Facultad
de
Ciencias

INSTALACIÓN E INTEGRACIÓN DE HERRAMIENTAS DE MONITORIZACIÓN CON SISTEMA DE ALARMAS E INVENTARIO DE EQUIPOS

(Installation and integration of monitoring applications with alarm system and host inventory)

Trabajo de Fin de Grado
para acceder al

GRADO EN INGENIERÍA INFORMÁTICA

Autor: Marcos Ateca Domínguez

Director: Jose Luis Bosque Orero

Co-Director: Roberto Hidalgo Cisneros

Septiembre - 2016

Agradecimientos

A mi familia, que me ha apoyado durante estos cuatro años en los buenos momentos y en los no tan buenos.

A mis amigos de la facultad, que gracias a ellos he conocido a unas personas magníficas que han hecho de estos años algo más divertido.

A mi grandísimo amigo Tito, por ser la persona que más me ha ayudado en los momentos duros y por ser la persona de la que más he aprendido.

A CIC y al departamento de sistemas por permitirme realizar este proyecto allí, del cual he aprendido muchísimo, con especial mención a Jose y a Jose Luis por lo grandes compañeros que han sido.

A mi director del proyecto, Jose Luis Bosque por la grandísima ayuda recibida para realizar la memoria y guiarme por el buen camino.

Resumen

En las empresas cada vez hay más equipos, ya sean máquinas físicas, como servidores, o máquinas virtuales. Es debido a esto por lo que se diseña este TFG, que da una solución de monitorización completa de equipos a la empresa CIC, incluyendo un sistema de alarmas y un inventario de equipos. Esta monitorización completa se refiere a una monitorización de los archivos de log que generan los equipos y por otra parte al comportamiento de dichos equipos, como puede ser la carga de CPU o la memoria RAM ocupada. Además, se incluye un sistema de alarmas relacionado con la monitorización y un inventario de equipos.

El proyecto cuenta con cinco herramientas ya desarrolladas. En concreto CIC cuenta con dos de ellas ya instaladas, por lo que es tarea de este proyecto instalar las otras tres y finalmente integrarlas, ya que cada una por sí sola no hace la función de monitorización completa.

Al final del proyecto, con las cinco herramientas integradas, se cuenta con un paquete de monitorización, sistema de alarmas e inventario de equipos muy potente, que podría ser usado como producto para ser vendido a otras empresas interesadas en este sistema.

Palabras clave: maquinas, virtuales, monitorización, integración, alarmas, inventario.

Abstract

In companies there are more and more hosts, whether physical machines such as servers, or virtual machines. It is because of this that it is designed this TFG, which gives a complete monitoring solution of hosts to CIC company, including an alarm system and a host inventory. This complete monitoring refers to monitoring log files generated by the hosts and on the other hand the behaviour of the hosts, such the CPU load or RAM occupied.

The project has five already developed tolos. Specifically CIC has two of then already installed, so it is the task of this project to install the other three and finally integrate them, because each alone doens't the complete monitoring feature.

At the end of the Project, with the five integrated tools, it has a monitoring packet, alarm system and host inventory very powerful that could be used as a product to be sold to other companies interested in this system.

Keywords: virtual, machines, monitoring, integration, alarms, inventory.

ÍNDICE

ÍNDICE DE FIGURAS

CAPÍTULO 1

1 INTRODUCCIÓN

1.1 Virtualización y máquinas virtuales

"Llegó un momento en la historia de los microprocesadores x86 en que se estaban infrautilizando sustancialmente los ciclos de reloj. Entonces hubo que reinventar el concepto de Máquina Virtual (MV), que ya existía en los primeros sistemas IBM en los años 60. VMware, en 1999 sacó al mercado VMware Workstation y permitió la posibilidad de ejecutar diversos sistemas operativos (MVs) dentro de otro instalado en el equipo físico" [1] (servidor virtualizado).

Desde entonces, tanto las pequeñas y medianas empresas como los grandes datacenters se han aprovechado de esta tecnología, provocando que surjan incluso nuevas formas de negocio como el cloud computing. Básicamente, la *virtualización* consiste en un software llamado hypervisor que permite aislar y gestionar los diferentes sistemas operativos, y sus respectivos recursos, que se instalen sobre un mismo servidor físico. De este modo, todas las máquinas virtuales que estén instaladas en un mismo servidor físico comparten los recursos físicos que de otro modo quedan infrautilizados, y tendrán un número de CPUs, una cantidad de memoria RAM y de disco y cada MV será totalmente independiente del resto de MVs.

Virtualizar aporta ventajas y posibilidades en la actualidad. Permite reducir costes en prácticamente todos los campos de administración de sistemas. Desde la instalación y configuración de equipos hasta procesos de copias de seguridad, monitorización, y gestión y administración de la infraestructura. Disminuye el número de servidores físicos necesarios y el porcentaje de desuso de los recursos de los que disponen, aumentando su eficiencia energética.

También ofrece la posibilidad de centralizar y automatizar procesos cuya administración normalmente consume mucho tiempo, pudiendo aprovisionar y migrar máquinas virtuales de una manera rápida, manteniendo alta la calidad del servicio y bajo el tiempo de respuesta ante una caída del mismo.

La Figura 1 muestra un esquema de cómo, sobre un servidor físico en el que corre un sistema operativo Linux o Windows, hay instalado un hypervisor VMware Server en el que se están ejecutando máquinas virtuales con diferentes sistemas operativos y son completamente independientes unas de otras.

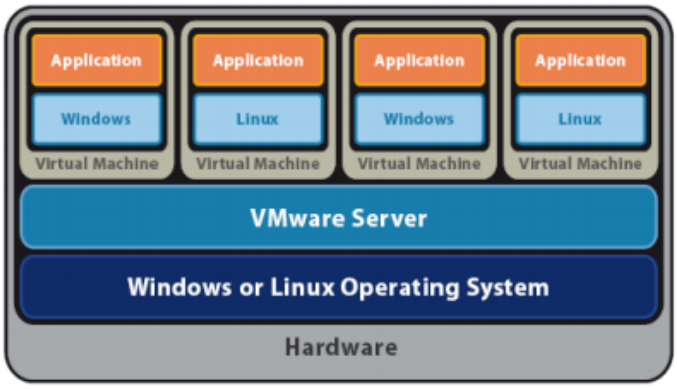


Figura 1: Esquema de un servidor con VMs[1]

1.1.1 Monitorización de máquinas virtuales

A día de hoy prácticamente cualquier empresa utiliza técnicas de virtualización. Gracias a ello, disponen de una arquitectura TI flexible que pueden dimensionar y ajustar según sus necesidades. Con un simple clic podemos agregar o eliminar una máquina virtual.

Debido al rápido crecimiento de las necesidades en computación y almacenamiento de las empresas, los administradores de las máquinas virtuales se encuentran con un arduo trabajo, que es la gestión eficiente de una infraestructura TI. De aquí nacen las herramientas de monitorización de máquinas virtuales y de sus servicios, mediante el envío de alertas y el aporte de información sobre tendencias de uso del sistema. Los datos proporcionados por estas herramientas permiten detectar errores o situaciones límite de funcionamiento, ayudan a la toma de decisiones, tanto automática como guiadas, o detectan anomalías de funcionamiento o seguridad que pueden disparar una alerta para que el administrador del sistema actué en consecuencia

¿Qué parámetros de una máquina virtual pueden ser interesantes para ser monitorizados? Entre ellos se encuentran el porcentaje de disco duro utilizado, la carga de trabajo de la CPU, el porcentaje de memoria RAM utilizado, las operaciones de entrada/salida, servicios, red, el consumo de energía o la temperatura a la que se está trabajando, que tiene un fuerte impacto en el consumo… Es por esto que debido a la gran cantidad de datos que generan cientos o miles de máquinas virtuales, es necesario automatizar todo el sistema de monitorización.

1.1.2 Monitorización de logs

Hasta ahora se ha hablado de la monitorización a nivel de máquina virtual, pero también podemos hablar de monitorización a nivel de archivos de log que generan las aplicaciones instaladas en las máquinas virtuales. Estos archivos de gran volumen contienen muchísima información que bien administrada pueden ser una fuente de datos muy útil para el negocio. Esto se debe a que los logs permiten analizar errores de programas y sistemas e incluso pueden servir para determinar el comportamiento de los usuarios. Es así por lo que es altamente necesario contar con una potente herramienta de análisis de logs, con el fin de que la empresa estudie sus sistemas en función de los millones de líneas que contienen estos ficheros.

En muchas ocasiones los logs se utilizan para encontrar problemas que puedan tener las aplicaciones. Pero se puede ir más allá y utilizar los archivos de log para recopilar información potencial como picos de uso de un sistema, horarios con mayor número de acceso de usuarios e incluso desde que país o región lo hacen, entre otros diversos y abundantes datos. Además, con la herramienta adecuada es posible llegar a recopilar todos estos logs en tiempo real.

1.1.3 Inventario de equipos

Por último, como ya se ha mencionado, la infraestructura TI puede crecer muy rápidamente por lo que también es interesante tener un inventario de los equipos que tenemos en nuestra infraestructura. Para ello es interesante tener una base de datos que contenga dicho inventario. Una solución es tener una CMDB (*Configuration Management DataBase*) que contenga los detalles relevantes a cada elemento y la relación entre ellos.

1.2 Monitorización y Gestión de MVs en grades instalaciones

La empresa Consulting Informático Cantabria (CIC), como cualquier empresa de hoy en día, utiliza la virtualización para ahorrar espacio físico. Como es una empresa en expansión, el número de máquinas virtuales que están en funcionamiento crece mes a mes.

El departamento de sistemas de esta empresa es, principalmente, el encargado de gestionar y administrar tanto los recursos físicos, como las MVs que el resto de departamentos utiliza para el desarrollo y prototipado de nuevas herramientas y aplicaciones. CIC necesita tener todas las MVs controladas, es decir, monitorizadas, ya que como empresa todo debe funcionar correctamente de cara a los clientes. Profundizando más en este problema, la empresa necesita que se controlen los archivos de log de dichas MVs monitorizadas, creados automáticamente por las diferentes aplicaciones que se ejecutan y además controlar el estado del sistema (CPU, RAM, disco…).

Además, debido al alto número de MVs en funcionamiento, es útil contar con un sistema de alarmas que alerte a los administradores de los sistemas del incorrecto funcionamiento de las MVs. Es por esto que CIC desea implementar dicho sistema de alarmas automático, que se disparará cuando algo no vaya bien, ya sea una aplicación (mediante los logs que la propia aplicación crea) o una máquina virtual en sí, si algo del sistema no funciona correctamente.

Por último, debido a este crecimiento, es necesario en cualquier empresa tener un sistema que almacene los datos de los servidores y MVs que hay ejecutándose en ellos ya que de esta forma cualquier persona de la empresa puede conocer fácilmente la infraestructura. Es por esto que CIC desea contar con una *Configuration Management DataBase* (CMDB) para poder tener un inventario de equipos y de esta forma que todos los trabajadores que se encargan de la infraestructura puedan conocer rápidamente el estado de ésta.

El departamento de sistemas de CIC se enfrenta en la actualidad con los problemas anteriormente descritos, y ha decidido crear un proyecto interno para tratar de resolverlos. Este es el punto de arranque del presente Trabajo de Fin de Grado, por lo que nos enfrentamos a un problema real que tiene una empresa y el objetivo de este TFG es solucionarlo mediante un proyecto de integración de herramientas software. Es decir, el proyecto no pretende desarrollar una nueva aplicación que resuelva este problema, sino analizar una serie de herramientas que solucionen los diferentes problemas planteados, e integrarlas para formar una única herramienta que aporte una solución flexible, económica y eficaz a todos ellos.

1.3 Desarrollo vs. Integración de software

Para solucionar el problema planteado en la sección anterior, el departamento de sistemas de ha optado por un proyecto de integración de herramientas software ya desarrolladas por terceros, alguna de ellas de software libre y otras como aplicaciones comerciales.

La integración de aplicaciones presenta una serie de ventajas para este proyecto concreto, frente al desarrollo de software:

- Existen multitud de herramientas de software libre que pueden ser elegidas para solucionar los problemas de la empresa sin invertir dinero, sólo trabajo en la búsqueda de éstas y tiempo en comprenderlas y desarrollar sus interfaces para integrarlas.
- Al elegir herramientas ya desarrolladas se ahorra tiempo en el desarrollo software de las herramientas que en el caso de hacerlo de esta manera se deberían realizar las fases de análisis de requisitos, diseño y arquitectura, implementación (programación), pruebas y documentación.
- CIC trabaja ya hace tiempo con dos herramientas. Una llamada **Elastic** que recoge logs de diferentes aplicaciones y los guarda en una base de datos no relacional. Otra llamada **Jira**, que es un sistema de gestión de proyectos e incidencias, la cual se desea que sea usada para implementar el sistema de alarmas. Esto puede ahorrar ligeramente el trabajo ya que estas herramientas ya están instaladas y en pleno funcionamiento. Además, Jira es una herramienta que usa toda la empresa por lo que el funcionamiento de dicha herramienta es ampliamente conocido. Por otra parte, puede llegar a ser una desventaja ya que da menos versatilidad debido a que se obliga a integrar estas dos herramientas con el resto, con las restricciones y limitaciones que esto ha provocado.

