Escuela de Doctorado
y Estudios de Posgrado
Universidad de La Laguna

Trabajo de Fin de Máster

Sistema de Monitorización para ULL-CloudIDE

Monitoring system for ULL-CloudIDE

Eduardo Ezequiel Barrio Pareja

La Laguna, 2 de julio de 2019

D. **Vicente José Blanco Pérez**, con N.I.F. 42.171.808-C profesor Titular de Universidad adscrito al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como tutor.

D. **Alberto Hamilton Castro**, con N.I.F. 43.773.884-P profesor Titular de Universidad adscrito al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como cotutor

C E R T I F I C A (N)

Que la presente memoria titulada:

"*Sistema de Monitorización para ULL-CloudIDE*"

ha sido realizada bajo su dirección por D. **Eduardo Ezequiel Barrio Pareja**, con N.I.F. 78.717.782-Y.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 2 de julio de 2019

Agradecimientos

A mis tutores Alberto y Vicente por darme la libertad de explorar libremente las posibles soluciones de este proyecto y guiarme por el camino correcto.

Licencia

Resumen

En el entorno actual donde las Tecnologías de la Información forman parte de nuestro de día a día se hace necesario conocer el estado de nuestras infraestructuras, servicios, aplicaciones, etc, para poder dar un buen servicio a los usuarios de las mismas.

Disponer de una visión en tiempo real de los datos que generan nuestros sistemas nos ayudará a poder tomar mejores decisiones, evitar futuros problemas y debilidades, todo ello incluso antes de que llegue a manifestarse.

Es por ello por lo que este proyecto se basa en el estudio e implementación de herramientas de monitorización.

De una forma empírica y orientada a la resolución de errores, se ha implementado un completo sistema de monitorización de una infraestructura ya existente, a modo de caso de uso, la cual contaba con algunos problemas que debían ser solventados haciendo uso de este nuevo sistema.

Además, se ofrece un completo estudio del sistema analizado proponiendo recomendaciones de mejoras basadas en los datos que se han ido recopilando a lo largo del proyecto.

Palabras clave: Tecnologías de la Información, visión, tiempo real, datos, sistema, debilidades, servicios, herramientas, monitorización.

Abstract

Nowadays IT is a very important part of our daily lives. Therefore, it will be necessary to know the status of our infrastructures, services, and applications, in order to provide a good quality of service to all our users.

The access to real-time monitoring data generated by our systems will help us make better decisions. It will help to avoid future problems and system's weaknesses, even before the incidents will arise.

The project "Monitoring system for ULL-CloudIDE" is based on the study and use of tools to manage data monitoring within this project. In an empirical and solution-oriented way, the project provides a complete monitoring system for a particular existing infrastructure.

In addition, a complete study of the analyzed system is carried out, proposing recommendations for improvements based on the data that we have been collected during the lifetime of the project develpment.

Keywords: IT, real-time view, data, system, weaknesses, services, tools, monitoring.

Índice general

Índice de Figuras

Índice de Tablas

Capítulo 1

Introducción

1.1. Introducción

En los últimos años el gasto que se dedica al sector de las TIC en todos los ámbitos de la sociedad no deja de crecer, es por ello que se pone de manifiesto la necesidad de mantener un control de todos los sistemas que componen este sector.

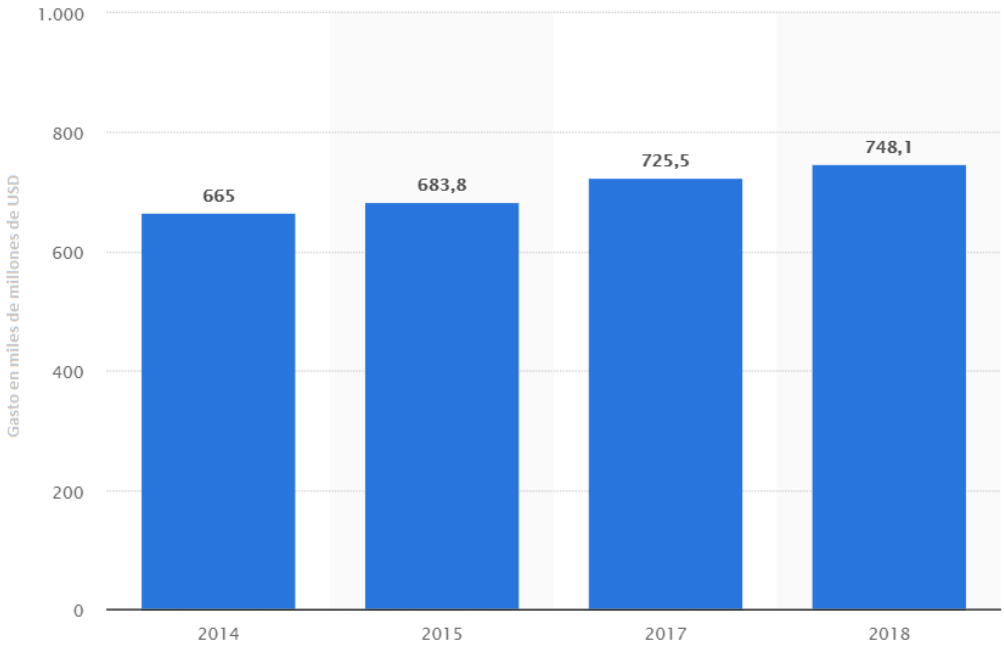

Figura 1.1: Gasto TIC mundial en miles de millones de USD 2014-2018 [1]

Desde grandes empresas multinacionales como Amazon, Google, Microsoft, Facebook, etc... hasta pequeñas empresas con presencia en internet, todas ellas deben mantener sus sistemas funcionando si quieren ser competitivas.

Para poder mantener un correcto funcionamiento de estas empresas se hace uso de gran cantidad de sistemas TIC los cuales requieren de cada vez más número de especialistas en estas áreas de conocimiento, lo que implica que el número de personas empleadas en el sector de las TIC crezca año tras año.

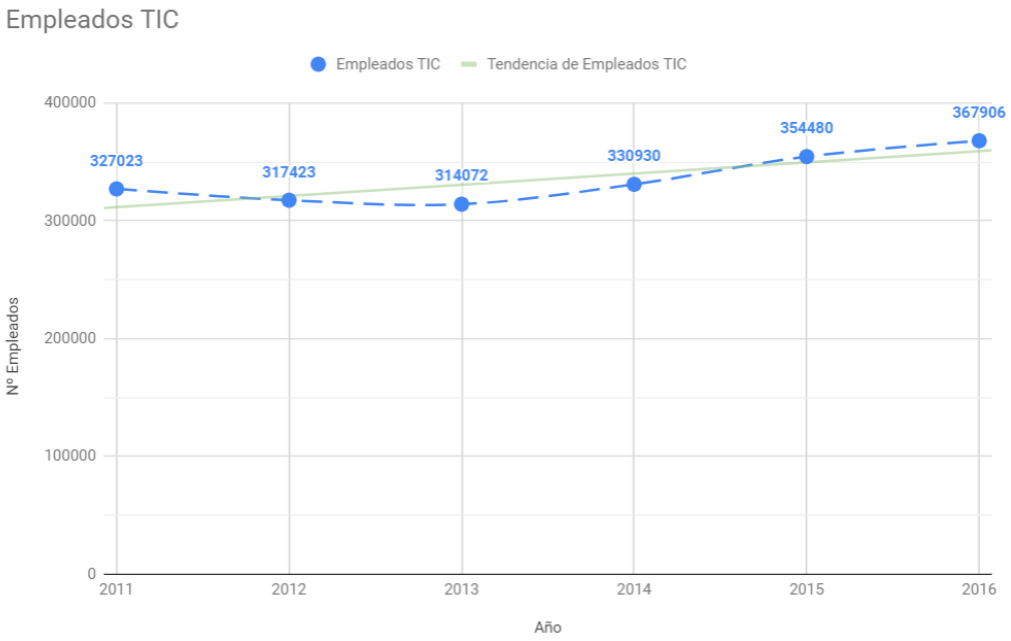

Figura 1.2: Crecimiento empleados TIC en España [2]

Con el fin de realizar un buen mantenimiento de esta gran cantidad de sistemas TIC se necesita poder monitorizar sus estados de manera centralizada sino se quiere perder gran cantidad de tiempo analizando aisladamente cada uno de los sistemas que componen un entorno TIC el cual da servicio desde una pequeña empresa a grandes corporaciones internacionales.

En el sector de la monitorización existen gran cantidad de herramientas que nos permiten de una manera más o menos fácil, centralizada y fiable obtener en tiempo real una imagen conjunta del estado de salud de nuestro entorno TIC.

1.2. Antecedentes

Dado que este proyecto ha sido orientado para la resolución de problemas existentes haciendo uso de herramientas de monitorización, en este apartado se definirá el estado actual del proyecto en el que se resolverán los problemas detectados.

El proyecto que servirá como caso de uso del sistema de monitorización elegido es, "ULL-CloudIDE: Plataforma de entornos de desarrollo para la docencia". En él se implementó una infraestructura que permite a los usuarios de la Universidad de La Laguna tener acceso a un IDE (entorno de desarrollo integrado) online.

De esta manera todo usuario que haga uso de este servicio que ahora proporciona la Universidad puede comenzar a programar rápidamente aislándose de toda la configuración necesaria para comenzar a programar, de manera sencilla y transparente.

Actualmente este servicio se encuentra en fase de pruebas ya que se han detectado algunos errores que no han permitido el uso de la aplicación en un entorno de producción.

Una vez estos errores sean solventados la aplicación podrá pasar a ser utilizada en un entorno de producción por el alumnado de la Universidad.

1.3. Estado actual

Existen gran cantidad de herramientas que realizan funciones de monitorización de diferentes sistemas, desde aparatos médicos que miden la frecuencia cardiaca, hasta torres de vigilancia aérea que controlan de manera global cientos de kilómetros cuadrados. En nuestro caso en particular hemos tenido que trabajar con sistemas que proporcionan información relevante del estado de las aplicaciones.

En el sector TIC en concreto existen multitud de herramientas, las cuales resuelven unos u otros problemas, a continuación se nombran algunas de ellas:

1.3.1. Nagios

Nagios es un sistema de monitorización de redes, de código abierto, que monitoriza tanto elementos hardware como software bajo demanda, permitiendo configurar alertas bajo ciertos patrones [6].

Figura 1.3: Logo Nagios

1.3.2. Splunk

Splunk es un software que busca, monitoriza y analiza grandes cantidades de datos de aplicaciones, sistemas e infraestructuras TIC. Almacena toda esta información para posteriormente generar gráficos, alertas y paneles.

Actualmente cuenta con más de 600 empleados y da soporte a más de 3700 empresas, incluyendo más de la mitad de las 100 empresas que más dinero generan de los Estados Unidos [7].

Figura 1.4: Logo Splunk

1.3.3. Stack ELK

ELK es un conjunto de herramientas compuesto por Elastisearch, Logtash y Kibana, un potente motor de búsqueda de texto, una herramienta de administración de logs y otra para visualizar y explorar datos [8].

Se ha convertido prácticamente en un estándar a la hora de realizar la monitorización de infraestructuras y servicios basadas en logs, con compañías tan importantes como Netflix, Stack Overflow, LinkedIn o Accenture [9].

Figura 1.5: Logo Stack ELK

1.3.4. Graylog

Graylog es una solución para el almacenamiento centralizado de ficheros de log que además permite la consulta de datos almacenados en ellos. Incluye también la funcionalidad de visualización de éstos datos.

Da servicio a imporantes compañías como SAP, T-System, Petronas o Cisco [10].

Figura 1.6: Logo Graylog

1.3.5. Zabbix

Es una solución Open-source para la gestión de logs y su posterior visualización. Usada por grandes empresas como Huawei, TP-Link, Dell o HP.

Figura 1.7: Logo Zabbix

1.4. Descripción del Proyecto

El proyecto *"Sistema de Monitorización para ULL-CloudIDE"*, consiste en definir e implantar una herramienta de visualización de logs (o registros que genera una aplicación por su uso) de una plataforma TIC ya existente con el fin de mostrar de manera gráfica toda la información necesaria para la búsqueda de posibles problemas, mejoras e información clave.

La/s herramienta/s que se utilizarán en este proyecto no están definidas previamente, por lo que se necesitará realizar un estudio previo al desarrollo del mismo con el fin de elegir la/s herramienta/s que mejor se adapten a los requisitos de nuestro proyecto.

1.5. Objetivos

Vistos los puntos anteriores, los objetivos principales a cubrir en este proyecto son los siguientes:

1. **Búsqueda, análisis y elección** de las herramientas necesarias para la implantación del sistema de monitorización.
2. **Desarrollo e implantación** del sistema de monitorización.
3. **Detección y corrección** de errores haciendo uso del sistema de monitorización implementado.

Los objetivos derivados son:

1. **Análisis** del sistema actual y **recomendación** de mejoras.

En resumen: la meta principal de este proyecto es la implantación de un completo sistema de monitorización cuyo primer objetivo es la detección y corrección de errores. Además, se realizará un estudio del sistema en el que se implanta con el fin de realizar recomendaciones de mejora.

1.6. Solución Propuesta

Una vez definidos los objetivos a desarrollar en este proyecto se ha optado por dividir la solución propuesta en dos apartados, arquitectura y requisitos mínimos.

Por un lado la arquitectura será el cuerpo del proyecto, mientras que los requisitos mínimos nos indicarán los características que se deben cumplir en él.

