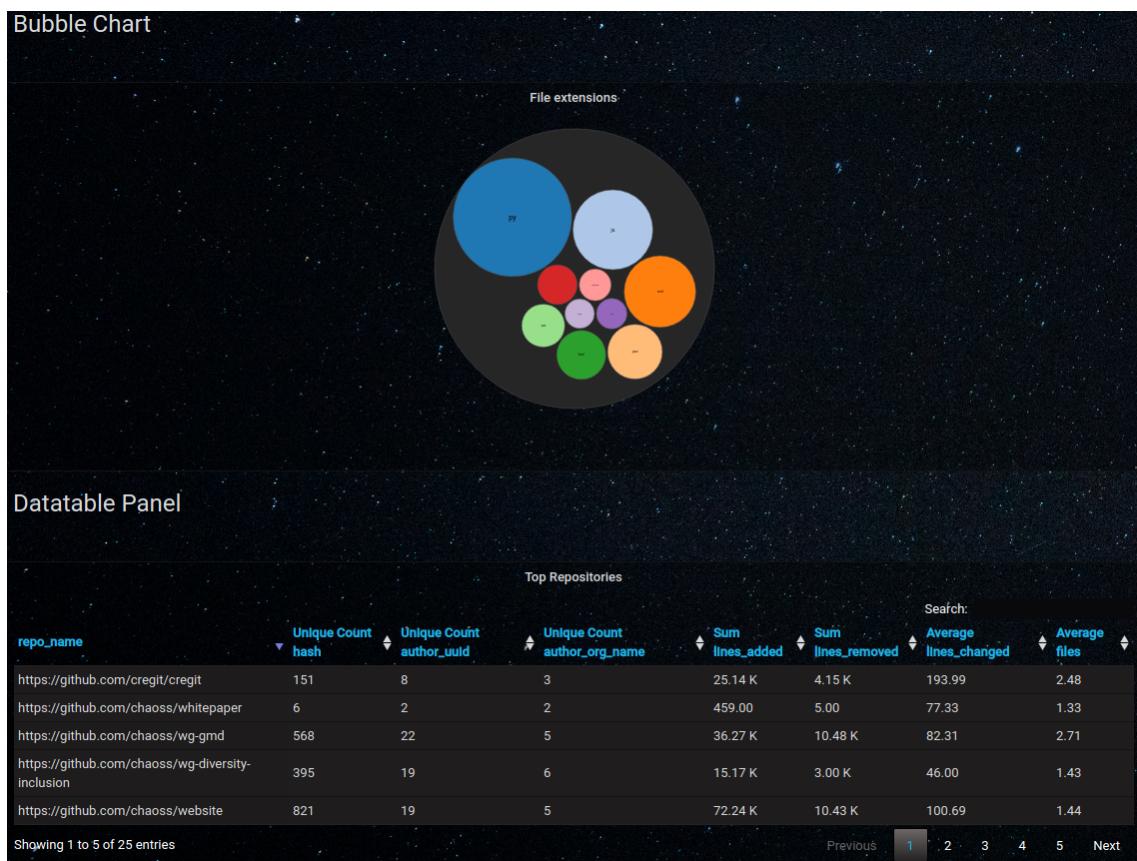


Figura A.9: Extracto del dashboard *Plugins* (5)

Figura A.10: Extracto del dashboard *Plugins* (6)

Bibliografía

Referencias

- [1] Jose Manrique Lopez de la Fuente. *GrimoireLab - Analiza tu proyecto de desarrollo de software con esta plataforma open source.*

<https://ingenieriadesoftware.es/grimoirelab-analiza-proyecto-desarrollo-software-open-source/>

- [2] Stelian Subotin. *Dashboard Design - Considerations and Best Practices.*

[https://www.toptal.com/designers/data-visualization/
dashboard-design-best-practices](https://www.toptal.com/designers/data-visualization/dashboard-design-best-practices)

- [3] Simple KPI. *Dashboard Charts and Graphs Documentación.*

<https://www.simplekpi.com/Resources/Dashboard-Charts-And-Graphs>

- [4] Turnbull, James. *The Docker Book: Containerization is the new virtualization.* James Turnbull, 2014.

Enlaces de interés

- [5] Grafana Documentation.

<https://grafana.com/docs/grafana/latest/>

- [6] Kibana Guide.

<https://www.elastic.co/guide/en/kibana/current/index.html>

- [7] Elasticsearch Reference.

<https://www.elastic.co/guide/en/elasticsearch/reference/current/index.html>

[8] CHAOSS Project.

<https://chaoss.community/>

[9] Scrum (software development methodology).

[https://en.wikipedia.org/wiki/Scrum_\(software_development\)](https://en.wikipedia.org/wiki/Scrum_(software_development))

[10] GrimoireLab Tutorial.

<https://chaoss.github.io/grimoirelab-tutorial/>

[11] Docker Hub Documentation.

<https://docs.docker.com/docker-hub/>

[12] Cauldron Project.

<https://gitlab.com/cauldronio>

[13] Overleaf Documentation.

<https://es.overleaf.com/learn>