En un proyecto como el que se presenta, que se centra en integrar herramientas para construir un sistema complejo, es de gran importancia seleccionar las herramientas que más satisfactoriamente solucionen cada uno de los problemas expuestos. Seleccionadas las herramientas, el siguiente paso es estudiar su funcionamiento. Este paso requiere un arduo y complejo trabajo debido a que se debe estudiar las funcionalidades de las herramientas seleccionadas previamente y las diferentes tecnologías asociadas que estas usen, como por ejemplo un servidor web o una base de datos concreta.

Si se desarrollara un único software que resolviera todos los problemas planteados en este TFG, su desarrollo sería extremadamente complejo, ya que los objetivos requeridos son muy amplios y requeriría excesivo trabajo. Sin embargo, este tipo de desarrollo solo requería usar un lenguaje de programación y las librerías necesarias para poder realizarlo. En cambio, al usar herramientas ya desarrolladas se debe estudiar y familiarizarse con cada una de las tecnologías que éstas usen. Las herramientas son heterogéneas al estar desarrolladas por diferentes empresas o equipos y cada una usará la tecnología que mejor funcione en cada aplicación.

Otro trabajo unido a un proyecto de integración de aplicaciones es la necesidad de desarrollar una serie de interfaces posteriormente a la elección y estudio de las herramientas escogidas. La implementación de estas interfaces no es una tarea sencilla debido a que, como se ha comentado anteriormente, las herramientas estarán desarrolladas con diferentes tecnologías, lo que hace que haya que adaptar las diferentes interfaces que se necesiten a dichas tecnologías.

1.4 Objetivos

El objetivo principal de este Trabajo de Fin de Grado es el diseño e implementación de un sistema automático de monitorización, gestión de alertas e inventario de equipos en el departamento de sistemas de la empresa CIC. Este sistema debe ser capaz de monitorizar todos los parámetros necesarios según cada entorno y sistema. Además, debe permitir a los administradores del sistema definir una serie de condiciones de funcionamiento y si alguna de ellas no se cumple, el sistema lo indicará mediante una alerta. También definirá un sistema que permita realizar y mantener un inventario de equipos (servidores y máquinas virtuales) que hay en la empresa.

Se plantean los siguientes objetivos:

- Preparación de un paquete de instalación de acuerdo a las versiones de los sub-productos utilizados (excepto Elastic).
- Implementación de un sistema de monitorización de máquinas virtuales y descubrimiento automático de equipos en la red, que pasarán a estar monitorizados.
- Implementación de un sistema de alarmas que emita avisos en base a criterios pre-establecidos, mediante la definición de condiciones de funcionamiento de los distintos entornos y sistemas de la empresa.
- Integrar el sistema de alarmas con un gestor de incidencias para permitir la apertura automática de incidencias y el escalado al personal que corresponda dentro del flujo de resolución.
- Implementación de una Configuration Management DataBase (CMDB) que almacene un inventario de equipos de la empresa.

Actualmente el sistema cuenta con dos herramientas, Elastic y Jira, como se ha mencionado en el apartado 1.3. Esto supone en primer lugar buscar herramientas que cumplan con los objetivos descritos y que puedan ser integradas con las herramientas ya existentes en la empresa.

Una vez seleccionadas, se procede a su estudio y aprendizaje y a decidir finalmente si las herramientas cumplen los objetivos que se proponen.

En último lugar, se diseñan las interfaces de las herramientas para finalmente implementar todo el sistema integrado.

Una vez que se hayan alcanzado todos estos objetivos para culminar el desarrollo de este Trabajo de Fin de Grado, se podrá afirmar que el proyecto ha sido desarrollado con existo. De esta forma, se habrá conseguido resolver un problema real en una empresa (no académico), aplicando una gran cantidad de conocimientos de sistemas informáticos y redes (programación, sistemas operativos, virtualización, redes de computadores, etc…), adquiridos a lo largo de los cuatro años de estudio en la Universidad. Además de aplicar todos estos conocimientos, realizar un TFG en una empresa hace que también se adquieran un amplio abanico de conocimientos empresariales, como el trabajo en grupo y el aprendizaje de multitud de herramientas y sistemas, entre otros.

1.5 Estructura del documento

El presente Trabajo de Fin de Grado se ha estructurado en una serie de capítulos, que describen el trabajo realizado en este proyecto. Además de este capítulo introductorio se han definido los siguientes:

- El capítulo 2 se centra en presentar las características de las diferentes herramientas que componen el proyecto. Al ser un proyecto de integración de herramientas, el conocimiento previo de éstas resulta realmente importante ya que están completamente desarrolladas y hay que proceder directamente a su estudio funcional.
- El capítulo 3 se define exhaustivamente el problema de la necesidad de un sistema de monitorización con sistema de alarmas e inventario de equipos que tiene CIC. Asimismo, se definen los objetivos que se van a cumplir con el proyecto y finalmente el diseño de la solución para resolver el problema que plantea CIC.
- El capítulo 4 presenta cómo, tras analizar los objetivos, se implementa la solución realizando una serie de scripts que instalen cuatro de las cinco herramientas usadas explicando cada una de las tecnologías que cada una de las herramientas necesita para su correcto funcionamiento. En segundo lugar, en este capítulo se presentan las interfaces entre las herramientas y se incluyen diferentes ejemplos.
- Finalmente, en el capítulo 6 expone las conclusiones del proyecto y el trabajo futuro.

CAPÍTULO 2

2 HERRAMIENTAS BÁSICAS PARA EL DESARROLLO DEL PROYECTO

En un proyecto de integración de aplicaciones, es muy importante conocer de antemano qué es cada aplicación, qué características tiene y cómo funcionan. Es por ello que este capítulo se dedica exclusivamente a detallar las distintas herramientas.

2.1 Bases de datos

En esta sección se tratarán las bases de datos relacionales desde un punto de vista genérico, es decir, una breve introducción de qué es una base de datos y su estructura. Más adelante se hablará un poco más concretamente de los dos tipos de bases de datos relacionales que utilizan tres de las herramientas utilizadas en este proyecto: MySQL (Zabbix y Jira) y PostgreSQL (CMDBuild). También se hablará del concepto de bases de datos no relacionales y sus tipos ya que Elastic utiliza una base de datos de este tipo.

2.1.1 Introducción a las bases de datos relacionales

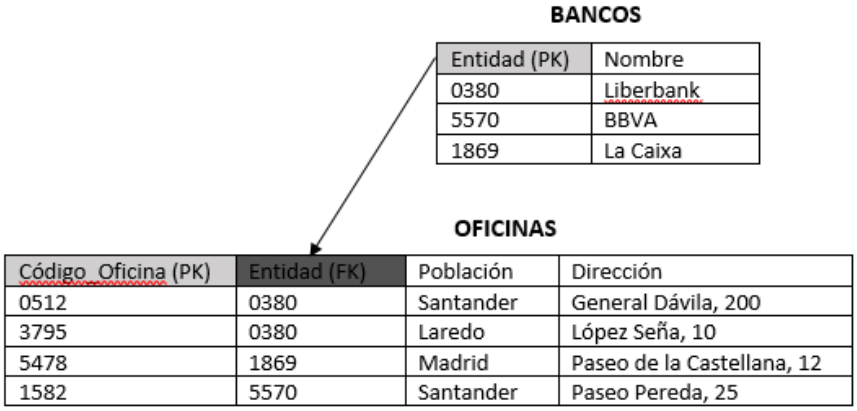
"Un *sistema gestor de bases de datos* (*SGBD*) consisten en una colección de datos interrelacionados y un conjunto de operaciones para acceder a dichos datos. La colección de estos datos suele ser denominada base de datos, que contiene información relevante. El objetivo principal de un SGBD es proporcionar una forma de almacenar y recuperar la información de una base de datos de manera práctica y eficiente." [2]

Una *base de datos* podría ser definida como una colección organizada de datos, relativa a un problema concreto, que puede ser compartida por un conjunto de usuarios/aplicaciones. Las bases de datos sirven para almacenar, controlar, consultar y actualizar grandes cantidades de información.

Las bases de datos cuentan con niveles de abstracción ya que existe una independencia física y lógica. Esto es, centrarse en determinar qué datos se han de almacenar y cómo se relacionan, olvidándose de los detalles internos.

"Una base de datos relacional consiste en un conjunto de tablas a cada una de las cuales se le asigna un nombre único. En general, una fila de una representa una relación entre un conjunto de valores." [2]

Toda tabla tiene una columna o conjunto de columnas que permiten identificar cada una de sus filas. A este conjunto se le llama *clave principal* (*Primary Key, PK*). Los valores de la PK no se pueden repetir. Unas tablas se refieren a otras mediante columnas que tengan el mismo tipo de dato en las dos tablas y se denomina *clave ajena* (*Foreign Key, FK*). En la Figura 2 se puede observar un ejemplo sencillo de lo desarrollado anteriormente.



BANCOS

Entidad (PK)	Nombre
0380	Liberbank
5570	BBVA
1869	La Caixa

OFICINAS

Código_Oficina (PK)	Entidad (FK)	Población	Dirección
0512	0380	Santander	General Dávila, 200
3795	0380	Laredo	López Seña, 10
5478	1869	Madrid	Paseo de la Castellana, 12
1582	5570	Santander	Paseo Pereda, 25

Figura 2: Ejemplo de tablas en una base de datos

Cada banco tiene un número único asociado. De esta forma, en la tabla donde están contenidas las oficinas, gracias a la FK, se puede obtener el nombre del banco al cual pertenece una oficina.

2.1.2 MySQL

MySQL es el servidor de bases de datos relacionales más popular. Como base de datos relacional, utiliza múltiples tablas para almacenar y organizar la información. MySQL fue escrito en C y C++ y destaca por su gran adaptación a diferentes entornos de desarrollo, permitiendo su interactuación con los lenguajes de programación más utilizados como PHP, Perl y Java y su integración en distintos sistemas operativos.

MySQL es Open Source, lo que significa que la persona que quiera puede usar y modificar MySQL. Cualquiera puede descargar el software de MySQL y usarlo sin pagar por ello. Inclusive, cualquiera puede descargar el código fuente y cambiarlo de acuerdo a sus necesidades. MySQL usa la Licencia Pública General (en inglés, GPL), para definir qué es lo que se puede y no se puede hacer con el software para diferentes situaciones.

El software de bases de datos MySQL consiste en un sistema cliente/servidor que se compone de un servidor SQL multihilo, varios programas clientes y bibliotecas, herramientas administrativas, y una gran variedad de interfaces de programación (APIs). [3]

2.1.3 PostgreSQL

PostgreSQL es un sistema de gestión de datos objeto-relacional (es orientado a objetos), distribuido bajo licencia BSD (menos restrictiva que GPL) y con su código fuente disponible libremente. Es el sistema de gestión de bases de datos de código abierto más potente del mercado y en sus últimas versiones no tiene nada que envidiarle a otras bases de datos comerciales.

PostgreSQL utiliza un modelo cliente/servidor y usa multiprocesos en vez de multihilos para garantizar la estabilidad del sistema. Un fallo en uno de los procesos no afectará el resto y el sistema continuará funcionando.

Su desarrollo comenzó hace más de 16 años, y durante este tiempo estabilidad, potencia, robustez, facilidad de administración e implementación de estándares han sido las características que más se han tenido en cuenta durante su desarrollo. PostgreSQL funciona muy bien con grandes cantidades de datos y una alta concurrencia de usuarios accediendo a la vez al sistema. [4]

2.1.4 Bases de datos no relacionales (NoSQL)

El término *NoSQL* se refiere a una amplia clase de sistemas de gestión de bases de datos que difieren del modelo clásico de gestión de bases de datos relacionales (MySQL, PostgreSQL, SQL Server, Oracle). Su principal característica es que no usan SQL como principal lenguaje de consultas. Los datos almacenados no requieren estructuras fijas como tablas, normalmente no soportan operaciones JOIN, ni garantizan atomicidad, consistencia, aislamiento y durabilidad. Suelen escalar bien horizontalmente, lo que quiere decir que al agregar más nodos, el rendimiento de la base de datos no relacional mejorará.

Los sistemas de bases de datos NoSQL crecieron con las principales compañías de Internet como Google, Amazon, Twitter y Facebook. Estas compañías se dieron cuenta que el rendimiento y sus propiedades de tiempo real de este tipo de bases de datos eran más importantes que la coherencia, en la que las bases de datos relacionales dedican un gran tiempo de proceso. [5]

Se pueden encontrar cuatro tipos diferentes [6]:

- **Orientadas a documentos**: son aquellas que gestionan datos semi estructurados (documentos). Estos datos son almacenados en algún formato estándar como XML, JSON o BSON. En esta categoría se encuentran bases de datos como MongoDB, CouchDB y la base de datos que usa Elastic, Elasticsearch.
- **Orientadas a columnas**: bases de datos pensadas para realizar consultas y agregaciones sobre grandes cantidades de datos. Funcionan de forma parecida a las bases de datos relacionales, pero almacenando columnas de datos en lugar de registros. En esta categoría se encuentran Cassandra y HBase.
- **Clave-valor**: guardan tuplas que contienen una clave y su valor. Cuando se quiere recuperar un dato, simplemente se busca por su clave y se recupera su valor. Aquí se encuentran bases de datos como DynamoDB y Redis.
- **Orientadas a grafos**: basadas en la teoría de grafos utilizan nodos y aristas para representar los datos almacenados. Son muy útiles para guardar información en modelos con muchas relaciones, como redes y conexiones sociales. En esta categoría se encuentran bases de datos como Infinite Graph y Neo4j.