1.6.1. Arquitectura

En este apartado se define la arquitectura propuesta para la implantación del sistema de monitorización, siempre teniendo en cuenta la actual infraestructura;

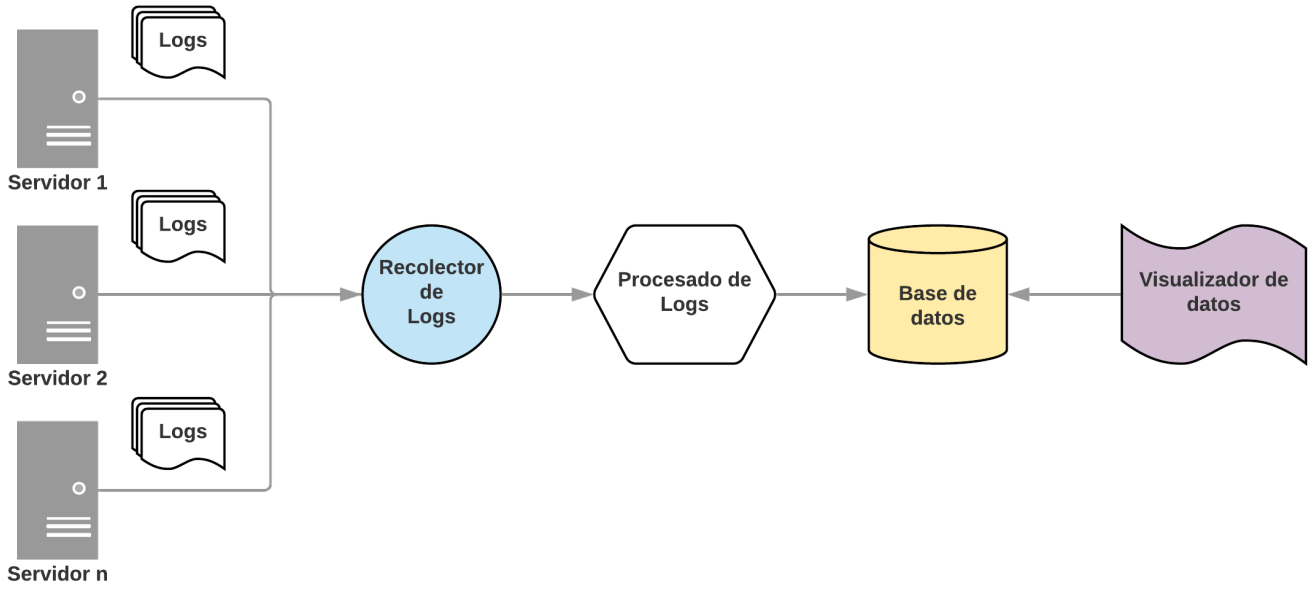

Figura 1.8: Diagrama de flujo de la infraestructura propuesta

Como podemos observar en la figura 1.8, la infraestructura propuesta debe contar con 4 elementos principales:

1. **Recolector de logs:** será el encargado de canalizar todos los logs que se generan actualmente en los servidores de la aplicación para incorporarlos al flujo de la nueva infraestructura.

2. **Procesador de logs:** este elemento realizará las transformaciones y agregaciones necesarias para conseguir extraer de los logs aquella información relevante para ser visualizada posteriormente.

3. **Base de datos:** estos nuevos datos deben ser almacenados en una base de datos para poder ser consultados por el visualizador de datos.

4. **Visualizador de datos:** el visualizador de datos nos permitirá visualizar y analizar los datos de una manera sencilla, rápida y de forma intuitiva.

1.6.2. Requisitos mínimos

Todos aquellos elementos, así como la nueva arquitectura planteada deben cumplir con los siguientes requisitos mínimos:

1. **Rapidez:** todos los elementos deben poder realizar su cometido prácticamente en tiempo real, sin apenas retrasos, ya que uno de los pilares sobre los que se basa este proyecto es el análisis en tiempo real del estado de la aplicación.

2. **Escalabilidad:** la infraestructura debe poder ser escalable horizontalmente [11], de manera que en un futuro si las necesidades de la aplicación crecen, sea posible añadir más instancias de cualquiera de sus elementos, absorbiendo las nuevas necesidades sin necesidad de realizar cambios significativos.

3. **Tolerancia a fallos:** aunque no es uno de los objetivos de este proyecto, el sistema tiene que ser capaz de soportar replicación de datos.

4. **Gratuito:** es importante que la implantación y uso de estos elementos no suponga coste alguno, al menos para las funciones básicas que en este proyecto se pretenden.

5. **Respaldo de la comunidad:** las herramientas de las que se haga uso deben contar con una comunidad que apoye el proyecto, que las mantenga continuamente actualizadas y que resuelvan errores que se vayan encontrando.

6. **Documentación:** se debe contar con documentación suficiente para la implantación de las herramientas.

7. **Compatibilidad con virtualización:** ya que la aplicación existente se basa en virtualización [12], se hace necesario que las herramientas de las que se haga uso soporten esta funcionalidad.

8. **Interoperabilidad:** para poder interactuar con los elementos que componen esta nueva infraestructura estos deben poder ser accesibles a otras aplicaciones con el fin de conseguir una mejor integración con otros servicios.

Capítulo 2

Búsqueda, análisis y elección de herramientas

En este capítulo se cubre la búsqueda, análisis, estudio y elección de las herramientas necesarias para la implantación del sistema de monitorización, tal como se define en el primer objetivo del proyecto.

2.1. Búsqueda y análisis

Cada uno de los elementos analizados en la figura 1.8 cuenta con gran cantidad de herramientas para su implementación, es por ello, que en este proyecto se tenderá a hacer uso de aquellas en las que se tenga experiencia de uso (cumpliendo siempre con los requisitos mínimos especificados), aquellas consideradas estándares y las que nos permitan una rápida implementación en nuestra infraestructura.

En la siguiente tabla se describirán las principales características de las herramientas analizadas:

Análisis herramientas de monitorización					
Nombre	Nagios (2)	Splunk	Stack ELK (5)	Graylog (7)	Zabbix
Monitoriza	Recursos del sistema, red, aplicaciones y servicios	Recursos del sistema, red, aplicaciones y servicios	Recursos del sistema, red, aplicaciones y servicios	Recursos del sistema, red, aplicaciones y servicios	Recursos del sistema, red, aplicaciones y servicios
Rapidez	Tiempo real	Tiempo real	Tiempo real	Tiempo real	Tiempo real
Escalable	Sí	Sí	Sí	Sí	Sí
Tolerancia	Sí	Sí	Sí	Sí	Sí
Gratuito	Sí (3)	Sí (4)	Sí (6)	Sí	Sí (3)
Comunidad	Grande	Grande	Muy Grande	Mediana	Pequeña
Interop. (1)	Sí (API RESTful)	Sí (API RESTful)	Sí (API RESTful)	Sí (API REST)	Sí (JSON-RPC 2.0)
Visualizador	Sí, aunque se recomienda combinarlo con Grafana para una mejor visualización de los datos	Sí	Sí (Kibana)	Sí, aunque se recomienda combinarlo con Grafana y Graphite para una mejor	Sí, aunque se recomienda combinarlo con Grafana para una mejor visualización de los datos
Experencia	No	No	Sí	Sí	No

Cuadro 2.1: Comparación herramientas de monitorización.

(1) - Interoperabilidad.
(2) - La versión analizada es Nagios Core, que es gratuita; existe otra herramienta que es Nagios Log Server especializada en el monitoreo de logs, pero es de pago, por lo que queda fuera de este análisis.
(3) - Código abierto (GLP) versión 2.
(4) - La versión gratuita nos permite el manejo de 500MB de datos diarios.
(5) - El Stack ELK se analiza de forma conjunta ya que es en este modo en el que se considera un estándar.
(6) - Apache License 2.0.
(7) - La versión analizada es Graylog open-source, que es gratuita.

2.2. Elección

Una vez analizadas las principales herramientas en la tabla 2.1 se ha elegido el **Stack ELK** [13] por las siguientes razones:

1. **Documentación:** existe una gran cantidad de documentación, tutoriales y ejemplos disponibles.
2. **Interoperabilidad:** facilidad de interacción con cada uno de sus elementos por separado.
3. **Visualizador:** cuenta con un potente visualizador integrado en la solución (Kibana [14]), por lo que no es necesario hacer uso de otras herramientas, ganando así en facilidad de implantación.
4. **Experiencia:** se cuenta con experiencia previa en esta herramienta.
5. **Popularidad:** actualmente el Stack ELK es una de las herramientas más populares (sino la que más) en lo que a monitorización se refiere.

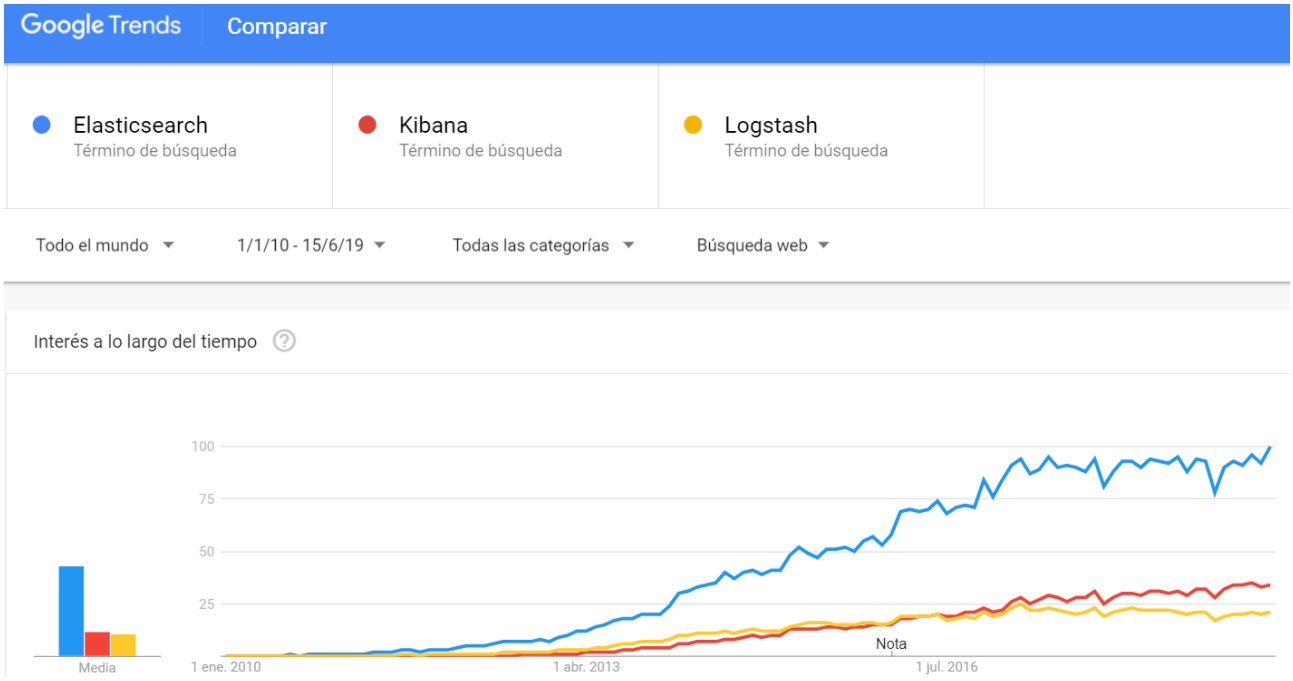

Figura 2.1: Popularidad ELK en los últimos 9 años

6. Capacidad de manejo de todo tipo de logs, pudiendo configurarse manualmente para casi cualquier caso.
7. Las funciones básicas que provee esta herramienta cumplen de sobra con las necesidades del proyecto.
8. **Facilidad de implantación:** el stack completo se puede montar en un único contenedor Docker [15] ya preparada para ello, sin necesidad de realizar complejas configuraciones.
9. Se sigue con la estrategia de la solución en la que se basa CloudIDE, haciendo uso de Docker para la implementación de funcionalidades, optimizando así los recursos hardware con los que cuenta el Servicio de Tecnologías de la Información y Comunicación (STIC).

2.3. Stack ELK

A continuación se describen los elementos que componen el Stack ELK, sus principales características y su flujo de trabajo.

2.3.1. Beats

Beats es una familia de agentes muy ligeros que recolectan datos en hosts o servidores y los envían a Elasticsearch [16]. Existen más de 40 Beats con diferentes características, en este proyecto se hará uso de **Filebeat** el cual está especialmente diseñado para el envío centralizado de logs y ficheros.

Figura 2.2: Logo Beats

2.3.2. Logstash

Logstash es un motor de procesamiento y recopilación de datos basado en plugins. Incluye una gran variedad de plugins que permiten configurarlo fácilmente para recopilar, procesar y reenviar datos en muchas arquitecturas diferentes [17].

Figura 2.3: Logo Logstash

En la siguiente figura puede apreciarse el flujo de trabajo típico de Logstash, pasando de datos desestructurados, a datos que pueden ser indexados por Elasticsearch gracias a las transformaciones realizadas por los muchos plugins que lo conforman.

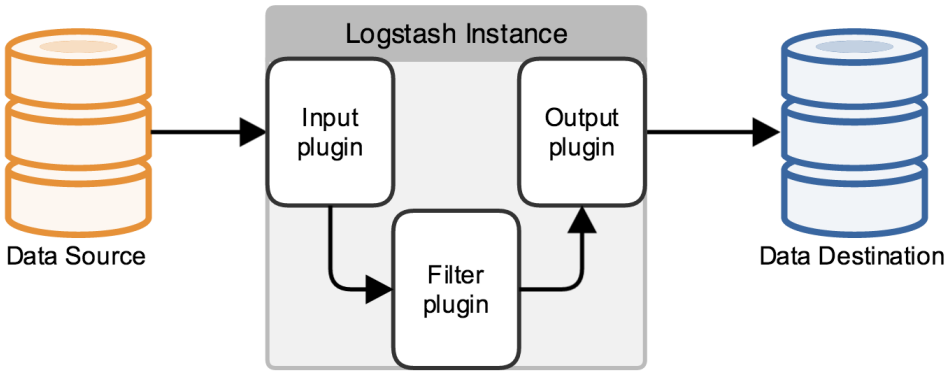

Figura 2.4: Flujo de trabajo de Logstash [3]

2.3.3. Elasticsearch

Elasticsearch es el motor del Stack ELK. Es un servidor de búsqueda orientado a documentos, que nos permite hacer búsquedas entre grandes volúmenes de datos. Está basado en Lucene, lo que provee a Elasticsearch de la funcionalidad de búsqueda de texto completo, ideal para el análisis de grandes volúmenes de texto como son los logs [18] [19].

Figura 2.5: Logo Elasticsearch

2.3.4. Kibana

Es la parte visual del Stack ELK. Kibana nos permite visualizar y explorar la gran cantidad de datos que Elasticsearch se ha encargado de indexar previamente. Kibana nos da la posibilidad de crear todo tipo de dashboards (cuadro de mandos) con gran cantidad de gráficas adaptadas a las necesidades de la aplicación.