2.2 Zabbix

2.2.1 Introducción

Zabbix es un software que monitoriza numerosos parámetros de una red, así como el estado de los servidores. Estos valores son almacenados en una base de datos MySQL que la herramienta utiliza para ofrecer informes y visualización en base a los datos almacenados. Zabbix usa un mecanismo de notificaciones que permite a los usuarios configurar alertas basadas en correo electrónico para cualquier evento. Todos los informes y estadísticas que ofrece, así como los parámetros de configuración son accedidos desde una interfaz web. Puede ser utilizado tanto por una organización pequeña, con pocos equipos, como por una organización grande con una gran cantidad de equipos. Zabbix está escrito y distribuido bajo licencia GPL (General Public License

versión 2). Esto significa que el código fuente puede ser libremente distribuido y está disponible para el público general. [7]

2.2.2 Características

A continuación, se describen las características más destacadas de la herramienta que se han utilizado en este proyecto.

2.2.2.1 Recolección de datos (monitores)

La recolección de datos se realiza mediante monitores. Un *monitor* es un proceso que recoge un dato de un equipo. En cada monitor se define qué tipo de dato se quiere recoger del equipo. Así, por ejemplo, el monitor *system.cpu.load* recopilará datos de la carga trabajo del procesador. Zabbix cuenta con un amplio abanico de monitores que permiten una completa monitorización del equipo. [8]

2.2.2.2 Iniciadores

Los *iniciadores* (triggers) son expresiones lógicas que evalúan los datos recolectados por los monitores y representan el estado actual del sistema. Mientras que los monitores se encargan de recolectar datos del sistema, los iniciadores se encargan de evaluar los datos recogidos. Las expresiones de los iniciadores permiten definir un umbral. Si se supera este umbral se considera que el sistema tiene un comportamiento problemático como, por ejemplo, si la carga de trabajo del procesador es mayor que el 80%. La expresión del iniciador se comprueba cada vez que el servidor Zabbix recibe un nuevo valor de cualquier monitor que use el iniciador. [9]

2.2.2.3 Notificaciones

No es útil estar continuamente mirando el estado de todos los iniciadores. Sería mucho más útil informar a un usuario (o grupo de usuarios) mediante una *notificación*, indicando la situación o el problema que ha ocurrido. Es la misma diferencia que la entrada/salida por encuesta o por interrupciones. Los iniciadores realizan un proceso de encuesta y la notificación puede ser enviada en cualquier momento (interrupción). Es por ello que el envío de notificaciones es una de las acciones principales ofrecidas por Zabbix. Las notificaciones pueden ser enviadas por diferentes medios como el correo electrónico, un mensaje de texto o incluso se pueden ejecutar scripts personalizados. [10]

2.2.2.4 Plantillas

Una *plantilla* es un conjunto de monitores e iniciadores. Como muchos equipos son idénticos o muy similares, pueden ser monitorizados con los mismos monitores e iniciadores. Para no tener que crear los mismos monitores e iniciadores en cada equipo, se puede crear una plantilla que agrupe una serie de elementos de monitorización que luego podrá ser usada en todos los equipos que se desee. Una vez que se ha asignado una plantilla a un equipo, este heredará todos los elementos de monitorización que contenga la misma. Por ejemplo, se pueden crear una plantilla para monitorizar equipos Windows y otra plantilla para monitorizar equipos Linux. Esto mismo se podría aplicar a plantillas que monitoricen servicios Apache, MySQL… [11]

2.2.2.5 Descubrimiento de red

El *descubrimiento automático de equipos en la red* permite acelerar el proceso de monitorización de los equipos. En vez de añadir uno a uno los equipos a la herramienta, el descubrimiento de red permite que Zabbix descubra y añada automáticamente los equipos para ser monitorizados.

Mediante reglas de descubrimiento de red, Zabbix escanea periódicamente rangos de IP predefinidos en las reglas. Cada una de las reglas de descubrimiento definidas es ejecutada por un único proceso. Cuando un equipo es descubierto, se pueden ejecutar acciones, las cuales pueden incluir enviar notificaciones, añadir o eliminar equipos, enlazar o desenlazar equipos de una plantilla, entre otras. Estas acciones son configuradas respecto a una serie de condiciones como puede ser el tiempo que el equipo lleva encendido o apagado, su IP… [12]

2.2.2.6 Acciones

Se ha hablado anteriormente del envío de notificaciones. Para que estas tengan lugar, es necesario configurar acciones. Las *acciones* son operaciones que se ejecutan cuando se dispara un iniciador o se descubre un equipo, como por ejemplo el envío de correo electrónico. [13]

2.2.2.7 Usuarios y grupos de usuarios

El acceso a Zabbix se realiza mediante la interfaz web con un nombre de usuario y una contraseña. Las contraseñas están encriptadas y almacenadas en la base de datos. En Zabbix hay diferentes tipos de usuario: [14]

- **Usuario de Zabbix:** solo tiene permiso para ver el estado de los equipos monitorizados. Los permisos a grupos de equipos para este tipo de usuario tienen que ser definidos explícitamente
- **Administrador de Zabbix:** tiene permiso para ver el estado de los equipos monitorizados y para usar los menús de configuración. Los permisos a grupos de equipos para este tipo de usuario tienen que ser estipulados explícitamente
- **Súper Administrador de Zabbix:** tiene permisos totales en la aplicación web.

Respecto a los permisos a equipos o grupos de equipos, no se puede dar permiso a un usuario para acceder a la configuración de un equipo o un grupo de equipos. Para poder dar este tipo de permisos, el usuario tiene que estar dentro de un grupo de usuarios y que este grupo de usuarios tenga los permisos de acceso a equipos o grupo de equipos. [15]

2.2.2.8 *Interfaz web*

Zabbix proporciona una *interfaz web* basada en PHP. Esta interfaz web permite al usuario administrar por completo la aplicación usando las funcionalidades descritas anteriormente, entre otras.

La Figura 3 muestra el panel de control de la interfaz web, donde se ve un resumen del estado del sistema. En este caso concreto, se están monitorizando 133 equipos que se encuentran en el grupo "Discovered hosts" y el estado de todos los equipos en correcto.

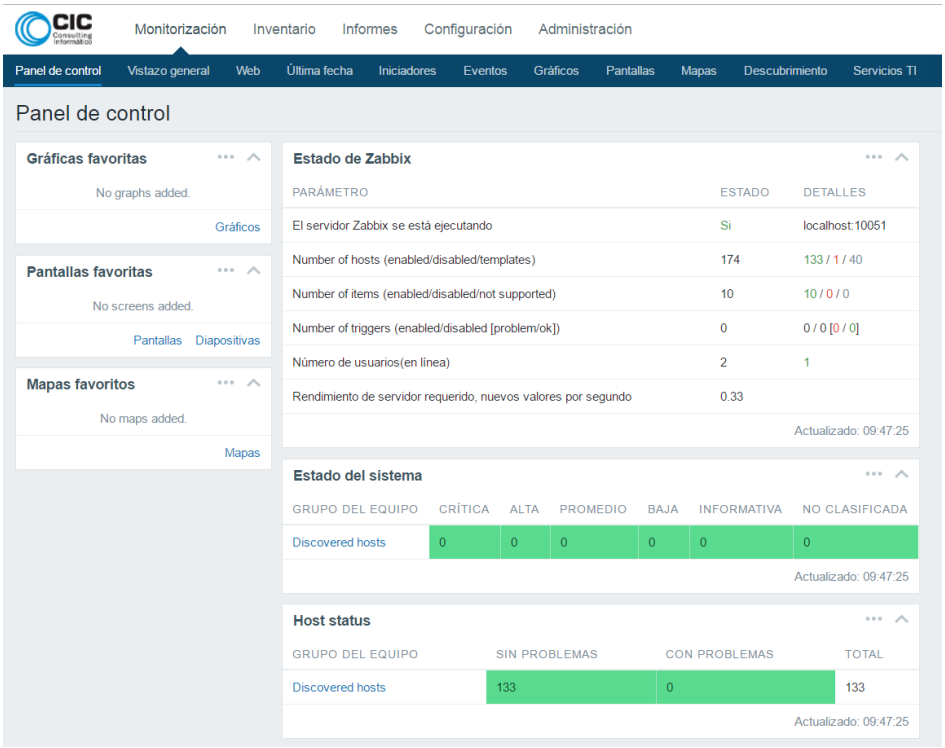


Figura 3: Panel de control de la interfaz web

La Figura 4 ilustra la pestaña "Configuración" donde se encuentran algunas de las características antes descritas: las plantillas, los equipos (añadidos mediante el descubrimiento de red), las acciones y el descubrimiento.

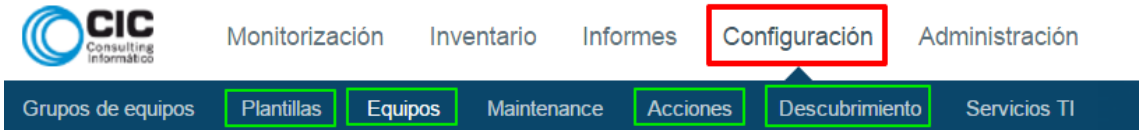


Figura 4: Configuración de plantillas, equipos, acciones y descubrimiento en la interfaz web

2.2.2.9 Servidor Zabbix

El *servidor Zabbix* es el proceso principal de la aplicación Zabbix y su misión principal es capturar los datos mediante encuesta (gracias a los agentes Zabbix). Este componente se encarga de almacenar toda la configuración, las estadísticas y los datos. También es el que se encarga de la configuración y envío de alertas. Zabbix está dividido en tres componentes: el servidor Zabbix, una base de datos y una interfaz web. Toda la configuración se guarda en la base de datos: equipos, plantillas, monitores, iniciadores, usuarios… Servidor e interfaz web interactúan con la base de datos. Por ejemplo, cuando se crea un equipo desde la interfaz web, este es almacenado en la base de datos. El servidor también consulta la base de datos para ejecutar los distintos monitores e iniciadores que se hayan creado previamente y almacenará los datos recogidos en la base de datos MySQL. El servidor Zabbix es un proceso que se ejecuta en todo momento con un usuario sin privilegios de superusuario. Se puede configurar dicho proceso para que ejecute como superusuario. El servidor Zabbix puede ser ejecutado en los siguientes sistemas: [16]

- Linux
- Solaris
- AIX
- HP-UX
- Mac OS X
- FreeBSD
- OpenBSD
- NetBSD
- SCO Open Server

2.2.2.10 Agentes Zabbix

Un *agente Zabbix* es desplegado en un equipo que se desee monitorizar y se encarga de recoger múltiples datos de dicho equipo. Mediante los monitores se recogen los datos que el agente ha obtenido del equipo. Los agentes Zabbix son muy eficientes debido a la utilización de llamadas nativas al sistema para recopilación de información estadística. Los agentes Zabbix están soportados en los siguientes sistemas: [17]

- Linux
- IBM AIX
- FreeBSD
- NetBSD
- OpenBSD
- HP-UX
- Mac OS X
- Solaris: 9, 10, 11
- Windows (desde el 2000)

En la Figura 5 se muestra como el servidor Zabbix realiza una petición al agente Zabbix instalado en un equipo monitorizado y como éste devuelve la información correspondiente.

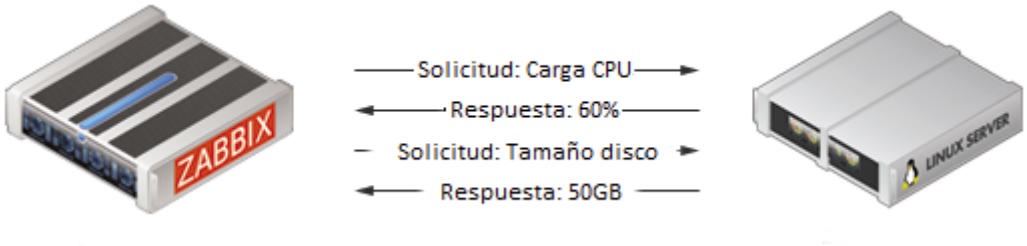


Figura 5: Interacción entre el servidor Zabbix y el agente Zabbix

2.3 CMDBuild

2.3.1 Introducción

Una *base de datos de gestión de la configuración* (*CMDB, Configuration Management Database*) es una base de datos que contiene detalles relevantes de cada elemento de configuración y de la relación entre ellos, incluyendo el equipo físico, software y la relación entre incidencias, problemas, cambios y otros datos del servicio de TI

CMDBuild es una aplicación web de código abierto cuyo objetivo es modelar y gestionar una CMDB. Un *objeto* se define como un equipo compuesto de una serie de atributos como pueden ser su nombre, el sistema operativo que tiene instalado, la arquitectura hardware, el número de serie, entre muchos otros. Su objetivo es facilitar a los trabajadores un control sobre todos los elementos de la empresa, pudiendo consultar su composición, ubicación, las relaciones entre distintos elementos y gestionar su ciclo de vida.

Las herramientas de configuración de CMDBuild permiten tener un CMS (Configuration Management System) completo para una buena gestión de los elementos. El CMS gestionará la configuración de elementos relacionados con la empresa, como por ejemplo las compras a proveedores como pueden ser ordenadores o servidores.