Figura 2.6: Logo Kibana

2.3.5. Flujo de trabajo

Este es el flujo de trabajo que hace de ELK una solución integral en el proceso de monitoreo:

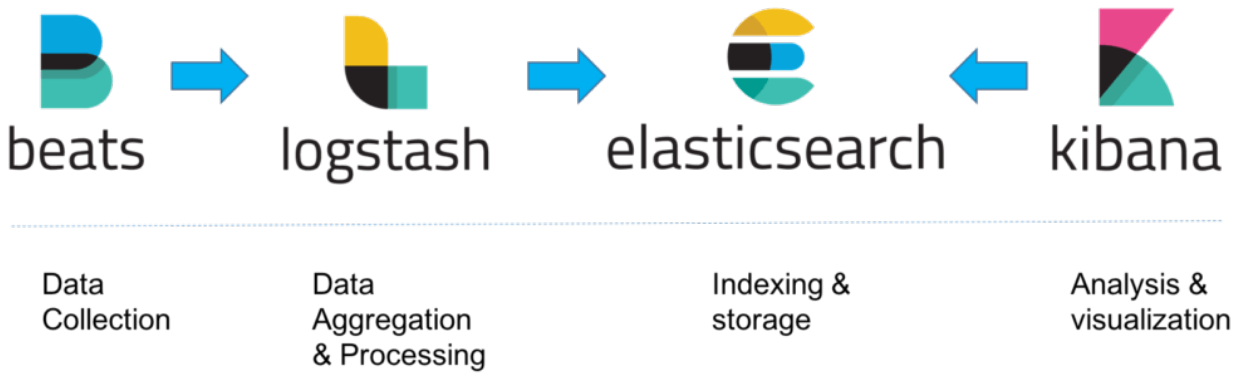

Figura 2.7: Flujo de trabajo Stack ELK [4]

1. **Beats** se encarga de la recolección de los datos.
2. **Logstash** realiza las transformaciones necesarias en estos datos para su almacenaje.
3. **Elastichsearch** almacena estos datos indexándolos (proceso que mejora la rapidez de búsqueda) para que puedan ser consultados casi en tiempo real.
4. **Kibana** nos permite la visualización de estos datos utilizando todo tipo de dashboard (conjunto de gráficas).

Capítulo 3

Desarrollo e implantación

En el siguiente capítulo se abordarán las tareas realizadas para cumplir con el segundo objetivo marcado en este proyecto.

3.1. Implantación del sistema de monitorización

3.1.1. OVirt STIC

Para la instalación de nuestro Stack ELK hemos hecho uso de las máquinas virtuales que nos proporciona el STIC de la Universidad a través de una plataforma de virtualización IaaS (Infraestructure as a Service [20]), **oVirt**, la cual nos permite controlar cada detalle de estas máquinas virtuales.

Podría haberse hecho uso de un servidor dedicado (servidor físico cuyo propósito principal es albergar un servicio concreto) para la instalación del Stack ELK, pero dado que todo el proyecto CloudIDE (caso de uso utilizado para la realización de este proyecto) se encuentra virtualizado haciendo uso de oVirt, lo lógico es mantener esta política, ahorrando así en costes al no necesitar recursos nuevos.

En nuestro caso hemos hecho uso del mismo usuario utilizado en el proyecto anterior, de esta manera se centraliza toda la infraestructura de ULL-CloudIDE en un mismo sitio, facilitando el mantenimiento. Además le hemos asignado una IP estática, de la siguiente manera para poder montar toda la configuración respecto de esta dirección, **10.6.134.254**.

```
auto lo
iface lo inet loopback

auto eth0
iface eth0 inet static
        address 10.6.134.254/24
        gateway 10.6.134.50
```

Este es el estado actual del cuadro de mandos que nos proporciona oVirt, en el que podemos observar cada una de las máquinas virtuales que componen ULL-CloudIDE.

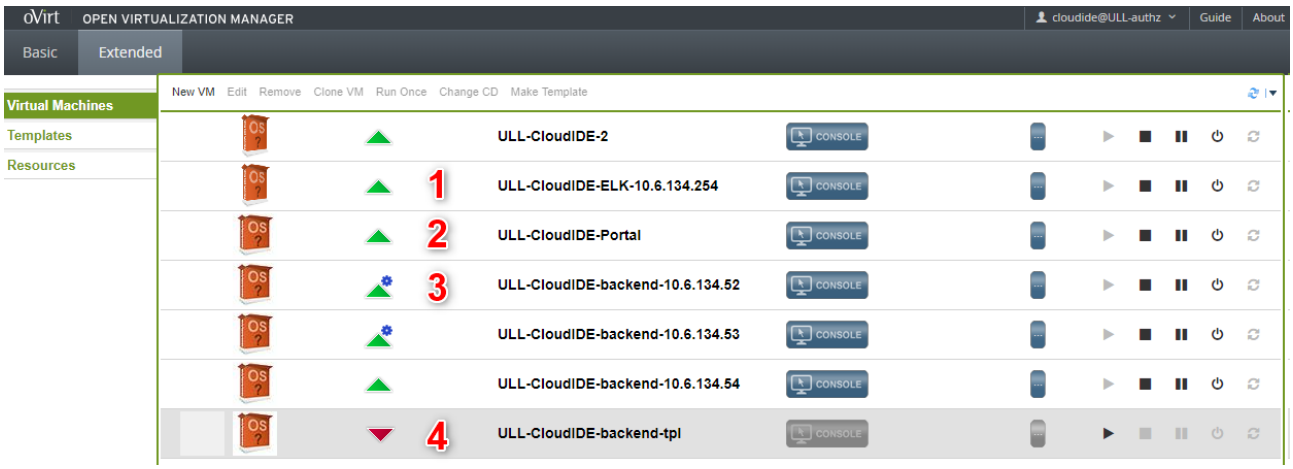

Figura 3.1: Cuadro de mandos oVirt

A continuación se describen cuatro de las máquinas virtuales más relevantes para nuestro proyecto:

1. La máquina virtual en la cual se aloja el **Stack ELK**. Dado los bajos requerimientos iniciales con una única instancia del Stack ELK se puede monitorizar toda la aplicación sin ningún tipo de problema.

2. **ULL-CloudIDE-Portal** se encarga de toda la gestión de la aplicación ULL-CloudIDE, desde gestión de usuarios, hasta creación dinámica de máquinas virtuales bajo demanda.

3. Esta máquina virtual es uno de los **servidores** a los que se conectan los usuarios para hacer uso de la aplicación ULL-CloudIDE. El número de máquinas de este tipo puede variar desde unas pocas para albergar a unos pocos usuarios, hasta más de doscientas, capaces de dar servicio a más de cuatrocientos usuarios. Éstas máquinas son creadas, arrancadas y eliminadas de forma dinámica según las necesidades de la aplicación.

4. **ULL-CloudIDE-backend-tpl** es la plantilla utilizada para crear los servidores descritos en el punto anterior. Es necesaria para que todas las máquinas tengan una misma configuración, misma versión de las aplicaciones, sea posible gestionarlas de la misma manera, etc...

Cada una de estas máquinas tiene unos recursos hardware diferentes según el uso que se les vaya a dar. Teniendo en cuenta que la documentación de la imagen Docker de ELK nos indica que como requisitos mínimos debemos reservar 4GB para Docker (Elasticsearch, por sí sólo requiere al menos 2Gb) [5] se han asignado los siguientes recursos:

Recursos hardware				
SO	**RAM**	**Almacenamiento**	**N° CPU**	**Arquitectura**
Ubuntu 16.04.4 LTS	8GB	20GB	2	64bits

Cuadro 3.1: Tabla de recursos hardware máquina Stack ELK.

Con éstos recursos hardware nos aseguramos que en el entorno actual no tendremos problemas dado el actual volumen de datos que se pretende manejar.

3.1.2. Docker ELK

Como se comenta en anteriores puntos de este proyecto, se hace uso de una imagen Docker para la implantación del Stack ELK, lo cuál simplifica mucho la tarea de instalar y configurar todo el entorno para que todo funcione correctamente.

Selección

Para la selección de la imagen Docker se ha recurrido a la enorme plataforma que nos provee Docker como repositorio, **DockerHub** [21]. Esta plataforma nos permite acceder a aquellas imágenes públicas que tanto empresas tan importantes como MySQL, Ruby, SQL Server, Redis, NGINX, Ubuntu, etc, como usuarios anónimos ponen a disposición de quién quiera hacer uso de ellas.

La imagen que hemos seleccionado es esta:

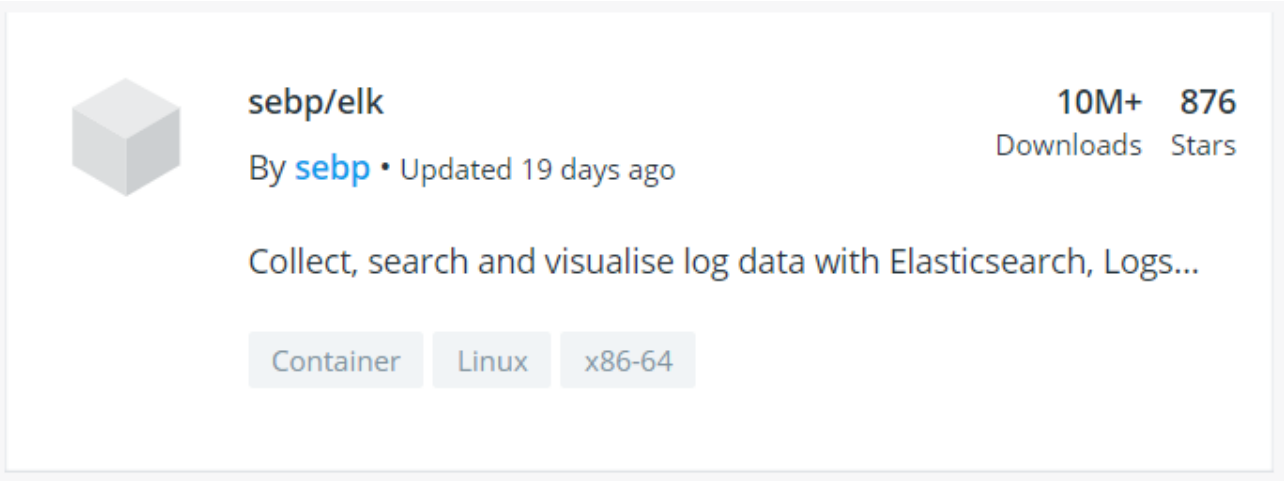

Figura 3.2: Imagen ELK Stack - DockerHub

Se ha elegido esta imagen dado que es la imagen más descargada (más de diez millones de descargas), con mejor puntuación, se actualiza periódicamente y la mejor documentada de las que se han podido encontrar [22].

En concreto se ha utilizado versión la **6.5.1**, la cual contiene las siguientes versiones:

- Kibana v.6.5.1
- Elasticsearch v.6.5.1
- LogStash v.6.5.1

Se debe tener en cuenta que el Stack ELK hace uso de unos puertos concretos, que debemos dejar libres para su uso. La siguiente imagen nos permite apreciar como trabaja internamente el Stack ELK.

El flujo de trabajo se define de la siguiente manera:

1. Los servidores donde instalamos Filebeat envían sus logs a Logtash a través del puerto 5044.

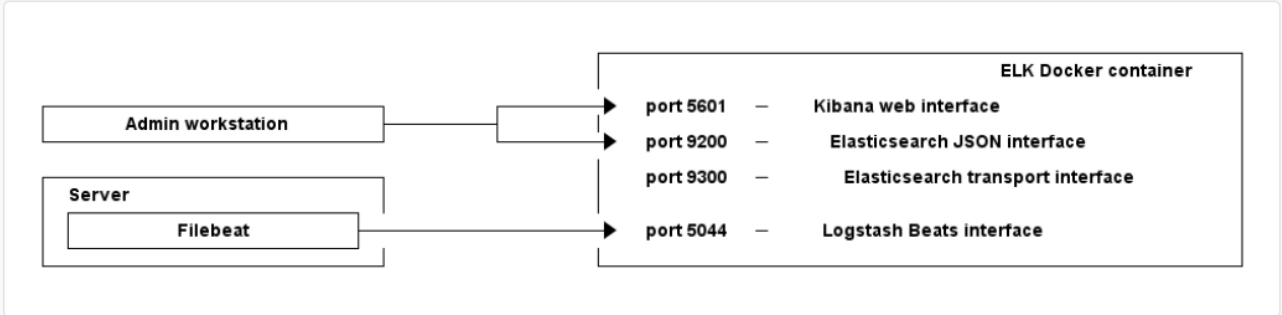

Figura 3.3: Flujo de trabajo interno ELK en Docker [5]

2. Logstash realiza las transformaciones de los logs y envía esta información a través del puerto 9200 a Elasticsearch que la almacena e indexa.

3. Elasticsearch a través del puerto 9200 recibe peticiones, y usa el puerto 9300 para la comunicación entre nodos de un clúster (no necesario en nuestra implementación).

4. Para acceder a Kibana lo hacemos a través del puerto 5601 que está habilitado específicamente para ello.

Instalación

La instalación de la imagen Docker se realiza a través del propio Docker. Para ello se han seguido los siguientes pasos:

1. Instalación de Docker.

```
$ apt-get install docker-engine -y
```

2. Descarga de la imagen del Stack ELK especificando la versión exacta que queremos usar.

```
$ docker pull sebp/elk:651
```

3. Ejecución de la imagen ELK indicándole el mapeo de puertos que debe hacer Docker, de esta manera cada puerto de la máquina host se vincula con el mismo puerto en el contenedor Docker.

```
$ docker run -p 5601:5601 -p 9200:9200
   -p 5044:5044 -it --name elk sebp/elk
```

Una vez realizada la instalación y el arranque del contenedor Docker con el correspondiente mapeo de puertos, debemos comprobar que Kibana está corriendo. Dado que toda esta instalación se ha realizado en las máquinas virtuales del STIC hay que realizar un túnel SSH (método de acceso a hosts) redirigiendo los puertos de la siguiente manera para poder ver la interfaz visual de Kibana en nuestro entorno local.

```
$ ssh -L18080:localhost:9999
   root@cloudide.iaas.ull.es
```

Con ello mapeamos nuestro puerto local 18080 al puerto 9999 de la máquina cloudide.iaas.ull.es.

```
$ ssh -L9999:localhost:5601 usuario@10
    .6.134.254
```

Y con el comando anterior mapeamos el puerto 5601 de la máquina cloudide.iaas.ull.es con el puerto 5601 de la máquina 10.6.134.254, donde se encuentra nuestra instancia ELK.

Con todo ello conseguimos poder tener acceso desde nuestro entorno local a la interfaz web de Kibana a través del puerto 18080.