La base de datos que utiliza la herramienta es PostgreSQL y se puede personalizar gracias a un sistema de configuración adecuado para agregar clases de objetos, atributos a dichas clases, crear relaciones entre objetos, crear informes del contenido de la base de datos, etc.

2.3.2 Características

En esta sección se describen las características más destacadas de la herramienta que se han utilizado en este proyecto.

2.3.2.1 Modelado de la base de datos

La aplicación permite a los usuarios modelar la base de datos, de tal forma que se puede crear una CMDB totalmente personalizada partiendo de cero gracias a las siguientes características: [18]

- Clases que se convierten en tablas de la base de datos
- Atributos de las clases, que en la base de datos son las columnas de las tablas
- Dominios: las relaciones entre los distintos objetos

- Lookups: lista de valores que se asocian a atributos de las clases
- Herencia de clases

2.3.2.2 Gráficos de relaciones

Los *gráficos de relaciones* son una herramienta que muestra las relaciones entre los objetos. Esta herramienta resulta muy útil ya que se puede ver de forma rápida la relación que existe entre dos objetos de la base de datos. Un ejemplo: un servidor que contiene varias máquinas virtuales se rompe. Accediendo al gráfico de relaciones se puede ver qué máquinas hay alojadas en ese servidor y tomar las medidas oportunas de forma rápida. [19]

2.3.2.3 Módulo de gestión de datos

Este es uno de los dos módulos principales de CMDBuild el cual incluye las siguientes características: [20]

- Creación, modificación y eliminación de objetos
- Búsqueda de objetos usando filtros, con la posibilidad de crear y guardar filtros para búsquedas futuras
- Creación y modificación de relaciones entre objetos
- Generación de informes…

2.3.2.4 Módulo de administración

Es el segundo módulo de CMDBuild. Sus principales características son: [21]

- Modelado de datos
    - Creación de clases que alojarán los objetos
    - Creación y modificación de atributos de las clases
    - Creación de dominios, es decir, las relaciones entre objetos y sus atributos
- Gestión de usuarios y grupos
    - Creación de usuarios
    - Creación de grupos de usuarios
    - Definición de permisos y roles para cada grupo y para clase subconjunto de filas y columnas de cada clase
    - Definición de menús personalizados para los diferentes tipos de usuarios
- Ajustes generales

2.3.2.5 Conectores

Un *conector* es un proceso interno de la herramienta que recoge datos de una base de datos externa a CMDBuild y los vuelca a la herramienta.

El uso de conectores en CMDBuild resulta muy útil si se desea tener la CMDB conectada a otras fuentes externas y así poder volcar datos de estas fuentes externas a la base de datos de la CMDB o verificar la equivalencia entre la información almacenada en CMDB con la información proporcionada por fuentes externas. En CMDBuild existen tres tipos de conectores: [22]

- **Conector Básico**: configurable a través del lenguaje de trasformación XSLT que es capaz de enlazar la base de datos de la CMDB con una base de datos externa.
- Conector Avanzado: basado en una infraestructura configurable a través del lenguaje de scripts Groovy (para más información consultar [23] y [24]) que es capaz de comunicarse con sistemas externos utilizando un servicio web.
- **Conector simple**: configurable desde el módulo de administración de la interfaz web. Utilizado cuando la sincronización entre la CMDB y la base de datos externa es menos

compleja. Se crea una correspondencia directa entre las entidades que se quieren actualizar, es decir, una columna de una tabla de la base de datos externa se corresponderá con una columna de una tabla (clase) de la base de datos de la CMDB

2.3.2.6 Histórico

Cada objeto tiene un *histórico* de los cambios que los usuarios han realizado en sus atributos. Desde el módulo de gestión de datos se puede consultar este histórico. Para ello se accede al objeto concreto del cual se desee conocer los cambios que se hayan realizado en sus atributos. Este histórico también almacena y muestra qué usuario ha realizado el cambio. [25]

2.4 Jira

2.4.1 Introducción

Jira es una aplicación basada en una interfaz web para el seguimiento de errores e incidencias, así como para la gestión de proyectos. Inicialmente Jira se utilizó para el desarrollo de software, sirviendo de apoyo para la gestión de requisitos, seguimiento del estado y más tarde para el seguimiento de errores. Asimismo, Jira puede ser utilizado para la gestión y mejora de procesos, gracias a sus funciones para la organización de flujos de trabajo. Toda la información manejada por Jira se almacena en una base de datos MySQL. Jira tiene una versión de pruebas, pero la herramienta en sí es de pago.

2.4.2 Conceptos

En esta sección se describen los conceptos necesarios para entender la funcionalidad básica de la herramienta Jira.

2.4.2.1 Proyecto

Un *proyecto* es un conjunto de *incidencias* y está definido en base a los requisitos de la empresa. Por ejemplo, un proyecto de Jira puede ser de desarrollo de software, un sistema de asistencia… Cada proyecto tiene un nombre (p.e. Trabajo Fin de Grado) y una clave (p.e. TFG).

"Diferentes organizaciones usan Jira para realizar un seguimiento de diferentes tipos de incidencias. Dependiendo de cómo tu organización esté usando Jira, una incidencia puede representar un bug en un software, una tarea de un proyecto, un formulario de solicitud de licencia…" [26]

Existen diferentes tipos de incidencias. Las incidencias por defecto son: "Error", "Mejora", "Nueva funcionalidad", "Tarea" y, por último, el administrador puede crear incidencias personalizadas que se ajusten al proyecto.

Todas las incidencias tienen una prioridad asociada que por defecto son: "Highest", "High", "Medium", "Low" y "Lowest". Al igual que en el caso de los tipos de incidencias, las prioridades también son personalizables.

Cada incidencia tiene un estado, que indica cómo se encuentra la incidencia en un momento dado de su ciclo de vida. En el apartado siguiente se puede ver de forma más detallada el flujo de trabajo de una incidencia. El flujo de trabajo también es personalizable.

2.4.2.2 Flujo de trabajo

Un *flujo de trabajo* es un conjunto de estados y transiciones que una incidencia puede tener en su ciclo de vida. La Figura 6 muestra un diagrama del flujo de trabajo que puede tener una incidencia. En dicha figura se puede ver los estados que puede tener una incidencia: "Abierta", "En Progreso", "Cerrada", "Resuelta" y "Reabierta". Además, las flechas que hay entre los distintos estados son las acciones que había que realizar para pasar de un estado a otro. Una incidencia siempre empieza con el estado "Abierta" y acaba con el estado "Resuelta".

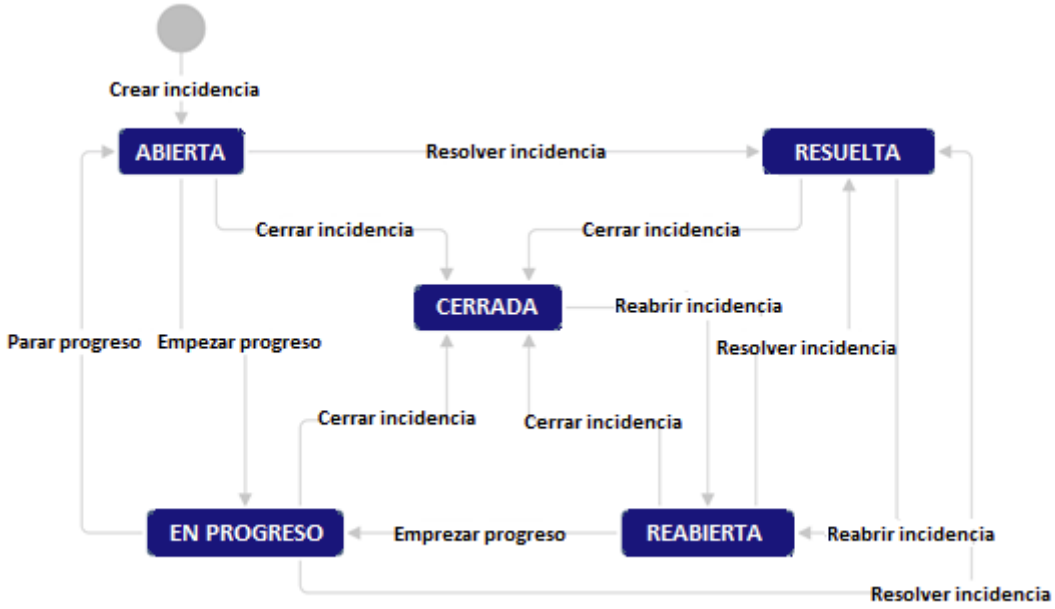


Figura 6: Flujo de trabajo de una incidencia Jira

2.4.2.3 Jira REST API

La *interfaz de programación de aplicaciones* (*API, Application Programming Interface*), es el conjunto de subrutinas, funciones y procedimientos que se ofrecen para ser utilizado por otro software como una capa de abstracción. Así, el programador puede crear aplicaciones de tal forma que no se tenga que empezar desde cero. [27]

La *transferencia de estado representacional* (*REST, Representational State Transfer*), describe una interfaz entre sistemas que utilicen HTTP para obtener o enviar datos. REST es un protocolo cliente/servidor sin estado, esto es, que cada mensaje HTTP contiene toda la información necesaria para realizar la petición. De esta forma ni el cliente ni el servidor necesitan recordar ningún estado de las comunicaciones entre mensajes. [28]

La *REST API de Jira* da acceso a los recursos de la herramienta mediante un *identificador de recursos uniforme* (*URI, Uniform Resource Identifier)*. La REST API de Jira usa el lenguaje JSON como formato de comunicación y los métodos *GET, PUT, POST* y *DELETE* del protocolo HTTP. La URI que utiliza es: *http://ip_maquina:puerto/rest/nombre_api/version_api/ nombre_recurs.* [29]

2.4.2.4 JSON

JSON (*JavaScript Object Notation*), es un lenguaje de marcado que se utiliza para intercambio de datos entre un cliente y un servidor y, debido a su amplia adopción como alternativa a XML, se considera un formato de lenguaje independiente. Una de las ventajas que se puede encontrar en

JSON sobre XML es que es mucho más sencillo escribir un analizador sintáctico de JSON. En JavaScript se puede realizar este análisis con la función *eval()*. Esta función no es del todo segura por lo que se emplea habitualmente en entornos donde la fuente de los datos es completamente fiable. [30]

En JSON se admiten los siguientes tipos de datos:

- Números.
- Cadenas de caracteres (entre comillas dobles).
- Booleanos.
- Valores nulos (null).
- Vectores, que deben de ir entre corchetes y cada elemento del vector se separa por una coma.
- Objetos, que son colecciones no ordenadas de pares de la forma "nombre:valor" separados por comas y todo ello entre llaves. El valor puede ser cualquiera de los antes descritos.

2.5 Elastic

2.5.1 Introducción

Elastic, también conocido como ELK, es una herramienta de software libre que permite indexar un gran volumen de datos y posteriormente hacer consultas sobre ellos en tiempo real. La aplicación está basada en la combinación de tres herramientas: Logstash, Elasticsearch y Kibana.

La Figura 7 ilustra el diagrama de flujo de los datos entre las tres herramientas que componen Elastic.

Figura 7: Interacción de Logstash, Elasticsearch y Kibana [31]

2.5.2 Logstash

Logstash es un motor de recopilación de datos en tiempo real. Logstash unifica dinámicamente datos desde fuentes de información que pueden ser muy diferentes entre ellas y normaliza estos datos. Logstash analiza línea a línea (evento) los logs que recibe para almacenarlos en una base de datos. El análisis de cada evento se basa en tres fases: input, filter y output.

- **Input**: complemento de Logstash usado para recibir logs mediante TCP, UDP, HTTP, Redis, archivos (logs), entre otros muchos.
- **Filter**: evento a evento realiza acciones como análisis sintáctico y semántico del evento, cambiar formato de fecha…
- **Output**: envía los datos que se han analizado en el "filter" a destinos como Redis, envío de email, Elasticsearch…

2.5.3 Elasticsearch

Elasticsearch permite almacenar, buscar y analizar grandes volúmenes de datos en tiempo (casi) real. Es la base de datos no relacional (NoSQL) que usa Logstash para almacenar todos los eventos ya procesados que llegan de los distintos logs. Una vez que los datos están en Elasticsearch, se pueden ejecutar búsquedas y agregaciones para extraer cualquier información que resulte interesante.

2.5.3.1 Conceptos

En esta sección se detallan los conceptos necesarios para entender cómo funciona la base de datos no relacional Elasticsearch

Cluster: un cluster es un conjunto de uno o más nodos (servidores) que conjuntamente almacenan los datos y proporciona herramientas de indexación y búsqueda en todos los nodos. Un cluster se identifica por un nombre único. Este nombre es importante porque un nodo sólo puede ser parte de un cluster si el nodo está configurado para unirse al cluster por su nombre.

Nodo: un nodo es un único servidor que es parte de un clúster, donde se almacenan los datos, y participa de forma activa en la indexación y búsqueda dentro del clúster. Al igual que un clúster, un nodo se identifica por un nombre. Este nombre es importante si se quiere saber qué servidores de la red corresponden al clúster de Elasticsearch.

Índice: Un índice es un conjunto de documentos (líneas de un archivo de log) que tienen características similares. En un clúster se pueden definir tantos índices como se desee.