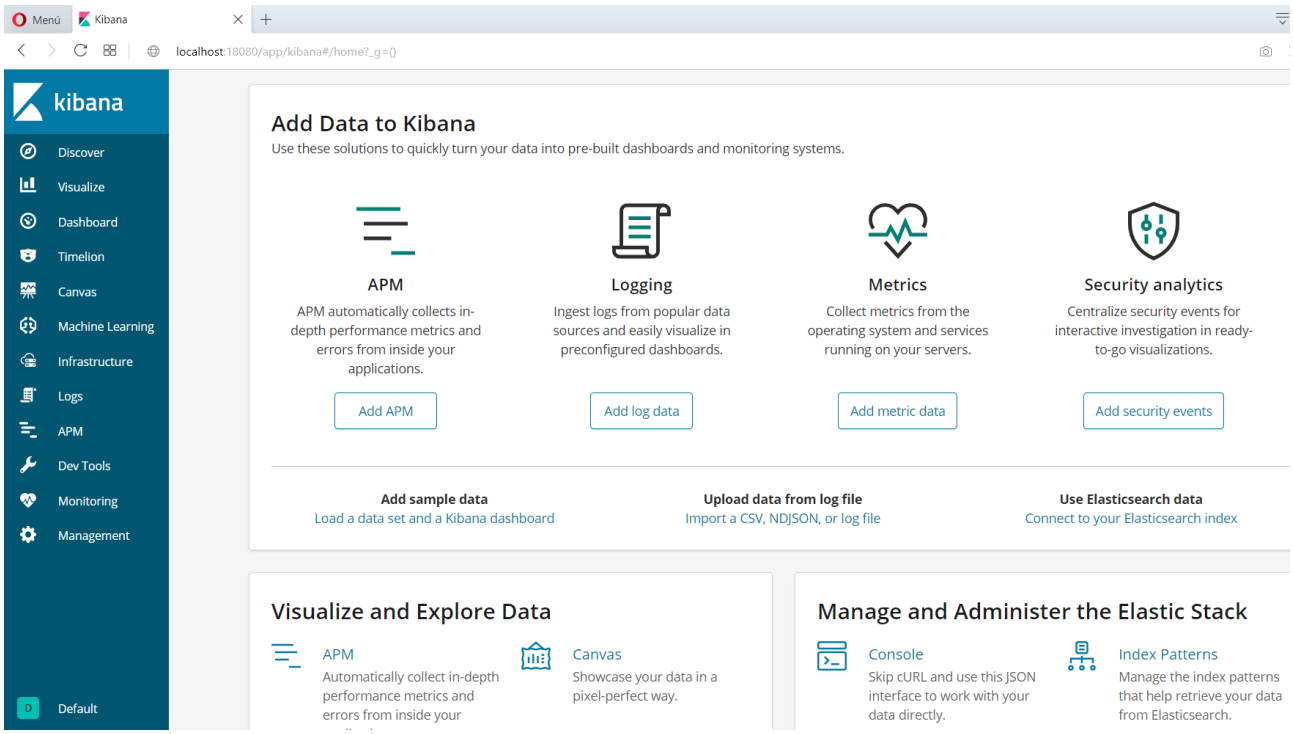

Figura 3.4: Pantalla de inicio de Kibana

3.1.3. Conexión ULL-CloudIDE con ELK

Una vez el Stack ELK está operativo el siguiente paso es realizar el envío de los logs desde los servidores de la aplicación mediante Filebeat a Logstash.

Filebeat

El primer paso es instalar Logstash en los servidores. Dado que los servidores son creados y destruidos de manera dinámica según las necesidades de la aplicación se hace necesario realizar la instalación y configuración de Filebeat cada vez que se cree un nuevo servidor.

Para la automatización de este tipo de procesos de instalado y configuración de herramientas, la actual infraestructura ya contaba con una buena solución implantada, por lo que se ha seguido con la estrategia marcada en el proyecto anterior.

Todos los servidores de la aplicación tienen montado un directorio compartido (carpeta a la que todos los servidores tienen acceso) para archivos comunes, **/mnt/cloudIDE**. En él se alojan aquellos ficheros (ya sean programas o configuraciones) que deben ser utilizados por cada uno de los servidores de la aplicación. En nuestro caso específico hemos creado el siguiente directorio para alojar nuestros ficheros de Filebeat, **/TFG-ULL-CloudIDE/software**. En él podemos encontrar lo siguiente:

```
root@cloudide:/mnt/cloudIDE/TFG-ULL-CloudIDE/software# ls -p | grep -v /
filebeat-6.6.1-amd64.deb
filebeat-6.6.1-amd64.deb_BACKUP
```

Figura 3.5: Directorio Filebeat

El fichero **filebeat-6.6.1-amd64.deb** contiene la herramienta Filebeat en su versión 6.6.1, compatible con nuestro Stack ELK. El fichero BACKUP es exactamente el mismo fichero pero lo guardamos por si el fichero principal se corrompe al ser instalado en uno de los muchos servidores que hacen uso de él.

```
root@cloudide:/mnt/cloudIDE/TFG-ULL-CloudIDE/software# dpkg -i /mnt/cloudIDE/TFG-ULL-CloudIDE/software/filebeat-6.6.1-amd64.deb
(Leyendo la base de datos ... 183710 ficheros o directorios instalados actualmente.)
Preparando para desempaquetar .../filebeat-6.6.1-amd64.deb ...
Desempaquetando filebeat (6.6.1) sobre (6.6.1) ...
Configurando filebeat (6.6.1) ...
Procesando disparadores para ureadahead (0.100.0-19.1) ...
Procesando disparadores para systemd (229-4ubuntu21.21) ...
root@cloudide:/mnt/cloudIDE/TFG-ULL-CloudIDE/software#
```

Figura 3.6: Instalando Filebeat

Con el comando anterior se realiza la instalación de Filebeat en el servidor. Una vez instalado Filebeat necesitamos cargar la configuración común para todos los servidores que permita a Filebeat enviar nuestros logs a Logstash.

En el fichero **filebeat.yml** podemos encontrar dos partes claramente diferenciadas;

1. **output**: en esta sección indicamos que Filebeat debe comunicarse directamente con Logstash, a la dirección 10.6.134.254, puerto 5044, el cual hemos definido anteriormente como puerto de entrada de datos a Logstash en la figura 3.3. Además se especifica que este envío debe realizarse cada 30 segundos.

2. **filebeat**: con esta configuración indicamos a Filebeat que el fichero que debe enviar es el fichero **dockerStats.log**, localizado en /var/log.

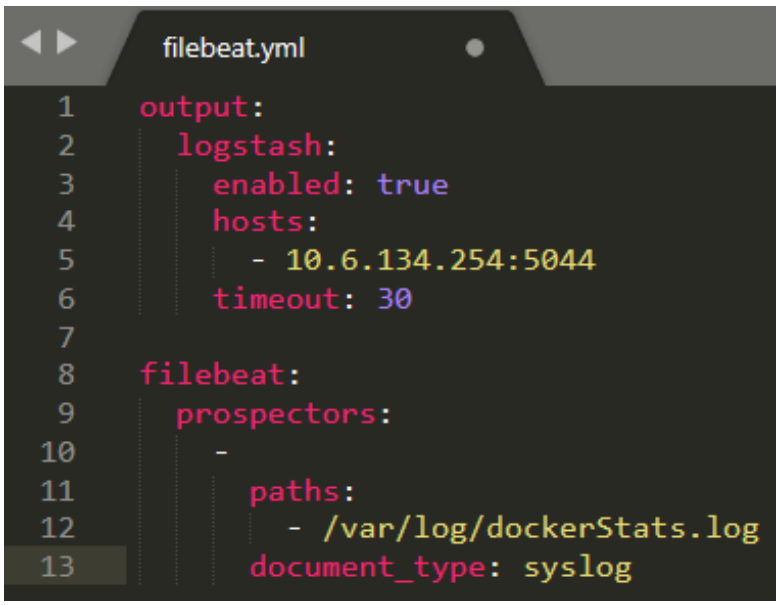

Figura 3.7: Fichero de configuración de Filebeat, filebeat.yml

dockerStats

En la figura 3.7 se puede observar que el fichero que se envía a Logstash es un fichero llamado dockerStats.log. Este fichero no existe por defecto en Docker, sino que es un fichero que se ha generado para este proyecto y es la piedra angular de toda la información que envía Filebeat de nuestros servidores a Logstash.

Para generar este fichero se ha tenido que crear un ***script*** (conjunto de comandos agrupados en un fichero) que se ejecuta en cada uno de los servidores para que de forma automática envíen la información necesaria a Logstash.

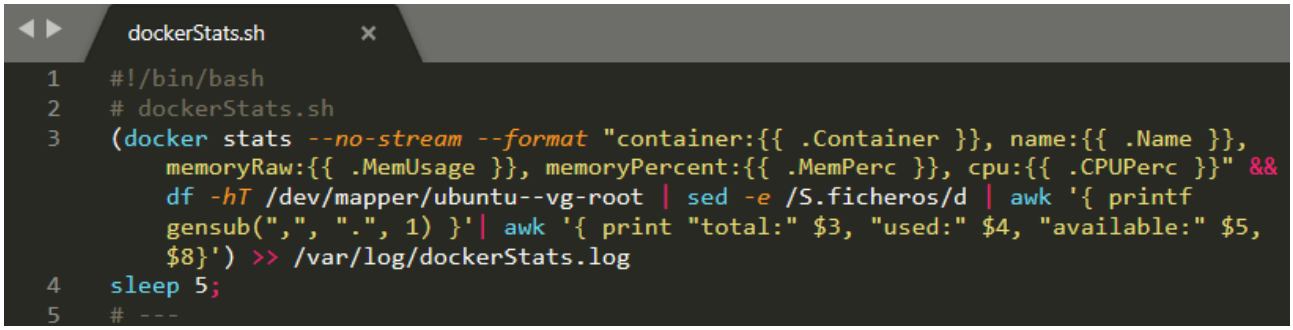

Figura 3.8: Script dockerStats.sh

El script está compuesto principalmente por dos comandos;

- **docker stats:** el primero de ellos es el encargado de recopilar la información sobre Docker gracias a una API que provee Docker para poder mostrar la información más relevante de cada uno de sus contenedores.
- **df -hT:** nos muestra el uso del disco duro de nuestro servidor.

Para **Docker** se analiza individualmente cada uno de los contenedores, de esta manera se puede afinar más en la búsqueda de posibles problemas. En concreto se envían los siguientes datos (para cada contenedor):

- **ID** del contenedor que se está analizando.
- **Nombre** del contenedor.
- **Memoria** RAM utilizada.
- Porcentaje de la memoria RAM total utilizada.
- Porcentaje de **CPU** utilizada.

Para la monitorización de los **servidores** se envía la siguiente información:

- Capacidad de disco total.
- Capacidad de disco utilizada.
- Capacidad de disco disponible.

```
usuario@backend_52:~$ tail /var/log/dockerStats.log
container:c7e1955d70d5, name:workspacegmoqh13egd9581dc_null_che_db, memoryRaw:192.2MiB / 1GiB, memoryPercent:18.77%, cpu:0.05%
container:d3fc8622800c, name:ULLcloudIDE-8083, memoryRaw:211.2MiB / 750MiB, memoryPercent:28.16%, cpu:19.89%
container:4800bff7f957, name:ULLcloudIDE-8082, memoryRaw:209MiB / 750MiB, memoryPercent:27.86%, cpu:4.96%
total:18G used:11G available:6.4G
```

Figura 3.9: Contenido del fichero de log dockerStats.log

Toda esta información se almacena en el fichero nombrado anteriormente, **dockerStats.log** cada 30 segundos gracias a la ejecución de un comando que ejecuta el script dockerStats.sh cada 30 segundos.

Este comando se ejecuta una única vez al iniciar la máquina virtual y se mantiene en ejecución hasta que la máquina se destruya, por lo que conseguimos monitorizar el sistema en todo momento.

scriptInitializaDockerStats.sh

```
1  while sleep 30;
2  do (sh /mnt/cloudIDE/TFG-ULL-CloudIDE/scripts/dockerStats.sh &) ;
3  done &
```

Figura 3.10: Comando de arranque del script dockerStats.sh

Logstash

Una vez hemos realizado la instalación y configuración de Filebeat tenemos los ficheros de los servidores ya se están enviando a Logstash, ahora falta realizar algunas modificaciones en Logstash para poder realizar las tareas que se exponen en la figura 2.4.

El primer paso corresponde a la **entrada de datos**. En el fichero de configuración 02-beats-input.conf debemos indicar que los logs que proceden de **Filebeats** lo harán a través del puerto 5044, para ello se debe modificar el fichero 02-beats-input.conf de la siguiente manera:

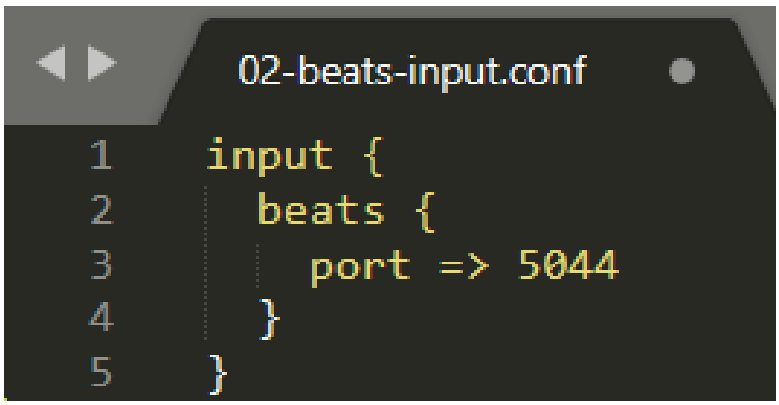

Figura 3.11: Fichero de configuración de Logstash, 02-beats-input.conf

El segundo paso consiste en la realización de las transformaciones necesarias para que Elasticsearch pueda indexar esta información. Estas transformaciones se realizan en el fichero **10-syslog.conf**.

Para realizar esta tarea de conversión de datos desestructurados en datos estructurados, Logstash cuenta con un plugin que se ha convertido en parte prácticamente indispensable de Logstash, **Grok** [23].

Grok nos permite descomponer cualquier cadena de texto en bruto en elementos estructurados, facilitando la consulta de éstos datos posteriormente. Trabaja haciendo uso de algo muy parecido a expresiones regulares, que denomina *"grok patterns"*. Estos *grok patterns* nos permiten "mapear"parte específicas de nuestros ficheros de logs, a variables concretas.

Figura 3.12: Logo Grok

Este es el *grok pattern* que hemos utilizado para el mapeo de los logs que nos envían los servidores en el fichero dockerStats.log y por medio de Filebeat.

```
grok {
  match => { "message" => [ "container:%{WORD:containerId}, name:%{WORD:name}-%{WORD:machine},
    memoryRaw:%{NUMBER:memoryNow:float}%{NOTSPACE:unit} / %{NUMBER:memoryMax:float}%{NOTSPACE:unitTotal},
    memoryPercent:%{NUMBER:memoryPercent:float}%, cpu:%{NUMBER:cpu:float}", "total:%{NUMBER:total:float}%{NOTSPACE:unit}
    used:%{NUMBER:used:float}%{NOTSPACE:unit} available:%{NUMBER:available:float}%{NOTSPACE:unit}" ] }
}
```

Figura 3.13: Grok en fichero configuración de Logstash, 10-syslog.conf

Una vez hayamos realizado la manipulación de los datos que nos llegan a Logstash el siguiente paso es enviar estos datos ya estructurados a Elasticseach para que sean indexados y posteriormente consultados por Kibana.

Para ello necesitamos hacer uso de otro fichero de configuración que nos proporciona también Logstash, **30-output.conf**

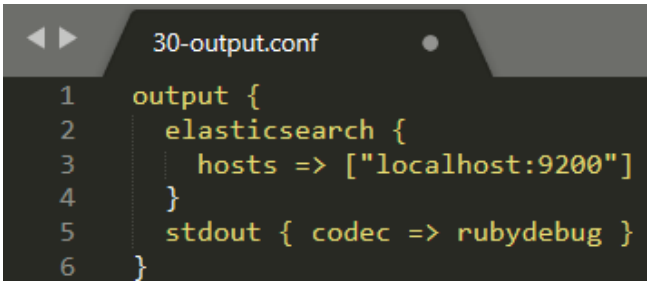

Figura 3.14: Fichero de configuración de Logstash, 30-output.conf

En él se indica que la salida se envíe a Elasticsearch a la dirección local (ya que todo el Stack ELK se encuentra en la misma máquina) por medio del puerto 9200, tal como se especifica en la figura 3.3.

Una vez realizadas las configuraciones anteriores debemos cerciorarnos de que la comunicación entre las diferentes partes de nuestra infraestructura se está realizando de forma correcta, para ello ejecutamos el siguiente comando en uno de los servidores con Filebeat instalado. Y comprobamos que todo funciona correctamente.

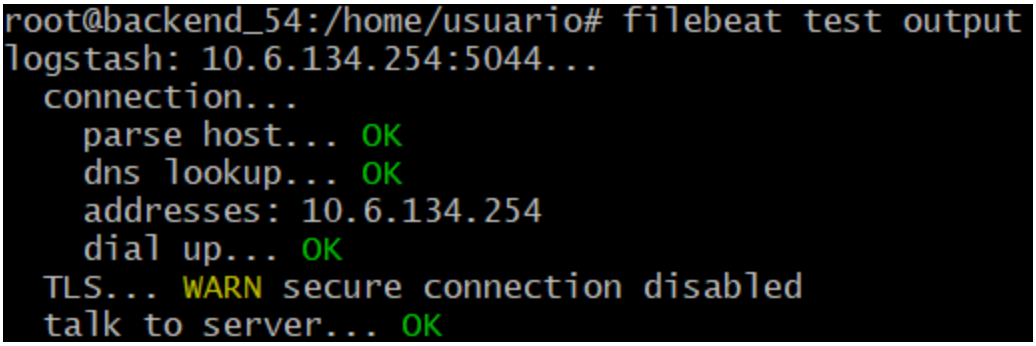

Figura 3.15: Comunicación Filebeat - Logstash funcionando correctamente

Elasticsearch

Y el último paso corresponde al indexado de datos por parte de Elasticsearch.

Kibana nos proporciona una interfaz para el acceso a los datos por medio de Elasticsearch, por lo que es ahí donde vamos a comprobar que los datos que nos llegan son los correctos.

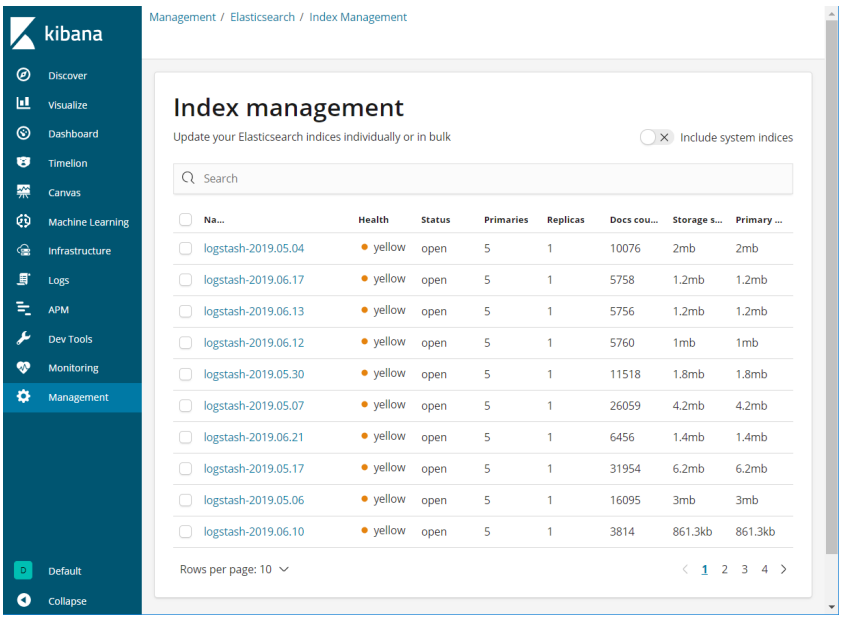

Figura 3.16: Logs indexados por Elasticsearch

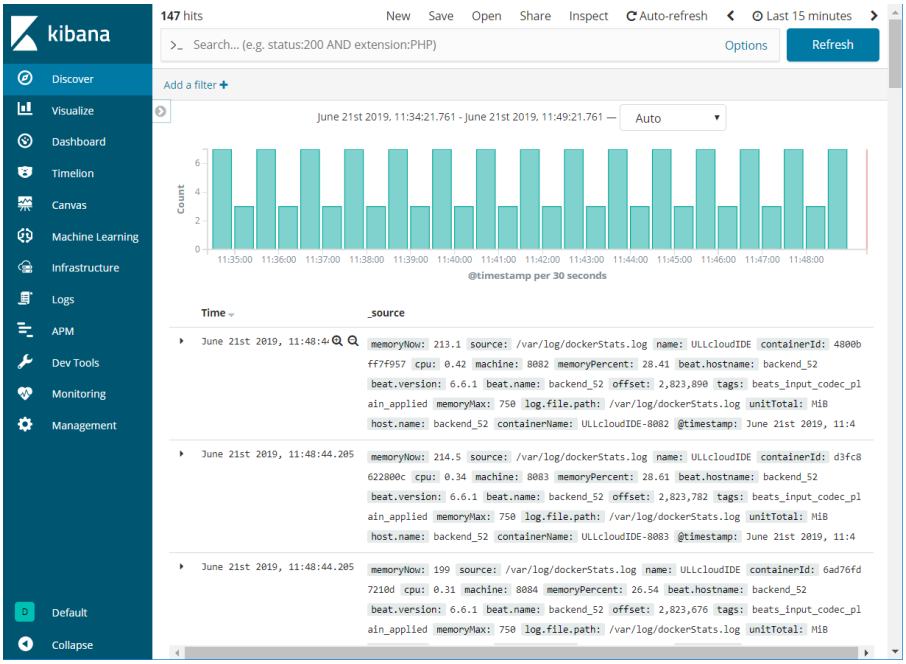

Figura 3.17: Logs llegando a Elasticsearch

Como se puede observar los logs van llegando de forma ininterrumpida y son indexados por Elasticsearch, por lo que ya tenemos lista la infraestructura para comenzar el proceso de monitorización de nuestros servidores.

3.2. Monitorización

En este apartado abordaremos la monitorización de la aplicación haciendo uso de Kibana como visualizador de gráficas haciendo uso de los datos ya indexados por Elasticseach.

3.2.1. Kibana

Tal como se expuso en apartado *sobre Kibana*, una de las razones de la elección de esta herramienta es su capacidad de creación de gran cantidad gráficas adaptadas a las necesidades de la aplicación, pues es en este punto donde vamos a sacarle rentabilidad a la decisión tomada.

En la siguiente figura podemos ver algunos de los tipos de gráficas de los que nos provee Kibana.

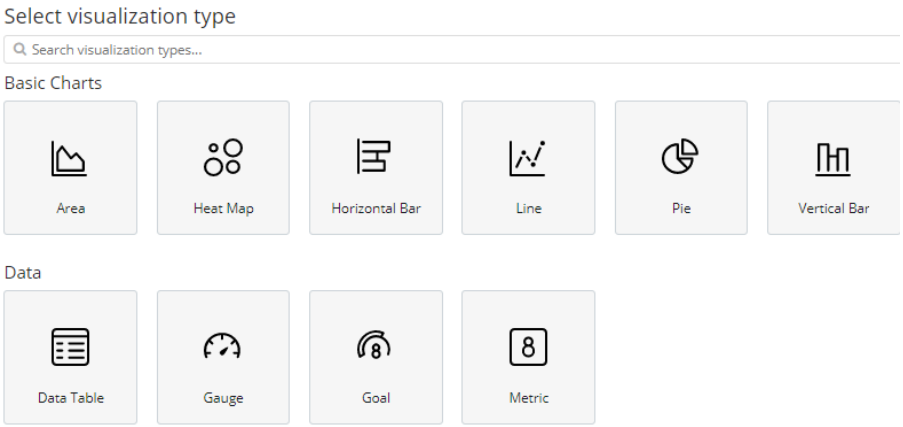

Figura 3.18: Tipos de gráficas seleccionables en Kibana

Kibana, además de hacer uso de las variables que le provee Elasticsearch también nos da la posibilidad de crear nuestras propias variables haciendo cualquier tipo de transformación de las variables ya existentes, funcionalidad de la que hemos hecho uso para hacer nuestras gráficas un poco más visuales.

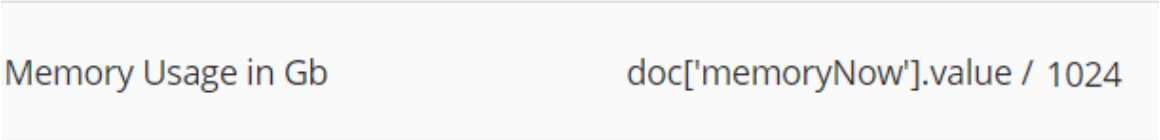

Figura 3.19: Nuevo campo creado para Kibana

3.2.2. Docker

A continuación se muestran todas las gráficas que se han creado para la monitorización de la actividad de Docker y se definirán con un poco más de precisión las que creemos que aportan más información a este proceso.

Title	Type
Docker - Core usage per host	Line
Docker - Hits by hostname	Pie
Docker - Memory usage in Gb per container	Line
Docker - Memory usage per host	Line
Docker - Memory Usage per host - Circle	Gauge
Docker - Number of containers per host	Metric
Docker - Number of containers per host - Line	Line

Figura 3.20: Gráficas creadas para monitorización Docker

Gráfica 1 - Docker

En la siguiente gráfica observamos la **suma de la memoria RAM** utilizada por todos los contenedores Docker en cada uno de los servidores. Se han definido tres intervalos de colores; de 0 a 1.5GB nos encontraríamos en un estado con alta disponibilidad de RAM actual (color verde); de 1.5 a 4GB ya pasaríamos a un estado de disponibilidad normal (amarillo); y de 4 a 8GB ya entraríamos en un estado de alerta. De esta manera, simplemente viendo este "semáforo"se puede apreciar cuál es el estado de la RAM consumida por Docker en cada servidor.

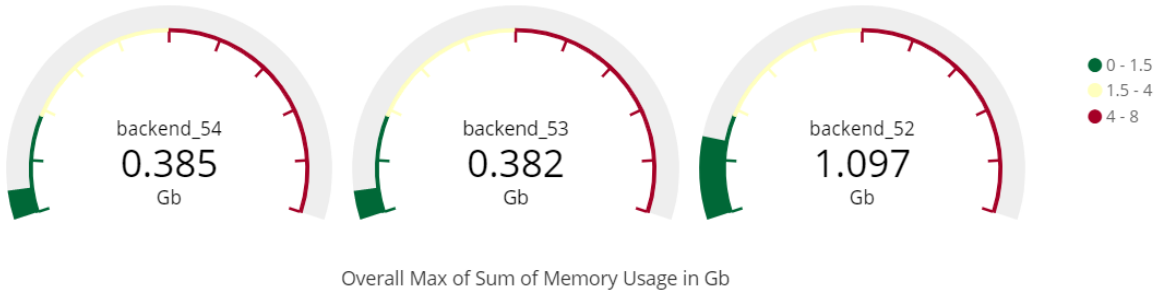

Figura 3.21: Suma de toda la memoria RAM utilizada por Docker en cada servidor

Gráfica 2 - Docker

En esta gráfica podremos identificar los problemas relacionados con el consumo de CPU por parte de los diferentes servidores en el tiempo ya que en ella se muestra el histótico del porcentaje total de CPU utilizado por cada servidor.