Tipo: Dentro de un índice se pueden definir uno o más tipos. Un tipo es una categoría lógica del índice, es decir, un tipo es un conjunto de documentos que tienen un grupo de campos comunes.

Fragmentos ("shards") y réplicas: Un solo índice puede almacenar una gran cantidad de datos, por ejemplo, un billón de documentos que ocupen en disco un terabyte puede dar lugar a búsquedas excesivamente lentas si existe un sólo nodo. Para arreglar este problema, Elasticsearch divide los índices en varias piezas llamadas fragmentos. Cuando se crea un índice, se puede definir el número total de fragmentos que se deseen. Esto permite distribuir y paralelizar operaciones a través de los fragmentos (en múltiples nodos) aumentando así el rendimiento de las búsquedas. Respecto a fallos o caídas en los nodos, Elasticsearch permite realizar una o más copias de los fragmentos de los índices llamados réplicas. De esta forma existe una tolerancia a fallos y además gracias a las réplicas se podrán realizar búsquedas en paralelo sobre todas las réplicas. Por defecto, a un índice en Elasticsearch se le asignan cinco fragmentos primarios y una réplica. Por ejemplo,

si el cluster cuenta con 2 nodos, el índice tendrá cinco fragmentos primarios y otros cinco fragmentos replicados que es una réplica completa. [32]

2.5.4 Kibana

Kibana es una plataforma web de análisis y visualización diseñada para trabajar con Elasticsearch. Kibana es usada para buscar, visualizar e interactuar con los datos almacenados en Elasticsearch. La herramienta permite crear paneles gráficos y tablas variados para visualizar grandes cantidades de datos, almacenados en índices en la base de datos, de forma sencilla.

La Figura 8 muestra la interfaz de Kibana con ejemplos de los logs recogidos por logstash A la izquierda de la interfaz se ven todos los campos posibles que pueden ser consultados en el panel central. Arriba, en verde, se ve una gráfica que representa el número de eventos (líneas de log) que han sido recogidos por logstash y depositados en la base de datos.

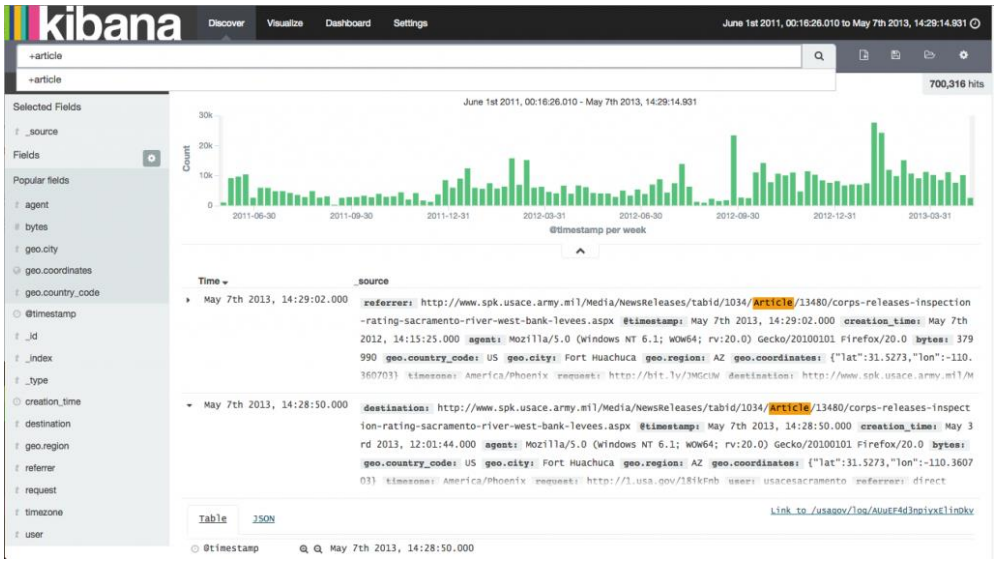


Figura 8: Interfaz web de Kibana [33]

2.6 Elastalert

2.6.1 Introducción

Elastalert es una infraestructura escrita en Python que sirve para generar alertas en base a los datos que contenga la base de datos Elasticsearch. Debido a la gran cantidad de información que generan los logs (aunque se puedan ver en Kibana) resulta muy útil recibir alertas casi en tiempo real de lo que pasa en nuestras aplicaciones.

2.6.2 Reglas y alertas

Elastalert trabaja en base a *reglas* y *alertas*. Las reglas son consultas que se realizan periódicamente contra la base de datos Elasticsearch. Cuando una regla se cumple, se activarán una o más alertas, que habrán sido previamente definidas.

Ejemplos de reglas:

- Cuando hay más de X eventos en Y tiempo. (tipo: "*frequency*")
- Cuando el ratio de eventos aumenta o disminuye. (tipo: "*spike*")
- Cuando hay menos de X eventos en Y tiempo. (tipo: "*flatline*")
- Cuando un cierto campo coincide con una lista negra o una lista blanca. (tipos: "*blacklist*" y "*whitelist*")
- Cuando un evento coincide con un filtro dado. (tipo: "*any*")
- Cuando un campo tenga dos valores diferentes en un tiempo. (tipo: "*change*")

Una alerta es un evento que se produce cuando una regla se cumple. Los tipos de alertas disponibles en la herramienta son:

- Comando
- Email
- Jira
- OpsGenie
- SNS
- HipChat
- Slack
- Telegram
- Debug

Elastalert tiene ciertas características que hacen que funcione correctamente si la base de datos Elasticsearch deja de funcionar o no está disponible. Por ejemplo, si Elasticsearch deja de estar disponible, cuando se recupere la conexión a la base de datos, Elastalert continuará realizando las consultas desde el momento en el que se perdió la conexión. [34]

2.6.3 YAML

El formato YAML (*YAML Ain't Another Markup Language*), en español, YAML no es otro lenguaje de marcado, como sus propias siglas indican, fue propuesto en el año 2001. Está inspirado en lenguajes como XML, C, Python o Perl. Fue diseñado para ser fácilmente legible gracias a la relativa sencillez en su sintaxis y fácilmente asimilable a los tipos de datos más comunes en la mayoría de lenguajes de alto nivel. [35]

Algunas características del formato YAML:

- Utiliza caracteres Unicode (UTF-8 o UTF-16)
- No se permite el uso de la tabulación para ajustar el sangrado del fichero. En su defecto, se utiliza el espacio.
- Los miembros de las listas se denotan encabezados por un guion, con un miembro por cada línea, o bien entre corchetes y separados por una coma y un espacio.
- Los arrays asociativos se representan usando los dos puntos seguidos por un espacio, en la forma "clave: valor", bien uno por línea o entre llaves y separados por una coma y un espacio.
- Los caracteres arroba y el acento grave (`) están reservados.

En este capítulo se han definido las cinco herramientas principales junto con sus características. De esta forma, las aplicaciones quedan presentadas y se procede a la definición del problema y a la solución propuesta para la solución del problema.

CAPÍTULO 3

3 DEFINICIÓN DEL PROBLEMA Y DISEÑO DE LA SOLUCIÓN PROPUESTA

3.1 Descripción del problema

El presente Trabajo de Fin de Grado ha sido desarrollado en la empresa Consulting Informático Cantabria S.L. (CIC), situada en Parque Científico y Tecnológico de Cantabria. CIC realiza trabajos como Empresa de Ingeniería y Desarrollo de Proyectos de Informática y Comunicaciones. Presta servicios de Consultoría IT, Desarrollo Software e Integración IT, Soporte Mantenimiento, ELK (Elasticsearch, Logstash y Kibana), Servicios Cloud, Formación y Externalización de servicios. Trabaja con los sectores de Utilities, Industria y Servicios, Pequeña y Mediana Empresa, Sector Público, Transporte y Logística, Telecomunicaciones y Ciudades inteligentes y cuenta con alrededor de 250 empleados.

Concretamente el TFG ha sido realizado en el departamento de Sistemas, dentro de la empresa. En este departamento son expertos en cuestiones tales como: Sistemas, Comunicaciones unificadas, Ciberseguridad, Network Monitoring y Servicios de Soporte.

El departamento de sistemas de CIC cuenta con un problema y es la necesidad de un sistema moderno de monitorización. CIC, al ser una empresa tan grande, necesita automatizar el proceso de monitorización.

El departamento trabaja con Elastic para la monitorización de logs de las diferentes aplicaciones de la empresa. Actualmente se consulta Kibana para ver el estado de estas aplicaciones en función de los logs, pero esto no resulta productivo debido a que la cantidad de logs que hay.

Asimismo, debido al amplísimo número de equipos con los que cuenta la empresa es necesario un sistema de monitorización moderno cuya función sea controlar los equipos que hay en la empresa.

Por último, el departamento necesita una CMDB que guarde múltiples datos de los activos de la empresa, para una mejor organización. De esta forma, por ejemplo, si un servidor virtualizado se rompe, con un simple vistazo a la CMDB se puede ver cuáles son las máquinas virtuales que se encontraban dentro de dicho servidor y que por lo tanto han dejado de funcionar.

El objetivo de este TFG es solucionar los problemas descritos anteriormente.

3.2 Definición concreta de los objetivos que se pretenden conseguir con el desarrollo del presente proyecto

El departamento de sistemas cuenta con un Elastic ya instalado, que recibe logs continuamente de las distintas aplicaciones corporativas. Pero el departamento quiere ir más allá, implementando un sistema de alarmas que se active cuando se cumplan ciertas reglas de alertas de negocio, que avise al personal responsable de que algo no funciona correctamente en función de los datos recogidos en los logs.

Además, el departamento utiliza la herramienta Jira como gestor de incidencias en la empresa por lo que uno de los objetivos es que el sistema de alarmas abra incidencias en Jira, y que serán escaladas al personal correspondiente.

Por otra parte, debido a la necesidad de monitorización de máquinas virtuales, es necesario contar con una herramienta moderna de monitorización de equipos, cuyos mecanismos deberán ser aprendidos y usados para adaptarlos a los requerimientos del departamento. También es objetivo de este proyecto implementar un sistema de alarmas, que al cumplirse ciertas condiciones, sea capaz de abrir incidencias en la herramienta Jira de forma automática.

Por último, el departamento tiene la necesidad de contar con una CMDB para tener un inventario de los activos que hay en la empresa. También se requiere que los datos de los equipos que vayan a estar inventariados se recojan de forma automática.

Todo esto implica un complejo sistema de herramientas que hay que integrar, desarrollando una serie de interfaces entre ellas, y programando cada una para realizar su trabajo y comunicarse con el resto. El objetivo final es conseguir un sistema completamente automatizado, que sea capaz de realizar la funcionalidad explicada anteriormente.

3.3 Diseño de la solución

Para llegar a la solución cumpliendo con los objetivos desarrollados en el apartado anterior, se ha elegido un desarrollo basado en integración de aplicaciones, frente a una posible solución de desarrollo software. Se ha escogido esta solución por dos razones.

La primera es que en el departamento ya se contaba con dos herramientas, Elastic y Jira. Elastic ya permite la monitorización una máquina a nivel de log, monitorizado estos archivos generados por las aplicaciones que se ejecuta en cada uno de los equipos monitorizados. Por otra parte, el gestor de incidencias Jira puede ser usado para la implementación del sistema de alarmas ya que se pueden diseñar interfaces entre las herramientas que generen las alertas y Jira, convirtiendo las alertas en incidencias.

La segunda razón es que la solución de integración es más rápida que el desarrollo de aplicaciones, y sobre todo el desarrollo de las aplicaciones que se usan en este proyecto, que cuentan con una alta complejidad.

En la Figura 9 se puede ver un esquema de la comunicación entre las herramientas.

Figura 9: Integración de las herramientas

En el proyecto cada herramienta cumple una función:

- **Elastic**: se encarga de recoger logs, línea a línea y almacenarlos en una base de datos no relacional, que después serán presentados a través de una interfaz web, configurada a gusto del usuario.
- **Jira**: se usa en todo el proyecto para cumplir con el objetivo de la implementación de un sistema de alarmas
- **Elastalert**: se encarga de realizar consultas a la base de datos de Elastic, y en función de los datos recibidos, se generarán alertas basadas en incidencias Jira.

Con estas tres herramientas ya se tiene una parte del proyecto, que es la monitorización de máquinas virtuales a nivel de log con sistema de alarmas.

- **Zabbix**: monitoriza diferentes parámetros de una máquina virtual como pueden ser el porcentaje de CPU utilizada, el disco, el tráfico de red… El sistema de alarmas de Zabbix también se implementa mediante incidencias Jira.
- **CMDBuild**: es la herramienta que se encarga de almacenar en una base de datos un inventario de los activos de la empresa.

Con estas dos herramientas se completa el proyecto. Monitorización de máquinas virtuales a nivel de sistema con su sistema de alarmas e inventariado de máquinas virtuales.

En la Figura 9, las flechas identificadas con los números 1-6 son las interfaces entre las herramientas, es decir, la integración de las mismas.

- **Elastic – Elastalert – Jira**: Elastalert realiza consultas (flecha número 1) a la base de datos de Elastic, y éste devuelve los datos correspondientes a Elastalert (flecha número 2). Por último, en función de los datos recibidos desde la base de datos, se abrirán incidencias en Jira (flecha número 3) en base a reglas de alertas de negocio definidas. Las consultas a la base de datos, las reglas de alertas de negocio y la apertura de incidencias en Jira con la información correspondiente se configuran en Elastalert con ficheros de

configuración en formato YAML. Todo esto lo realiza la infraestructura desarrollada en Python que ejecuta Elastalert

- **Zabbix – Jira**: la herramienta Zabbix realiza sondeos, previamente definidos, a las máquinas virtuales que estén siendo monitorizadas. En función de los datos recibidos de las máquinas, se realizará una apertura automática de incidencias Jira (flecha número 4) con la información correspondiente. Esta apertura de incidencias se realiza mediante la ejecución de un script en Linux, que hace uso de la REST API de Jira.
- **Zabbix – CMDBuild**: Zabbix permite construir un inventario de equipos ya que puede obtener parámetros de las máquinas virtuales como su nombre, el sistema operativo que se ejecuta, el número de CPUs… Con esta información en la base de datos, en CMDBuild se define una interfaz en la cual se especifican datos a recoger de la base de datos de Zabbix (flecha número 5) y que serán insertados en CMDBuild (flecha número 6).

Es muy importante, en primer lugar, realizar un buen trabajo de búsqueda de las herramientas que más se adecuen al cumplimiento de los objetivos del proyecto. Posteriormente se le debe dedicar tiempo y trabajo al estudio de la funcionalidad y de las tecnologías de las que hace uso cada herramienta, y así, por último, realizar el diseño e implementación de las interfaces utilizando las tecnologías de las que dispone cada herramienta (por ejemplo: el uso de APIs)

El siguiente capítulo describe en detalle la implementación seguida para convertir este diseño en una herramienta informática que resuelva el problema planteado.

CAPÍTULO 4

4 IMPLEMENTACIÓN DE LA SOLUCIÓN PROPUESTA

4.1 Instalación de las herramientas

En este capítulo se presenta la forma en la que se ha resuelto uno de los principales objetivos del proyecto, la realización de un sistema automático de instalación que permita desplegar toda la infraestructura de forma ágil y sencilla, en diferentes MVs. La instalación y funcionamiento de cada herramienta puede necesitar que previamente estén instaladas y configuradas otra serie de herramientas de las que se hace uso. Por este motivo el proceso es complejo y se ha realizado una infraestructura de instalación automática. El mecanismo de instalación se basa en una serie de scripts que instalen las herramientas utilizadas.

Estos scripts han sido desarrollados en el lenguaje Shell de Linux, (en concreto con la Shell bash) ya que todas las herramientas están instaladas en este sistema operativo. En cada sección del capítulo se especifican los requisitos hardware de las máquinas virtuales para realizar la instalación de las herramientas. Asimismo, se explica el contenido de los scripts, desglosando las diferentes partes de la instalación.

Todas las herramientas han sido instaladas en máquinas virtuales diferentes. La ventaja de realizar esta instalación modular es que, si se rompe una máquina virtual, no se verá afectado el sistema al completo, sino una parte de él. Además del proceso de descarga de software y su posterior instalación, se ha hecho uso del comando *"sed"* para ajustar a la instalación los archivos de configuración necesarios de cada herramienta y de esta forma poder empezar a utilizar la aplicación nada más ejecutar su respectivo script de instalación.

Los scripts de instalación de Zabbix, CMDBuild, Jira y Elastalert se encuentran en los Anexos I, II, III y IV, respectivamente. No se incluye script de instalación de Elastic ya que la empresa no lo requirió.

A continuación, se describe en detalle todos los pasos necesarios para realizar la instalación y puesta en marcha de cada una de las herramientas anteriormente citadas. Cada uno de estos pasos se corresponde con un conjunto de instrucciones del script de instalación correspondiente a cada herramienta, recogidos en los Anexos I-IV. Las líneas a las que se hace referencia en el texto corresponden con las líneas de cada uno de los scripts.

El tamaño de los discos de las máquinas virtuales que contengan las herramientas Zabbix CMDBuild y Jira son orientativos, ya que este valor dependerá del tamaño de las bases de datos asociadas a cada herramienta.

4.1.1 Zabbix

Los requisitos hardware necesarios para instalar Zabbix son los siguientes:

- Sistema operativo: CentOS 7
- Número de CPUs: 2
- Memoria RAM: 2GB
- Disco duro: 80GB

A continuación, se describen los pasos para instalar y configurar Zabbix. El Anexo I incluye el script para la instalación de esta herramienta.

4.1.1.1 Instalación y configuración de la base de datos

El primer paso es instalar MySQL en la máquina virtual. Para ello se configuran los repositorios y posteriormente se instala el servidor de bases de datos MySQL (líneas 2-3).

Posteriormente se inicia el servicio *mysqld* y se configura dicho servicio para que se inicie cuando la máquina virtual se encienda (líneas 4-5).

El siguiente paso es configurar MySQL para que aloje los datos de la herramienta, es decir, los equipos monitorizados, las acciones, las plantillas… Ello implica crear la base de datos, que se llamará "zabbix", y crear el usuario "zabbix" al que se le darán privilegios totales sobre esta base de datos y se le asignará una contraseña. Esta contraseña se le solicita al usuario por consola mediante el comando *read* y se guarda en una variable, $PASS. Para realizar esta operación se crea un fichero con lo anteriormente descrito que luego será ejecutado mediante el comando *mysql* (líneas 6-12).

4.1.1.2 Instalación y configuración del servidor Zabbix

La herramienta requiere crear un usuario y un grupo Unix, ambos llamados "zabbix" (líneas 13-14).

La instalación de Zabbix se puede realizar mediante paquetes o mediante código fuente. En este caso se ha escogido la instalación mediante código fuente. La ventaja de instalar mediante este método es que el programa se creará un ejecutable con microinstrucciones del procesador en el que se ejecuta y estará optimizado específicamente para dicho procesador.

Previamente a la ejecución del script, se deberá obtener el código fuente de la herramienta, disponible en:

https://sourceforge.net/projects/zabbix/files/ZABBIX%20Latest%20Stable/3.0.1/zabbix-3.0.1.tar.gz/download

Se descargará un archivo que deberá de ser descomprimido (línea 15).

Tras de la extracción se creará un directorio donde se encuentran archivos SQL que modelan la base de datos "zabbix" para poder empezar a usar la herramienta (líneas 16-19).

Para la instalación del propio servidor Zabbix mediante código fuente se debe instalar `gcc mysql-devel libxml2-devel net-snmp-devel curl libcurl libcurl-devel` como dependencias (línea 20).

Por último, se ejecuta el archivo *configure*. La ejecución verificará que todas las dependencias estén instaladas y generará un archivo llamado *Makefile*. Después se ejecutará el comando *make*, que compilará el código fuente e instalará la herramienta (líneas 21-22).

Ahora se edita el fichero de configuración del servidor Zabbix para establecer el nombre donde se encuentra el servidor (localhost) y la contraseña del usuario "zabbix" de MySQL (líneas 23-24).

Cada vez que se enciende la máquina virtual se debe ejecutar el comando *zabbix_server* para que la herramienta empiece a funcionar. Para ello, en el script */etc/rc.d/rc.local* se escribe este comando, ya que este fichero se ejecuta al iniciar la máquina. También hay que darle permisos de ejecución (líneas 25-26).

4.1.1.3 Servidor web y PHP

Zabbix utiliza una interfaz web para gestionar por completo la herramienta. Para ello se debe instalar un servidor web Apache y PHP. También se debe iniciar el servicio y habilitarlo para que se inicie cuando se encienda la máquina (líneas 27-29).

El siguiente paso es configurar PHP según los requisitos de la herramienta. Para ello el fichero */etc/php.ini* deberá contener unos parámetros concretos (líneas 30-38).

Para terminar de configurar PHP se deberá instalar `php-bcmath php-mysql php-mbstring php-gd php-xmlwriter php-xmlreader php-ctype php-session php-gettext` como dependencias para que la interfaz web funcione correctamente (línea 39).

Seguidamente, se deberá copiar el código fuente de la interfaz web al directorio por defecto de Apache donde están publicados los contenidos web y asignar usuario y grupo a todos los archivos (líneas 40-43).

Por último, se habilitará la traducción al castellano de la interfaz web y se ejecutará el script de traducciones (líneas 44-45).

4.1.1.4 Comando fping

El comando *fping* permite enumerar rápidamente que equipos están en la red. Zabbix utiliza este comando para el descubrimiento de red y no viene instalado por defecto. La instalación será realizada mediante código fuente. Los pasos a seguir en la instalación son muy similares a los que se efectúan en la instalación de Zabbix. Una vez instalado se configurarán los permisos del propio comando y se especificará al fichero de configuración del servidor Zabbix la ruta del comando (*/usr/local/sbin/*) (líneas 49-60).

Finalizada la instalación, para acceder a la interfaz web de la herramienta, hay que escribir en un navegador *http://ip_máquina/zabbix*. Al acceder por primera vez aparecerá un pequeño instalador gráfico. Seguir las instrucciones y aparecerá el login. El usuario por defecto para acceder a la herramienta es:

- Nombre de usuario: Admin
- Contraseña: zabbix

4.1.2 CMDBuild

Los requisitos hardware necesarios para instalar CMDBuild son los siguientes:

- Sistema operativo: CentOS 7
- Número de CPUs: 2
- Memoria RAM: 4-8GB
- Disco duro: 100GB

A continuación, se describen los pasos para instalar y configurar CMDBuild. El Anexo II incluye el script para la instalación de esta herramienta.

4.1.2.1 Java Oracle 1.8

La herramienta está desarrollada en Java por lo que es necesario instalar este entorno. Java será descargado e instalado en */opt/* bajo el directorio *jdk1.8.0_72*. Posteriormente, mediante el comando *alternatives*, se especificarán los ejecutables "Java" a usar, es decir, los que se encuentran en */opt/jdk1.8.0_72* (líneas 1-10).

4.1.2.2 Servidor web

Simplemente se instala Apache y se habilita el servicio (líneas 11-14).

4.1.2.3 Tomcat 8

Tomcat, desarrollado bajo el proyecto Jakarta en la Apache Software Fundation, es un contenedor de aplicaciones web que será utilizado para desplegar CMDBuild. Tomcat requiere crear un usuario para ejecutar el servicio. En el Anexo V está el script de inicialización del servicio Tomcat el cual se añade como servicio a la máquina. También se deberá crear un "virtualhost" Apache que se encuentra en Anexo VI (líneas 18-32).

4.1.2.4 PostgreSQL 9.3

CMDBuild utiliza PostgreSQL como base de datos. Para su instalación se configurarán los repositorios para posteriormente realizar la descarga de PostgreSQL. A continuación, se inicializa el área de datos que utilizará la base de datos. También se iniciará el servicio y se habilitará para que arranque al encenderse la máquina. Por último, se configurarán los archivos relativos a conexiones a la base de datos. Para ello, se pedirá al usuario que introduzca la dirección IP de la máquina virtual (líneas 33-45).

4.1.2.5 Despliegue de la herramienta

Una vez configurados Java, Apache, Tomcat y PostgreSQL, se puede proceder a desplegar la aplicación en el contenedor Tomcat. La herramienta CMDBuild está empaquetada en un archivo ".war" que contiene todo lo necesario (librerías Java, archivos XML, páginas web…) para hacer uso de ella una vez desplegado el archivo en Tomcat.

Previamente a la ejecución del script, se deberá obtener el código fuente de la herramienta, disponible en:

https://sourceforge.net/projects/cmdbuild/files/2.4.0/cmdbuild-2.4.0.zip/download

Se descargará un archivo que deberá de ser descomprimido (línea 46).

En primer lugar hay que cambiar el tamaño máximo de los archivos ".war" que Tomcat permite desplegar. En segundo lugar se deben copiar dos archivos para realizar conexiones con la base de

datos y el conector Java para MySQL, necesario para la integración entre Zabbix y CMDBuild. Por último se copia el archivo que contiene ".war" (líneas 46-56).

Al finalizar la instalación se recomienda reiniciar la máquina. La herramienta es accesible desde el navegador introduciendo *http://ip_máquina:8080/cmdbuild*.

4.1.3 Jira

Los requisitos hardware necesarios para instalar Jira son los siguientes:

- Sistema operativo: CentOS 7
- Número de CPUs: 2
- Memoria RAM: 4GB
- Disco duro: 50GB

A continuación, se describen los pasos para instalar y configurar Jira. El Anexo III incluye el script para la instalación de esta herramienta.

4.1.3.1 Java Oracle 1.8

La instalación es exactamente igual que la explicada en el apartado 4.1.2.1 excepto por la introducción de una variable de entorno en el fichero */etc/environment* (líneas 1-11).

4.1.3.2 Instalación y configuración de la base de datos

Este proceso es similar al del apartado 4.1.1.1. Solo se debería cambiar el nombre de la base de datos (p.e. "jira") y el nombre del usuario de la base de datos MySQL (p.e. "jira") (líneas 15-27).

4.1.3.3 Instalación de la herramienta

Primeramente, se deberá descargar la herramienta y descomprimirla en su directorio de instalación. Después se creará el directorio "home" de Jira donde se alojarán datos, logs, contenidos temporales… En tercer lugar, se deberá descargar el conector Java para MySQL para que la herramienta pueda interactuar con la base de datos. Posteriormente habrá que obtener el fichero de configuración de la base de datos del Anexo VII, el cual será adaptado a la instalación. Por último, se iniciará la herramienta y se hará que ésta se inicie siempre que se inicie la máquina (líneas 28-42).