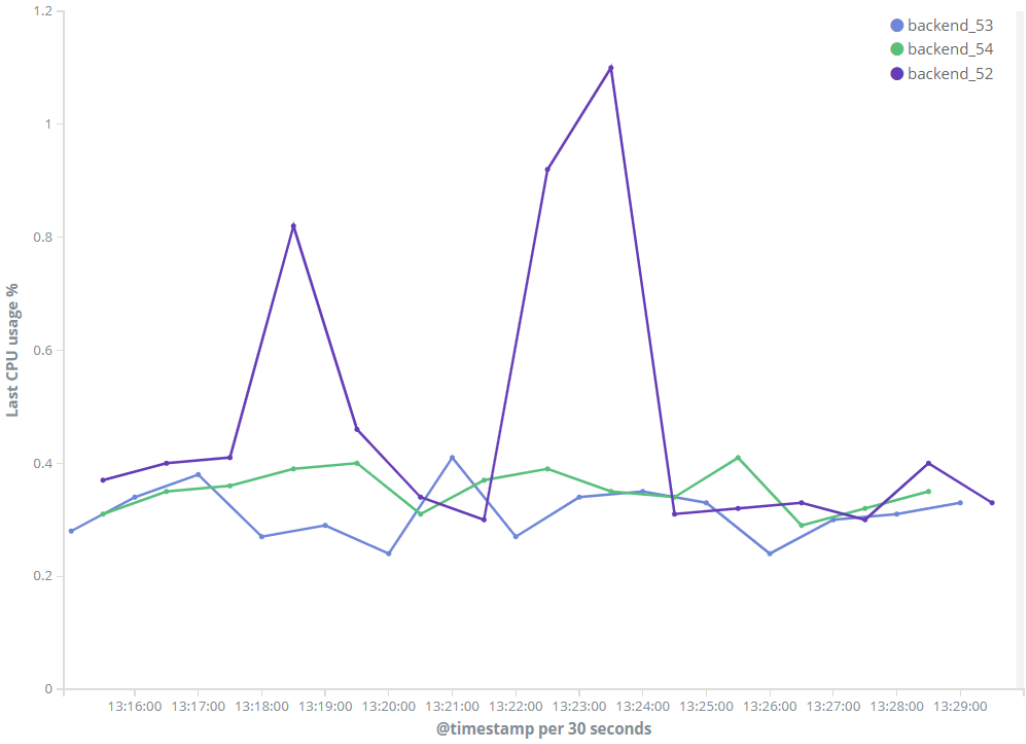

Figura 3.22: Porcentaje de CPU utilizada por cada servidor en el tiempo

Gráfica 3 - Docker

Y para poder llevar un control un poco más exhaustivo de la memoria RAM utilizada por cada servidor, se ha creado la siguiente gráfica que nos permite control el total de memoria RAM utilizada por cada contenedor Docker. De esta manera seremos capaces de identificar en qué contenedor se encuentra cualquier posible problema relacionado con la memoria RAM.

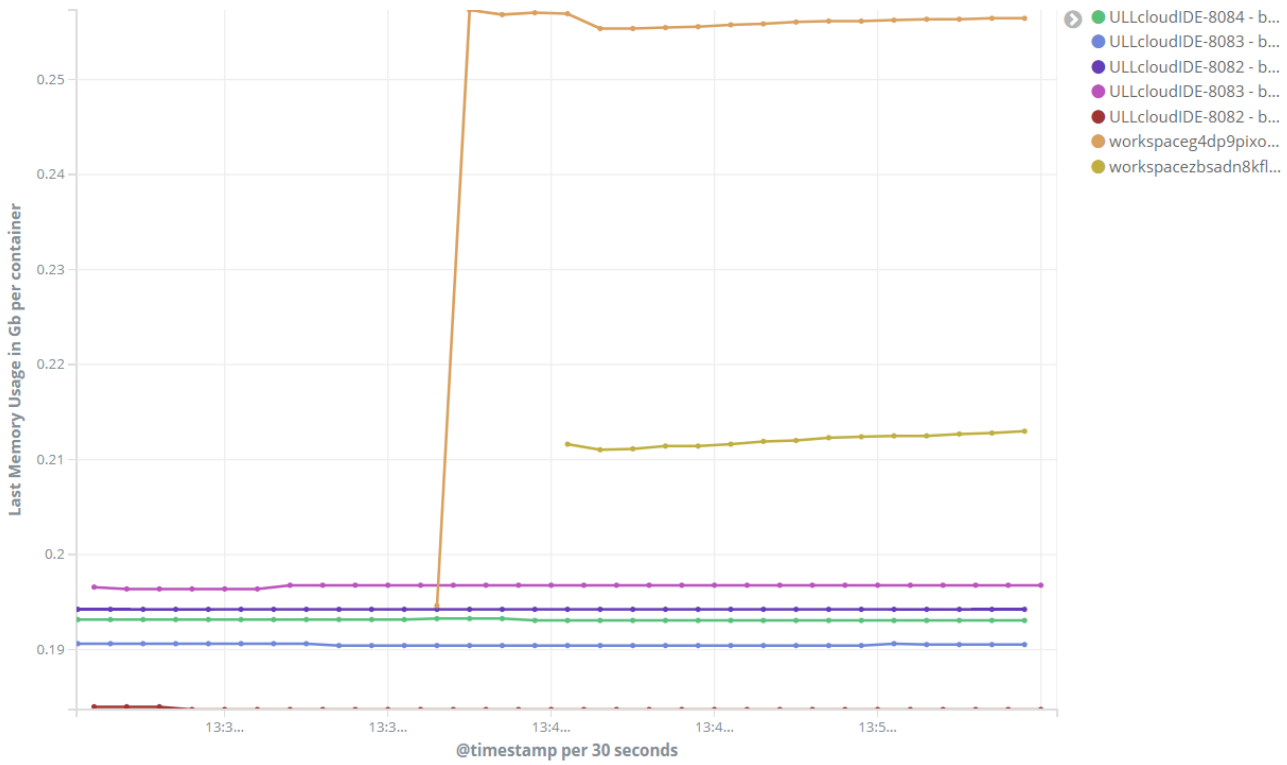

Figura 3.23: Memoria RAM utilizada por cada contenedor en el tiempo

Además también nos permite ver cuándo han sido creados los contenedores, como puede apreciarse en la figura 3.23 (línea de puntos intermedia).

3.2.3. Recursos del sistema

Las gráficas que hemos creado para monitorizar los recursos del sistema son bastante similares a las anteriormente creadas para Docker, así que nos limitamos a mostrarlas de forma genérica.

Figura 3.24: Gráficas de monitorización de servidores

3.2.4. Dashboards

Como hemos visto en los dos puntos anteriores Kibana nos proporciona gráficas muy interesantes, que pueden contener gran cantidad de información, pero además cuenta con una funcionalidad que hace que todas las gráficas anteriores cobren incluso más protagonismo, nos permite visualizarlas de forma conjunta.

Kibana define los *dashboards* como conjuntos de gráficas en una misma visualización. Esto nos permite realizar un análisis casi total de la situación actual de la infraestructura de un sólo vistazo.

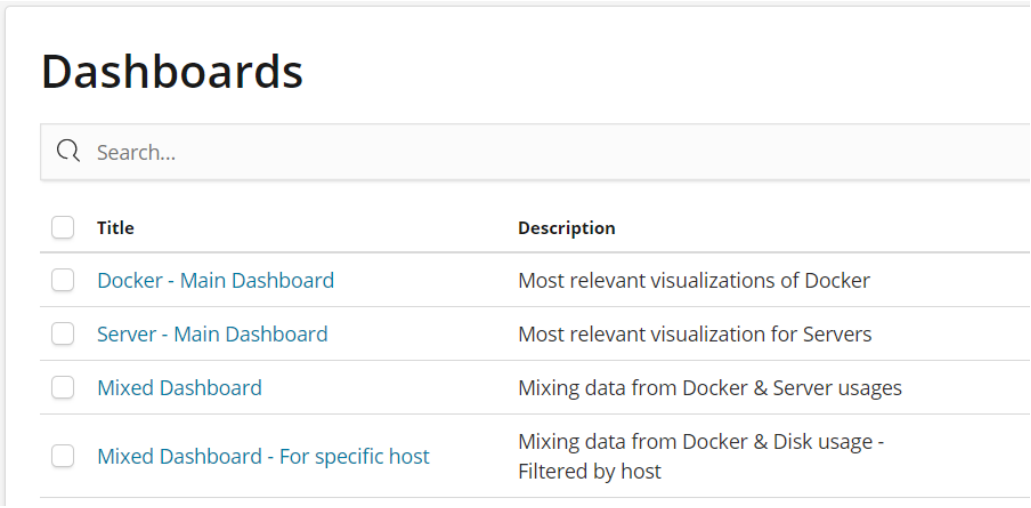

Figura 3.25: Dashboards implementados en Kibana

Estos son los *dashboards* que se han implementado para este proyecto. Han sido creados uniendo las diferentes gráficas implementadas en los pasos anteriores.

De esta manera tenemos un *dashboard* específico para Docker, otro para los Servidores, otro que contiene la información individual de cada uno y por último uno que aúna toda la información de ambos y nos permite hacer un filtrado más específico por servidor.

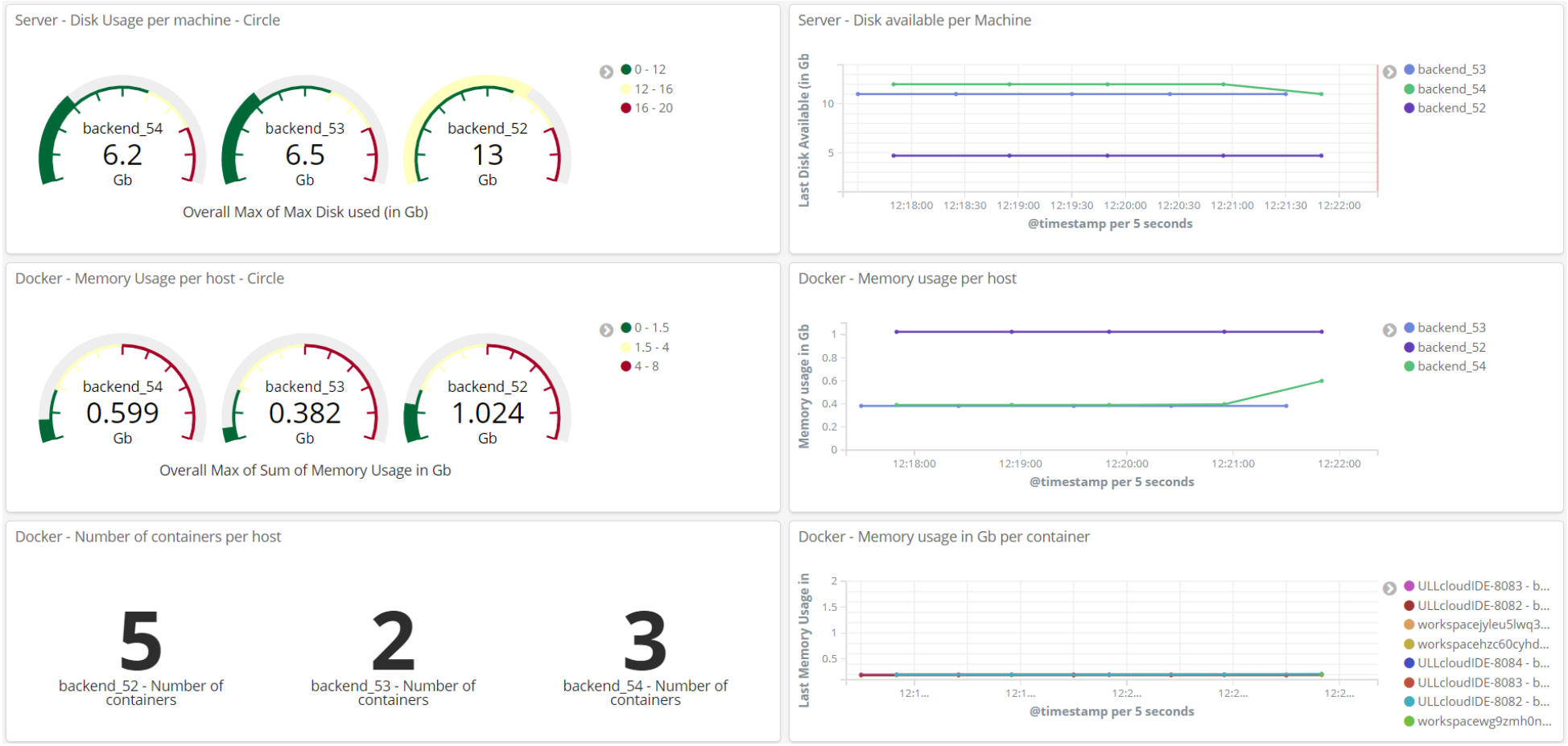

Figura 3.26: Dashboard que combina información de Docker y de los servidores

3.3. Automatización de tareas

Para la implantación de todo el sistema de monitorización se han tenido que automatizar algunas tareas que de otra forma tendrían que ser realizadas de forma manual cada vez que se crea un nuevo servidor.

La automatización de estas tareas se ha creado dentro de un script que ya existía previamente con el mismo propósito, ejecutarse cada vez que un nuevo servidor es creado. El fichero en cuestión es ***actualizaArrancaBackend.sh***, que se encuentra dentro del directorio compartido.

Instalación y configuración Filebeat

Como se describe en el apartado anterior, para realizar la instalación de Filebeat tienen que seguirse unos pasos en un orden concreto para que la conexión entre los servidores y el Stack ELK se cree de forma correcta, por ello se han añadido los siguientes comandos, que realizan la instalación y configuración de Filebeat, así como la ejecución del ya mencionado *dockerStats.sh*.

```
echo Instalamos Filebeat y ejecutamos dockerStats ========================
dpkg -i /mnt/cloudIDE/TFG-ULL-CloudIDE/software/filebeat-6.6.1-amd64.deb
sleep 10
cp /mnt/cloudIDE/TFG-ULL-CloudIDE/code/Filebeat/filebeat.yml /etc/filebeat/
sleep 10
/etc/init.d/filebeat start
while sleep 30; do (sh /mnt/cloudIDE/TFG-ULL-CloudIDE/scripts/dockerStats.sh &) ; done &
```

Figura 3.27: Instalando y configurando Filebeat y ejecutando dockerStats.sh

Cambio de hostname

Para poder diferenciar de qué servidor proviene la información que llega al Stack ELK, es necesario cambiar el hostname (nombre del servidor) por algún nombre que lo identifique de forma inequívoca. Dado que las direcciones IP que se manejan en este proyecto siguen siempre el mismo patrón, **10.6.134.XXX**, se han utilizado los últimos tres dígitos de la dirección IP para componer el nuevo nombre del equipo.

De esta manera, un equipo con la dirección 10.6.134.54 tendrá como hostname el siguiente nombre, **backend_54**.

```
echo Actualizamos Hostname ========================
IP=$(ifconfig eth0 | grep "inet:" | sed -e's/^.*inet:\([0-9.]*\) .*$/\1/')
NUM_MAQ=$(echo $IP | cut -d. -f4 )
NombreMaq="backend_${NUM_MAQ}"
echo "Fijando nombre a -$NombreMaq-"
hostnamectl set-hostname $NombreMaq
```

Figura 3.28: Actualizando hostname

Capítulo 4

Detección y corrección de errores

En este capítulo se desarrollará la detección y corrección de errores que hacen actualmente al proyecto no estar disponible para su uso en producción tal como se define en el tercer objetivo del proyecto.