Para acceder a la herramienta abrir el navegador y escribir *http://ip_máquina/*. Se abrirá una instalación gráfica en la web para configurar la herramienta y poder empezar a usarla.

4.1.4 Elastalert

Los requisitos hardware necesarios para instalar Elastalert son los siguientes:

- Sistema operativo: CentOS 7
- Número de CPUs: 2
- Memoria RAM: 2GB
- Disco duro: 20GB

A continuación, se describen los pasos para instalar Elastalert. El Anexo IV incluye el script para la instalación de esta herramienta.

Primeramente, se deberá instalar `git python gcc python-devel libevent-devel` como dependencias para poder descargar e instalar la herramienta. En segundo lugar, se descargará la herramienta. La descarga se realizará con el comando *git* ya que Elastalert se encuentra en GitHub (una plataforma de desarrollo colaborativo para alojar proyecto utilizando el sistema de control Git). Finalmente, se instalará Setuptools, que es una librería de desarrollo de paquetes, que servirá para realizar la instalación de Elastalert. Instalada la herramienta, se pedirán una serie de datos sobre el servidor donde se encuentre la base de datos Elasticsearch para la creación del índice donde se almacena información sobre reglas y alertas ejecutadas (líneas 1-8).

4.2 Definición de las interfaces

En este segundo apartado se presenta la implementación uno de los objetivos principales del proyecto, la integración de las cinco herramientas descritas en el capítulo 1.

4.2.1 Integración de Elasticsearch – Elastalert - Jira

El objetivo de esta integración es, a partir de los datos almacenados en la base de datos Elasticsearch, generar alertas basadas en incidencias Jira, que posteriormente serán resueltas por un usuario o grupo de usuarios definido en Jira.

La integración pretende tener un control automático de los logs almacenados en Elasticsearch. Resulta muy útil para la empresa ya que de esta forma se sustituye la monitorización manual basada en Kibana, por una monitorización automática, que indica a los responsables cuando se produce una situación anómala o un fallo en el sistema. No es necesario consultar Kibana para ver, por ejemplo, que una aplicación ha fallado observando su log correspondiente. Para conseguir esta integración se ha tenido que estudiar el funcionamiento de las reglas y alertas, basadas en ficheros de configuración YAML y cómo la herramienta se ejecuta con Python.

En este módulo se detalla la configuración general de la herramienta Elastalert, cómo se definen las reglas de alertas de negocio y cómo se abren incidencias en la herramienta Jira, en función de las reglas definidas.

La configuración de Elastalert está basada en ficheros YAML. La aplicación cuenta con un fichero de configuración global llamado *config.yaml*, que debe estar situado en el directorio de instalación de Elastalert. Este fichero contiene directivas generales que usa la aplicación en su ejecución. Algunas de las más importantes son:

- *rules_folder:* directorio que contendrá las reglas en formato YAML y que debe estar situado en el mismo directorio que el fichero de configuración general. Para que una regla sea leída y ejecutada, la extensión del archivo deberá ser "*.yaml"*. De esta forma, al ejecutar la aplicación, todas las reglas contenidas en el directorio que se indique en esta directiva, serán ejecutadas.
- *run_every:* cada cuánto tiempo se deben ejecutar las reglas, es decir, cada cuánto tiempo la herramienta consulta Elasticsearch. Este tiempo puede ir desde segundos hasta semanas.
- *es_host:* nombre del equipo o la IP donde se encuentra la base de datos Elasticseach.
- *es_port:* puerto de la base de datos Elasticsearch.
- *writeback_index:* índice de la base de datos (alojado en *es_host*) donde se guardarán datos de las reglas que se han ejecutado.

La Figura 10 es un ejemplo de cómo debería ser un fichero "*config.yaml*", con las directivas obligatorias para comenzar la ejecución. En este caso, las reglas están situadas en un directorio llamado *reglas_prod*, la herramienta consultará la base de datos Elasticsearch cada 10 minutos, la base de datos está alojada en una máquina que cuya IP es 192.168.15.221, escucha peticiones en el puerto 9200 y los datos de las reglas ejecutadas se guardarán en un índice llamado *elastalert_status*.

```
rules_folder: reglas_prod

run_every:
 minutes: 10

es_host: 192.168.15.221

es_port: 9200

writeback_index: elastalert_status
```

Figura 10: Ejemplo de fichero "config.yaml"

Además de este fichero global de configuración, las reglas también están escritas en ficheros con formato YAML y cada una de ellas en un fichero distinto. Al igual que el fichero *config.yaml*, en cada fichero de regla se pueden indicar las directivas *es_host* y *es_port*. Asimismo, se requieren otras directivas:

- *name:* nombre de la regla (único)
- *index:* índice de Elasticsearch que será consultado. Admite expresiones regulares.
- *type:* tipo de regla. Los tipos de regla que existen son "Any", "Blacklist", "Whitelist", "Change", "Frequency", "Spike", "Flatline", "New Term" y "Cardinality".
- *alert:* tipo de alerta que será usada cuando se cumpla una regla.

Cada tipo de regla tiene unas directivas requeridas para que dicha regla funcione. Por ejemplo, para la regla "Frequency" se deberán especificar las directivas *num_events* y *timeframe* en las cuales se indica cuantos eventos se deben producir en un periodo de tiempo para que se genere una alerta.

Para las alertas es igual, cada tipo de alerta tiene unas directivas asociadas. Para el envío de correos electrónicos se deberá especificar las siguientes directivas:

- *smtp_host:* nombre del equipo o la IP donde se encuentra el servidor de correo electrónico.
- *smtp_port:* si no se indica esta directiva, se usará por defecto el puerto 25. Si el servidor de correo usa otro puerto, se deberá especificar en esta directiva.
- *email:* dirección de correo electrónico a la que se enviará la alerta.

Estas tres directivas, si son comunes para todas las alertas, pueden ser incluidas en el fichero "*config.yaml*" y así no será necesario escribirlas en cada una de las reglas que se definan.

La Figura 11 muestra un ejemplo de regla que generará una alerta si en media hora se producen veinte eventos. Para concretar que estos 20 eventos sean conexiones SSH, se utiliza la directiva *filter* para seleccionar únicamente los eventos cuyo *syslog_program* (campo de la base de datos) sea *sshd*.

```
name: Conexión ssh

index: logstash-*

type: frequency

num_events: 20

timeframe:
 minutes: 30

filter:
- term:
    syslog_program: “sshd”


alert:
- “email”

smtp_host: 192.168.15.153

email:
- “elastalert@dominio.es”
```

Figura 11: Ejemplo de regla

A continuación se describe la interface entre Elastalert y Jira. Para realizar la apertura de incidencias en Jira cuando Elastalert genere una alerta, se deben especificar las siguientes directivas:

- *jira_server*: indica el nombre del equipo o la IP donde se encuentra el servidor Jira.
- *jira_proyect*: llave del proyecto en el cual se quiera abrir la incidencia.
- *alert_subject*: título de la incidencia
- *jira_description*: descripción de la incidencia.
- *jira_priority*: prioridad de la incidencia.
- *jira_issuetype*: tipo de incidencia.
- *jira_assignee:* nombre de usuario de Jira al que se le asigna la incidencia
- *jira_account_file*: la ruta al fichero que contiene las credenciales de un usuario administrador de Jira. Este fichero también debe estar en formato YAML y debe contener las directivas *user* y *password* que indicarán el nombre de usuario y la contraseña, respectivamente.

La Figura 12 es un ejemplo de apertura de una incidencia en la herramienta Jira cuando se cumplan los requisitos de la regla. En esta regla se indica que se quiere abrir una incidencia en Jira, que se encuentra en la máquina cuya IP es 192.168.15.30, cuya llave de proyecto es TFG, su título es "Alto número de conexiones ssh", cuya descripción es "Se ha producido un alto número de conexiones mediante el protocolo ssh", cuya prioridad es "High", el tipo de incidencia es "Incidencia", se le asigna al usuario "Marcos" y las credenciales de la cuenta de Jira para abrir la incidencia se encuentran en "/opt/elastalert/credenciales.yaml".

```
.
. (regla)
.
alert:
- "jira"
jira_server: http://192.168.15.30
jira_proyect: TFG
alert_subject: Alto número de conexiones ssh
jira_description: Se ha producido un alto número de
conexiones mediante el protocolo ssh.
jira_priority: High
jira_issuetype: Incidencia
jira_assignee: Marcos
jira_account_file: /opt/elastalert/credenciales.yaml
```

Figura 12: Ejemplo de apertura de una incidencia en Jira

Finalmente, para lanzar la aplicación se debe ejecutar el siguiente comando desde el directorio de instalación de Elastalert: *python -m elastalert.elastalert --start=NOW --verbose.* Este comando leerá el fichero *config.yaml* y ejecutará las reglas cada cierto tiempo (especificado en *config.yaml)*. Hay una solución alternativa a ejecutar este comando desde la consola, y es usar un proceso demonio llamado *supervisord*. De esta forma, arrancando o parando el demonio, se ejecuta o se para la aplicación, es decir, el comando.

Supervisor es un sistema cliente/servidor que permite monitorizar y controlar una serie de procesos en los sistemas operativos UNIX. Está destinado a ser utilizado para el control de los procesos e iniciarles y pararles como cualquier otro programa.

4.2.2 Integración de Zabbix – Jira

El objetivo de esta integración es la apertura de incidencias en Jira cuando se dispare un iniciador. Al ocurrir esto, se ejecutará una acción que llevará asociada una operación.

La política de la empresa obliga a abrir una incidencia en Jira cuando se disparan ciertos iniciadores de unas máquinas concretas que están siendo monitorizadas en Zabbix. La integración

permite que en el momento que se dispara un iniciador, se abra su correspondiente incidencia en Jira, que será asignada a un usuario definido. Gracias a la integración se ahorra tiempo en la apertura de la incidencia ya que de este modo se pasa directamente a su resolución. Para realizar esta integración se ha tenido que estudiar las funcionalidades que dan Jira y Zabbix. La utilidad de Jira usada para abrir las incidencias es su REST API que tuvo que ser estudiada para su posterior utilización, incluyendo el aprendizaje del lenguaje JSON, ya que la REST API utiliza este lenguaje. Por otra parte las acciones permiten ejecutar scripts en BASH, por lo que se diseñó uno concreto que creará un fichero JSON para ser enviado a la REST API de Jira.

En este módulo se detalla cómo se ejecuta una acción al dispararse un iniciador y las operaciones que se realizan al ejecutarse dicha acción. La operación principal que realizará la acción será ejecutar un script en el servidor Zabbix que abrirá una incidencia en la herramienta Jira.

La ejecución de las acciones se define en función de los iniciadores que haya en las diferentes plantillas definidas. De esta forma, si una plantilla tiene 5 iniciadores y dicha plantilla está asignada a 50 equipos, solo es necesario crear 5 acciones, una por iniciador. Una vez definida la ejecución, se pasa a concretar qué se debe ejecutar. En este caso se ejecutará un script con una serie de parámetros que finalmente abrirá una incidencia en Jira.

El script que se ejecuta para abrir la incidencia en Jira recoge seis parámetros que son:

- La clave del proyecto.
- Un resumen de la incidencia, que es el título de esta.
- La descripción de la incidencia.
- El usuario al cual se le asigna la incidencia.
- La prioridad de la incidencia.
- El tipo de incidencia.

Además, el script contiene cuatro parámetros más:

- El nombre del fichero que contiene los datos de la incidencia en formato JSON.
- La IP del servidor Jira.
- El nombre del usuario administrador.
- La contraseña del usuario administrador.

Con los seis parámetros que recoge el script y el nombre del fichero, se crea dicho fichero que debe de tener un formato específico. La Figura 13 representa un ejemplo de un fichero JSON creado a partir de los parámetros recogidos por el script que servirá para realizar la apertura de la incidencia.

```
{
      “fields”: {
            “project”: {
                  “key”: “TFG”
            },
            “summary”: “Pérdida de ping”,
            “description”: “Se ha producido una pérdida de ping en el
equipo 192.168.15.125”,
            “asignee”: {
                  “name”: “Marcos”
            },
            “priority”: {
                  “name”: “High”
            },
            “issuetype”: {
                  “name”: “Incidencia”
            }
      }
}
```

Figura 13: Fichero JSON para abrir una incidencia en Jira

Este fichero abre una incidencia en el proyecto cuya clave es “TFG”, cuyo título es “Pérdida de ping”, la descripción de la incidencia es “Se ha producido una pérdida de ping en el equipo 192.168.15.125”, se le asigna al usuario “Marcos”, la prioridad de la incidencia es “High” y el tipo de incidencia es “Incidencia”.

Creado el fichero, con el comando *curl* (comando orientado a la trasferencia de archivos desde o hacia un servidor) de Linux, se envía el fichero JSON a Jira usando su REST API. El comando utiliza los siguientes parámetros:

- -u: especifica el usuario y su contraseña para abrir la incidencia en Jira en la forma “usuario:contraseña”
- -X: especifica el método (también llamado verbo) HTTP a usar. En este caso se usa el método POST que sirve para enviar datos que serán procesados en el servidor.
- --data: especifica el fichero JSON donde se encuentran los datos de la incidencia que se va a abrir en Jira. Este es el fichero que procesará el servidor.
- -o: especifica el fichero donde se escribirá la respuesta HTTP del servidor
- -H: especifica un parámetro extra en la cabecera HTTP. En este caso es “Content-Type: application/json”. De esta forma se manda información adicional al servidor diciéndole que el formato del fichero es JSON.
- Al final del comando la URI de la REST API y el nombre del recurso, en este caso, incidencia: *http://ip_o_nombre_servidor/rest/api/2/issue*.