4.1. Caída del servicio

El principal problema con el que nos hemos encontrado en la aplicación es que pasado un cierto tiempo, aún sin determinar, la aplicación ULL-CloudIDE deja de funcionar sin razón aparente, mostrando el siguiente aviso.

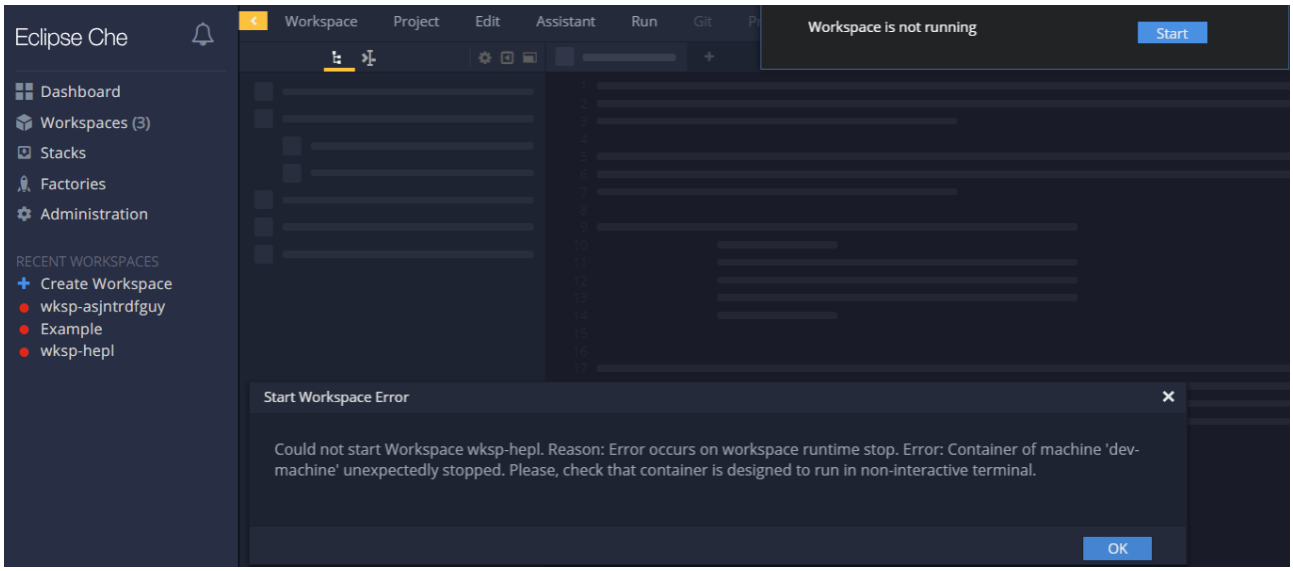

Figura 4.1: Error CloudIDE

Dado que en este punto la monitorización del sistema ya está totalmente montada y funcionando correctamente, se ha hecho uso de ella para intentar descubrir la raíz del problema.

En un principio se pensaba que el problema podía venir de un exceso de consumo de recursos como CPU o RAM por parte de algún contenedor Docker y por ello el sistema operativo mataba el proceso, con la consiguiente caída del servicio, así que nos centramos en esta posibilidad.

Para ello se creó un *Dashboard* específico que contara el número de contenedores Docker, el uso de RAM y de CPU en el tiempo, con el fin de dejar la aplicación corriendo

y posteriormente analizar cuándo se producía el error. Además se filtró la información para que únicamente se mostrara la información del servidor **backend_54** en el que se levantaron varios contenedores Docker para esta prueba.

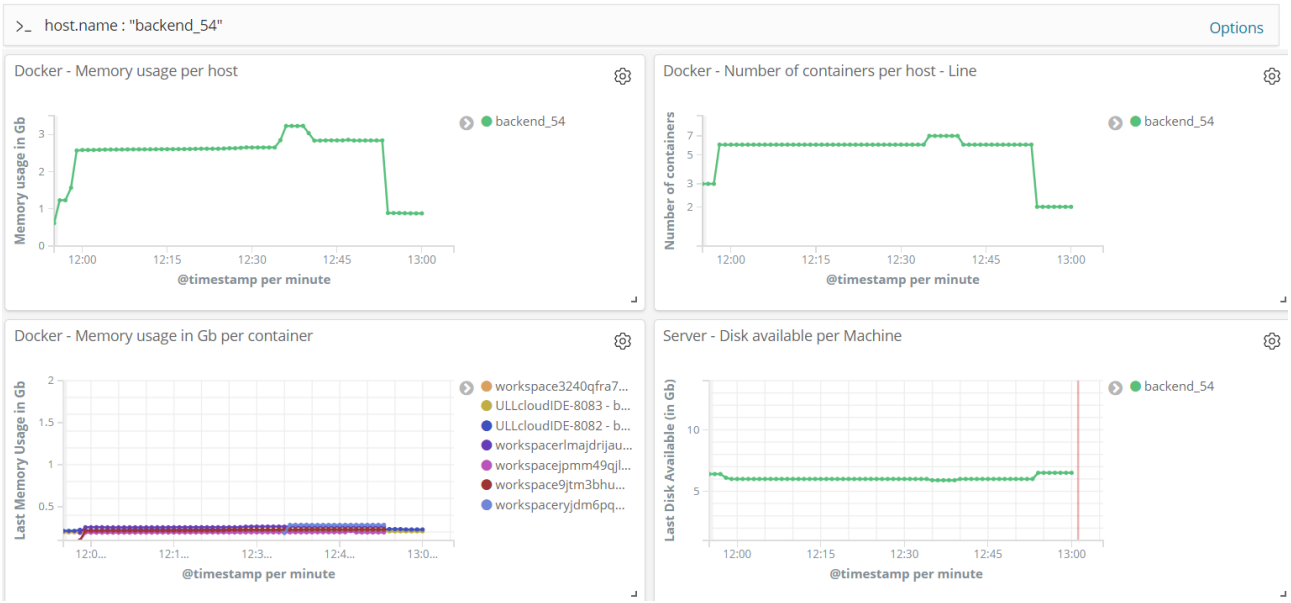

Figura 4.2: Dashboard con el error visualizado en Kibana

Como se puede observar en la figura 4.2 el error no tiene que ver con el uso de RAM o de CPU, ya que ambos parámetros se encuentran en rangos normales de uso, CPU <1 % y RAM algo por encima de 1GB, valores que son totalmente manejables. Tampoco con el número de contenedores, ya que se mantuvo constante el número de contenedores en el lapso de tiempo analizado.

Es por todo lo anterior, por lo que esta opción se descartó y se pasó a analizar una nueva idea. El problema podía estar en algún proceso que estuviera eliminando los contenedores Docker cada cierto tiempo.

Analizando la monitorización un poco más en profundidad se observó que este error ocurría cada hora exactamente, es decir, si a las 12:02:03 había dado el error, a las 13:02:03 el error volvía a aparecer. Por ello se procedió a analizar todo el código creado en el TFG que da origen a este proyecto. En este código se encontró el origen del problema:

cleandockerimages.sh

```
1  echo $1 | sudo -S su
2
3  sudo docker rmi $(sudo docker images | awk '$1 !~ /(alpine)|(ecli
   pse\/che$)|(eclipse\/che-action)|(eclipse\/che-test)|(eclipse\/ch
   e-server)|(eclipse\/che-init)|(eclipse\/che-dir)|(eclipse\/che-mo
   unt)|(eclipse\/che-ip)|(docker\/compose)/ {print $3}')
```

Figura 4.3: Script cleandockerimages.sh

El script de la figura 4.3 elimina todas los contenedores de Docker que no sean de un tipo específico , con el fin de ir limpiando los recursos no utilizados. El problema es que este borrado de contenedores afecta a los contenedores de Eclipse Che, programa que utilizan los alumnos para la realización de sus prácticas de programación.

Se decidió por tanto eliminar las llamadas a este script. Con ello se resolvió el problema principal de uso de la aplicación.

Capítulo 5

Análisis del sistema y recomendación de mejoras

En este capítulo se realizará un análisis del sistema para posteriormente proponer una serie de mejoras tal como se define en el cuarto objetivo del proyecto.

5.1. Asignación de recursos

Problema

Analizando los recursos necesarios para cada uno de los workspaces (entornos de trabajo, en este proyecto cada workspace corresponderá a una práctica de una asignatura) de los que puede hacer uso un alumno y los recursos con los que cuenta cada uno de los servidores que los contienen, nos encontramos con un problema que tarde o temprano aparecerá en un entorno de producción.

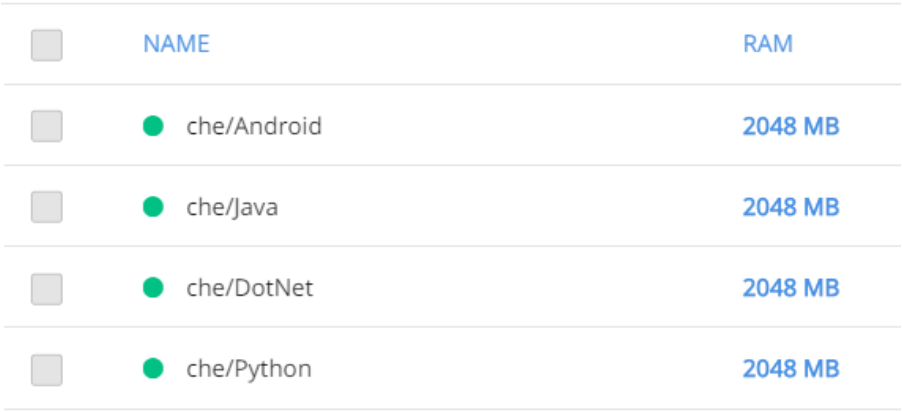

Figura 5.1: Recursos por Workspace

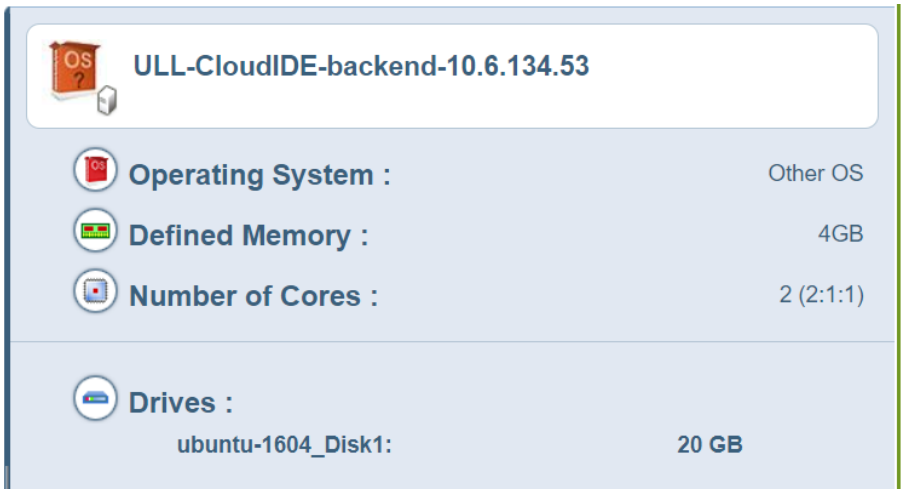

Figura 5.2: Recursos por Servidor

Como se puede observar existe una gran incongruencia en la asignación de ambos recursos. Un alumno puede crear y ejecutar cuántos workspaces quiera sin ningún tipo de limitación, si bien es cierto que el tamaño de la memoria RAM que aparece no es el que se usa, sino el máximo permitido. Ésto podría llevar a un escenario donde 4 workspaces estén utilizando 8GB de RAM mientras que el servidor sólamente tenga disponibles 4GB para ello, por lo que no sería posible ejecutar esos 4 workspaces. Ésto daría errores no controlados y de difícil manejo.

Recomendación de mejora

Es por ello que hay que realizar modificaciones, bien para limitar el número de workspaces por usuario, la memoria RAM máxima dedicada a cada uno de ellos, ambas opciones, y/o añadir más memoria RAM a los servidores.

Todas las modificaciones que afecten a Eclipse Che pueden hacerse a través de un fichero de configuración que Eclipse Che nos provee, ***CHE.ENV*** [24]. CHE.ENV nos permite modificar prácticamente cualquier configuración imaginable en Eclipse Che, desde añadir proxys, pasando por el número máximo de usuarios que pueden hacer uso de un workspace, hasta la máxima RAM de la que puede hacer uso cada workspace.

La única modificación que debe hacerse en la creación de los servidores es la asignación de RAM.

Para dar una propuesta de mejora de la actual configuración se han realizado las siguientes asunciones:

- Los proyectos que se realizarán en Eclipse Che, en principio, no requerirán gran cantidad de procesamiento, por lo que con **512MB de RAM por workspace** debería de ser suficiente, aunque en algunos casos se podría necesitar de más procesamiento.
- Se permitirán como **máximo 10 workspaces por asignatura**, ya que cada workspace suele corresponder a una práctica y por regla general ninguna asignatura tiene más de 10 prácticas.
- Como máximo se permitirá tener **en ejecución dos workspaces**, con lo que reduciremos la cantidad de RAM consumida.
- Se reservará como mínimo **1024 MB de RAM para el Sistema Operativo**.
- El número máximo de alumnos por servidor será de entre 2 y 3 para no sobrecargar al sistema operativo.

Con todo ello se ha elaborado la siguiente tabla de recomendación de configuraciones:

Configuraciones recomendadas					
Configuración	1	2	3	4	5
Mejoras(1)	Mejora Workspaces	Mejora SO - Usuarios	Mejora Workspaces	Mejora SO y CloudIDE	Mejora Workspaces - SO - Usuarios
Mem.CloudIDE (2)	512MB	512MB	512MB	1024MB	512MB
Usuarios(3)	2	3	2	3	3
Work.Simul. (4)	2	2	2	2	2
Mem.Work. (5)	512MB	512MB	1024MB	512MB	850MB
SO(6)	1024MB	1536MB	1024MB	2048MB	1556MB
Total(7)(8)	4GB	6GB	6GB	8GB	8GB

Cuadro 5.1: Posibles configuraciones CHE.ENV y Servidores

(1) - Mejora conseguida respecto a la configuración actual.
(2) - Memoria asignada a CloudIDE.
(3) - Número máximo de usuarios por servidor.
(4) - Número máximo de workspaces ejecutándose de forma simultánea.
(5) - Memoria máxima de la que puede hacer uso un workspace.
(6) - Memoria RAM dedicada al Sistema Operativo.
(7) - Memoria RAM que se debe asignar a cada servidor para poder hacer frente a la configuración seleccionada.
(8) - Fórmula aplicada: (Memoria CloudIDE * Usuarios) + Usuarios * (Workspaces simultáneos * Memoria Workspaces) + S.O

La elección de la configuración más adecuada no es sencilla y dependerá de varios factores, como el tipo de asignatura, si requiere mucha cantidad de procesamiento de datos, memoria máxima que nos provee el STIC para nuestros servidores, etc... Es por ello que esta decisión debe tomarse analizando previamente los requisitos de la asignatura.

5.2. Nombre Workspace

Problema

Los nombres de los workspaces actualmente no están bien formados (apareciendo incluso un valor null en ellos) y no son nada identificativos, por lo que si tenemos que analizar posibles problemas en ellos la tarea puede ser más complicada que si nos encontramos con nombres claros que indiquen qué propósito tienen estos workspaces.

```
CONTAINER ID        NAME
dc3f23ee5905        ULLcloudIDE-8082
ab3b6e6dd168        ULLcloudIDE-8083
2a83b1e88845        workspaceryjdm6pqhcufjbhx_null_che_dev-machine
1436a8efa9d0        workspacerlmajdrijauwdrol_null_che_dev-machine
b45bd4489492        workspacejpmm49qjld9gclv7_null_che_dev-machine
```

Figura 5.3: Nombres Workspaces

Recomendación de mejora

Se debe realizar una modificación a la hora de la creación de éstos workspaces, siguiendo por ejemplo este patrón:

```
nombre del workspace_ + nombre del
   servicio_ + usuario
```

Dando este resultado para un workspace *Android*, servicio *empotrados* y usuario alu0100506730:

```
android_empotrados_alu0100506730
```

5.3. Restricciones de base de datos

Problema

La base de datos que gestiona toda la aplicación no cuenta actualmente con ningún tipo de restricción de manera que es posible añadir datos incongruentes, eliminar datos que dejarían la aplicación en un estado no correcto, etc...

Recomendación de mejora

Es por ello que creo necesario añadir al menos unas restricciones básicas siguiendo los siguientes ejemplos:

1. No se puede asignar a un servidor un servicio que no exista.
2. No se puede añadir un servidor cuya IP no esté en el rango de IPs válidas.
3. Si se elimina un servicio se tienen que eliminar todas las asignaciones de usuarios que dependen de ese servicio.

5.4. Refactorización del código

Problema

El fichero ***index.js*** que maneja toda la lógica de ULL-CloudIDE cuenta con algo más de 2600 líneas de código. Esto lo hace prácticamente indescifrable a la hora de encontrar posibles errores e inmantenible.

```
46    var bloqueo_tablas = "LOCK TABLES VMS WRITE, VMS as v1 READ, Ovirt_Pendientes_Up_AddStart WRITE, Ovirt_Pendientes_Up_AddStart as
        ovpuas READ, Ultima_conexion WRITE, Ultima_conexion as uc READ, Eliminar_servicio_usuario WRITE, Eliminar_servicio_usuario as
        esu READ, Eliminar_servicio WRITE, Eliminar_servicio as es READ, Servicios WRITE, Servicios as s1 READ, Matriculados WRITE,
        Matriculados as m1 READ, Ovirt WRITE, Ovirt as ov READ, Ovirt_Pendientes WRITE, Ovirt_Pendientes as ovp READ, Banco_ip WRITE,
        Banco_ip as bip READ, Firewall WRITE, Firewall as f1 READ, Pendientes WRITE, Pendientes as p1 READ, Asignaciones WRITE,
        Asignaciones as a1 READ, Cola WRITE, Cola as c1 READ";
```

Figura 5.4: Ejemplo de código complicado de manejar

Recomendación de mejora

Debido a lo anteriormente menciado creo necesario realizar una refactorización de este fichero, dividiendo su funcionalidad en ficheros separados. Con ello se consigue mejorar la comprensión del código facilitando así en un futuro añadir nuevas funcionalidades y su mantenibilidad.

Capítulo 6

Conclusiones y líneas futuras

En este capítulo se abordarán las conclusiones extraídas de la realización de este TFM así como las posibles líneas futuras para su mejora.

6.1. Conclusiones

- Este proyecto ha sido realizado utilizando como caso de uso un proyecto anterior, con las limitaciones y ventajas que ello conlleva.
- Se ha realizado una búsqueda, análisis, y elección de las herramientas que más se ajustaban a los requisitos marcados y a las restricciones que el proyecto tenía para cumplir unos objetivos claros.
- Se ha implementado un sistema de monitorización que permite de manera sencilla, rápida y transparente comprobar el estado actual de la aplicación, así como la búsqueda de errores, posibles problemas futuros, cuellos de botella, etc... todo ello de forma centralizada.
- Este sistema de automatización no requiere ningún tipo de modificación para añadir nuevos servidores a su lista de monitorización, sino que se ha automatizado el envío de información desde los servidores hasta la plataforma de monitorización.
- Ha sido encontrado y solucionado el problema principal con el que contaba la aplicación, que fue detectado gracias a que la plataforma de monitorización nos permitió realizar un análisis rápido y efectivo del problema.
- En el transcurso del proyecto se ha ido analizando la aplicación sobre la que se basa este TFM a medida que se ha ido desarrollando la solución propuesta. De esta manera y posteriormente se han realizado múltiples sugerencias de mejora.

6.2. Líneas futuras

Los dos principales hitos de este proyecto han sido alcanzados, implementación del sistema de monitorización y la resolución de errores, pero la aplicación aún puede dar cabida a la implementación de numerosas líneas de trabajo, que creo que se pueden dividir en dos líneas principales, escalabilidad y línea general.

6.2.1. Escalabilidad

Se define la escalabilidad como la capacidad para estar preparado para hacerse más grande sin perder calidad en los servicios ofrecidos [25].

El tamaño actual de usuarios de esta aplicación es pequeño, pero su tamaño potencial es bastante mayor, es por ello, que creo que trabajar en prepararse para un posible incremento de usuarios conservando la calidad de los servicios es una de las líneas de trabajo más interesantes. A continuación se enumeran algunas de las posibilidades en este aspecto:

- Hacer uso de un buffer entre nuestros servidores y el Stack ELK de manera que permita un mejor manejo de los logs comprobando la disponibilidad del Stack ELK evitando que un exceso de flujo de datos cree problemas. Para ello puede hacerse uso de herramientas como Redis, Kafka o RabbitMQ.
- El Stack ELK permite configuraciones de alta disponibilidad (bajos tiempo sin servicio), tanto haciendo uso de Stacks enteros como de varias instancias de Elasticsearch, o Kibana, de esta manera se conseguiría que en caso de que alguno de los Stacks ELK dejara de estar disponible las peticiones fueran reenviadas al Stack que se encuentra operativo.

6.2.2. Líneas generales

- Actualmente cuando un workspace es levantado y no se cuenta con la última versión de la imagen Docker en el servidor, esta imagen es descarga directamente del repositorio Docker (DockerHub), donde ocupan entorno a 700Mb, por lo que esta descarga consume bastante tiempo, además de que ocupa una gran parte del almacenamiento de los servidores.

  Es por esto mismo que creo que sería muy interesante y ventajoso montar un repositorio local en la carpeta compartida, de manera que una vez descargada la imagen, esta, sea accesible para los demás servidores. De esta manera, en vez descargar cada imagen cada vez que un servidor la requiera, se obtenga la imagen del repositorio local, con el consiguiente ahorro de tiempo.
- Puesto que la cantidad de posibilidades que nos provee el fichero CHE.ENV es enorme, creo que se tiene aún grandes capacidades de mejoras en este aspecto, aparte de las ya especificadas en el capítulo anterior.
- Estructuración del código general de la aplicación para hacerlo más mantenible y entendible.
- Dada la cantidad de posibles gráficas que nos provee Kibana, pueden realizarse nuevos *Dashboards* para verificar otras partes de la aplicación que no se hayan tenido en cuenta en el presente proyecto.

Capítulo 7

Summary and Conclusions

In this chapter the conclusions and the future work of the project will be presented.

7.1. Conclusions

- The project was developed using as a use case from a previous project, with the limitations and advantages that it implies.
- The following processes have been carried out to meet the project requirements: a search, analysis and selection of different monitoring tools.
- A monitoring system was deployed, providing a simple, fast and transparent view of the current status of the application. The search for errors, possible future problems, and bottlenecks, were managed with a centraliced system.
- The automation system does not require any modification to add new servers in the monitoring list, furthermore the transfering of information from the servers to the monitoring platform has been automated.
- The main detected problem was solved using the new monitoring tool. The monitoring platform allows us to perform a quick and effective error analysis.
- During the developmen of the project, the application on which this TFM is based has been deeply analysed. Multiple suggestions for improvements have been proposed using the collected data.

7.2. Future work

The two main milestones have been completed: the monitoring system implementation and the finding and fixing errors. But there are several aspects to be considered for the future. These aspects can be categorized into scalability and general purposes:

7.2.1. Scalability

Scalability is defined as the ability to be prepared to become bigger without losing quality in the services offered. [25].

The current amount of users in the application is small, but the potential size is much larger. I believe that working to prepare for a possible increase of users while preserving the quality of services is one of the most interesting lines of future work. Here are some possibilities:

- Buffering the logs between the ELK Stack and our servers avoiding overflow of data that can be problematic. For this purpose, tools such as Redis, Kafka or RabbitMQ can be used.
- The ELK Stack allows high availability of configurations (low time without service), using several instances of the complete ELK Stack, Elasticsearch, or Kibana. Doing that we will achieve that in case any of the ELK Stacks ceases to be available the requests would be forwarded to the Stack that is operative.

7.2.2. General purposes

- Currently when a workspace is running and the latest version of the Docker image is not available on the server, these images are downloaded directly from the Docker repository (DockerHub). The image size is around 700MB of storage and takes a big amount of time to be donwloaded.

  For this reason, I think it would be very interesting and advantageous to set up a local repository in the shared folder. When the image is downloaded it will be accessible to other servers, reducing time waste and disk space.
- There is even more room for improvements in the CHE.ENV file apart of the already mentioned in previous chapters.
- Refactoring the application code base can be possible in order to make it more maintainable.
- Because of the amount of possible graphics provided by Kibana, new *Dashboards* can be developed to verify different parts of the application that have not been taken into account in this project.

Capítulo 8

Presupuesto

Presupuesto	
Recurso	**Dinero**
N° de horas de trabajo: 400 - (20€/hora)	8000€
Equipo de desarrollo	No se ha tenido en cuenta, ya que ya se contaba con el equipo necesario
Infraestructura	No se ha tenido en cuenta, ya que ya existía
Total	**8000€**

Cuadro 8.1: Presupuesto proyecto

Bibliografía

[1] Statista. Pronóstico de los gastos en tecnologías de la información en todo el mundo desde 2014 hasta 2019 (en miles de millones de dólares estadounidenses). https://es.statista.com/estadisticas/634249/prevision-del-gasto-mundial-en-tecnologias-de-la-informacion--2019. Online; Accessed: 2019-05-15.

[2] Observatorio Nacional de las telecomunicaciones y de la SI. *Informe Anual del Sector TIC y de los Contenidos en España*. Gobierno de España, 2018.

[3] Elasticsearch B.V. Flujo de trabajo de logstash. https://www.elastic.co/guide/en/logstash/2.0/advanced-pipeline.html. Online; Accessed: 2019-06-25.

[4] Logz.io. Flujo de trabajo stack elk. https://logz.io/learn/complete-guide-elk-stack/. Online; Accessed: 2019-06-25.

[5] Sébastien Pujadas. Elasticsearch, logstash, kibana (elk) docker image documentation. https://elk-docker.readthedocs.io/. Online; Accessed: 2019-06-26.

[6] LLC. Nagios Enterprises. Nagios. https://www.nagios.org/. Online; Accessed: 2019-06-30.

[7] Wikipedia. Splunk. https://es.wikipedia.org/wiki/Splunk. Online; Accessed: 2019-06-30.

[8] Wikpedia. Elk. https://es.wikipedia.org/wiki/Elasticsearch. Online; Accessed: 2019-06-26.

[9] Logz.io. These 15 tech companies chose the elk stack over proprietary logging software. https://logz.io/blog/15-tech-companies-chose-elk-stack/. Online; Accessed: 2019-06-26.

[10] Graylog. Graylog. https://www.graylog.org/. Online; Accessed: 2019-06-26.

[11] Mark D Hill. What is scalability? *Computer Architecture News*, 2(18), 1990.

[12] E Turban, D King, J Lee, and D Viehland. *Electronic Commerce A Managerial Perspective*. Prentice-Hall, 5 edition, 2008.

[13] Elasticsearch B.V. Página oficial stack elk. https://www.elastic.co. Online; Accessed: 2019-05-15.

[14] Elasticsearch B.V. Introducción a kibana. https://www.elastic.co/es/webinars/getting-started-kibana. Online; Accessed: 2019-06-15.

[15] Inc. Red Hat. ¿qué es docker? https://www.redhat.com/es/topics/containers/what-is-docker. Online; Accessed: 2019-06-25.

[16] Elasticsearch B.V. Introducción a beats. https://www.elastic.co/es/products/beats. Online; Accessed: 2019-06-25.

[17] Elasticsearch B.V. Introducción a logstash. https://www.elastic.co/es/products/logstash. Online; Accessed: 2019-06-25.

[18] Elasticsearch B.V. Introducción a elasticsearch. https://www.elastic.co/es/products/elasticsearch. Online; Accessed: 2019-06-25.

[19] Wikipedia. Elasticsearch. https://es.wikipedia.org/wiki/Elasticsearch. Online; Accessed: 2019-06-25.

[20] Microsoft. ¿qué es iaas? https://azure.microsoft.com/es-es/overview/what-is-iaas/. Online; Accessed: 2019-06-26.

[21] Docker Inc. Dockerhub. https://hub.docker.com/. Online; Accessed: 2019-06-26.

[22] Docker Inc. Imagen elk. https://hub.docker.com/r/sebp/elk/. Online; Accessed: 2019-06-26.

[23] Elasticsearch B.V. Grok filter plugin. https://www.elastic.co/guide/en/logstash/current/plugins-filters-grok.html. Online; Accessed: 2019-06-26.

[24] Inc Eclipse Foundation. Configuing che on docker. https://www.eclipse.org/che/docs/che-6/docker-config.html. Online; Accessed: 2019-06-26.

[25] Wikipedia. Escalabilidad. https://es.wikipedia.org/wiki/Escalabilidad. Online; Accessed: 2019-06-26.