4.2.3 Integración de Zabbix – CMDBuild

El objetivo de esta integración es disponer de una base de datos consistente (CMDBuild) que relacione los activos de la empresa con los objetos monitorizados mediante la herramienta Zabbix.

Para realizar esta integración se ha tenido que estudiar cómo recoger los datos concretos de las máquinas monitorizadas por Zabbix para crear el inventario de equipos, esto es, encontrar los monitores adecuados. Éste se encuentra almacenado en una tabla en la base de datos. Por parte de CMDBuild se han estudiado sus características, concretamente el uso de conectores y concretamente cómo funciona el conector "básico". En el propio conector se ha investigado cómo usarlo, es decir, la integración propiamente dicha, conectar con la base de datos externa y coger los datos de esta base de datos y volcarlos en la CMDB. Previamente al uso del conector se debe definir una clase según las necesidades de la empresa con sus correspondientes atributos para que sean rellenados con los datos tomados de Zabbix.

En este módulo se detalla el concepto de inventario de equipos de Zabbix y cómo estos datos (almacenados en el inventario) son volcados a la herramienta CMDBuild.

Zabbix cuenta con monitores que devuelven valores como el nombre del equipo, el sistema operativo, la MAC, entre otros. De esta forma, Zabbix crea un inventario de equipos con todos los datos recogidos por los monitores. Un inventario es una colección de múltiples datos que pueden ser leídos de un equipo. Cada equipo monitorizado en Zabbix tiene su propio inventario y es almacenado en la base de datos del servidor Zabbix, en la tabla "*host_inventory*". Dicha tabla contiene cada uno de los campos que se encuentran en el inventario de equipos de la interfaz web. Por simplicidad, la Figura 14 muestra unos pocos.

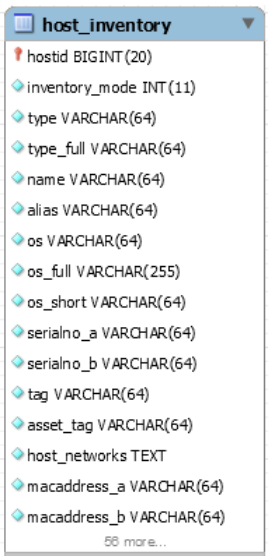


Figura 14: Tabla "host_inventory"

Para realizar la interface entre las herramientas Zabbix y CMDBuild se utiliza el conector simple. Este conector se crea y se configura desde la interfaz web, en el "módulo de administración" de CMDBuild y en concreto desde el "administrador de tareas". El proceso cuenta con los siguientes pasos:

1. En primer lugar, hacer clic en “Conector” (*Figura 15: 1*) y después en “Agrega tarea” (*Figura 15: 2*).

Figura 15: Creación del conector Zabbix – CMDBuild (Paso 1)

2. En segundo lugar, se introduce una breve descripción del conector (*Figura 16: 1*).

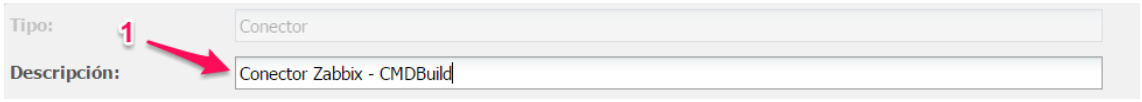


Figura 16: Creación del conector Zabbix – CMDBuild (Paso 2)

3. En tercer lugar, se especifica cada cuánto se debe ejecutar el conector, esto es, la sincronización de datos entre Zabbix y CMDBuild. Para realizar esta sincronización hay dos modos, el “básico” y el “avanzado” (*Figura 17: 1*). El “básico” permite ejecutar la sincronización cada hora, cada día, cada mes o cada año (*Figura 17: 2*). El modo “avanzado” (*Figura 17: 3*) permite introducir una expresión cron de Linux (*Figura 17: 4*).

Figura 17: Creación del conector Zabbix – CMDBuild (Paso 3)

4. A continuación, se configura la conexión con la base de datos de Zabbix. Se especifica el tipo de base de datos (*Figura 18: 1*), la dirección IP donde está alojada la base de datos (*Figura 18: 2*), el puerto (*Figura 18: 3*), el nombre de la base de datos de Zabbix (*Figura 18: 4*) y por último el nombre de usuario (*Figura 18: 5*) y la contraseña (*Figura 18: 6*) de un usuario que tenga privilegios sobre esa base de datos.

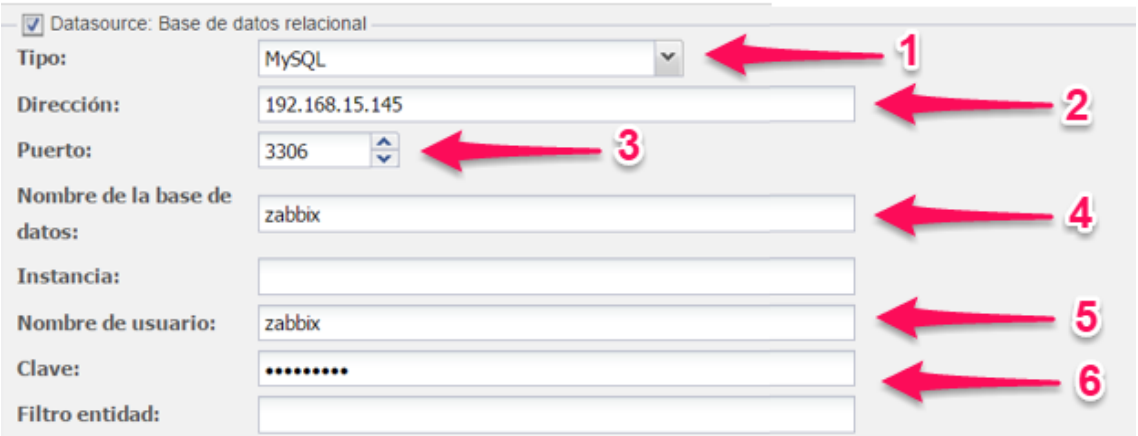


Figura 18: Creación del conector Zabbix – CMDBuild (Paso 4)

5. En quinto lugar, se establece la correspondencia entre la/s tabla/s de la base de datos de Zabbix (*Figura 19: 1*) y la/s clase/s de CMDBuild (*Figura 19: 2*). Cada correspondencia es única. Además, se especifica qué operaciones se deben realizar. Estas operaciones son crear nuevas entradas en la clase de CMDBuild si se encuentra una nueva entrada en la tabla de Zabbix (*Figura 19: 3*), actualizar datos de las diferentes entradas de la clase de CMDBuild si en la tabla de Zabbix ha habido cambios (*Figura 19: 4*) y borrar datos de la clase de CMDBuild si en la tabla de Zabbix se ha eliminado alguna entrada (*Figura 19: 5*). Estas operaciones se definen para cada una de las correspondencias

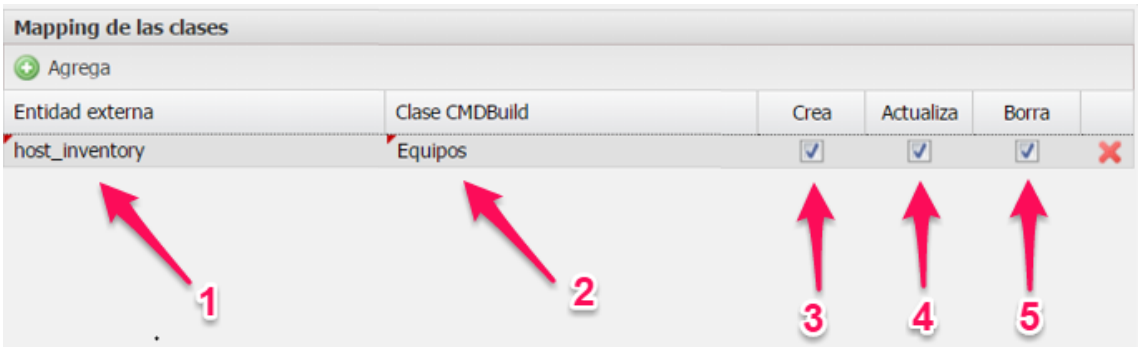


Figura 19: Creación del conector Zabbix – CMDBuild (Paso 5)

6. Finalmente, se establece más específicamente la correspondencia entre las distintas columnas de la/s tabla/s de la base de datos de Zabbix y la/s clases/s de CMDBuild. Para ello, una a una se establecen las correspondencias definiendo la tabla de la base de datos de Zabbix (*Figura 20: 1*), el nombre de la columna (*Figura 20: 2*), la clase de CMDBuild (*Figura 20: 3*) y el atributo de esta clase (*Figura 20: 4*) al cual se quiere volcar el dato. Por último, se selecciona un campo que sea único en toda la CMDB (*Figura 20:5*)

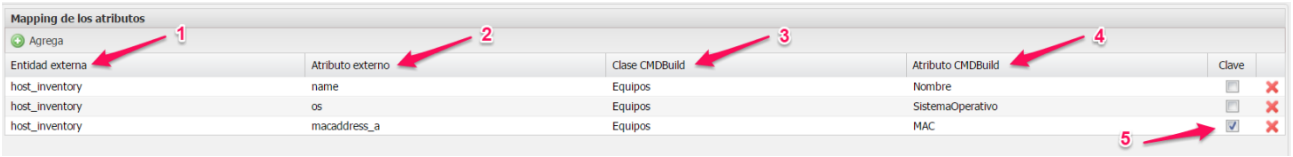


Figura 20: Creación del conector Zabbix – CMDBuild (Paso 6)

En concreto, las Figuras 15, 16, 17, 18, 19, 20 muestran un ejemplo práctico de la creación de un conector. En la Figura 15 se selecciona "Conector" y se hace clic en "Agrega tarea". En la Figura 16 simplemente se introduce la descripción del conector, "Conector Zabbix - CMDBuild". En la Figura 17 se elige el modo "Avanzado", para que la sincronización de datos se produzca cada 10 minutos. La Figura 18 muestra los diferentes parámetros de la base de datos de Zabbix, en concreto se especifica que la base de datos es MySQL, que está alojada en la máquina cuya IP es 192.168.15.145, el puerto 3306 que es el que usa MySQL por defecto, el nombre de la base de datos, "zabbix", el usuario, "zabbix" y la contraseña que se le haya dado a este usuario durante el proceso de instalación. En la Figura 19, se especifica como "Entidad externa" la tabla "*host_inventory*" y como "Clase CMDBuild" la clase "*Equipos*" que habrá tenido que ser creada previamente. Por último, en este paso se decide qué operaciones se quieren realizar y en este caso se realizarán las tres. En la Figura 20 se especifica la "Entidad externa" "*host_inventory*", y cada uno de las columnas de esta tabla de las cuales se quiera recoger sus datos. También se especifica la "Clase CMDBuild" "*Equipos*" y cada uno de los "Atributos CMDBuild" que habrán sido previamente creados dentro de la clase "*Equipos*". Para finalizar, se escoge la "Clave", que debe ser un valor único en toda la CMDB. En este caso se ha escogido la MAC de los equipos.

4.2.4 Integraciones de Zabbix

En esta sección se describen dos desarrollos personalizados que permiten a la herramienta Zabbix añadir nuevos campos a su inventario y ajustar la funcionalidad de descubrimiento de red, requeridos por el negocio para la integración con el resto de sus herramientas.

4.2.4.1 Inventario de equipos

El inventario de equipos de Zabbix contiene varios parámetros de un equipo recogidos por los monitores, pero hay un parámetro que no recoge: la dirección IP de un equipo. Por lo tanto, para poder tener la dirección IP de cada equipo en el inventario de equipos, se ha desarrollado un programa en Java.

Este programa se conecta con la base de datos "zabbix" y realiza la siguiente consulta:

```
"select a.ip, a.hostid from interface a, host_inventory b where a.hostid=b.hostid;"
```

Esta consulta devuelve la dirección IP de todos los equipos monitorizados en Zabbix que tengan el inventario de equipos activado junto con el identificador de dicho equipo.

Con estos datos se actualiza el inventario de equipos, insertando la IP de cada equipo en el campo *"host_networks"* de la tabla *"host_inventory"*:

```
"update host_inventory a set a.host_networks='"+ip+"' where a.hostid="+hostid+";"
```

La Figura 21 muestra las dos tablas implicadas en este proceso.

Figura 21: Tablas "host_inventory" e "interface"

Con el código de este programa se crea un archivo "ZabbixInventoryIPs.jar" que será ejecutado cada 10 minutos mediante la herramienta cron de Linux. El archivo deberá estar en el directorio */opt/* .Para que se realice la ejecución del programa añadir al archivo */etc/crontab* la siguiente línea:

```
*/10 * * * * root /usr/bin/java -jar /opt/ZabbixInventoryIPs.jar
```

Después reiniciar el servicio:

```
systemctl restart crond
```

4.2.4.2 Descubrimiento de red

El descubrimiento de red en Zabbix presenta varios problemas. El primero se produce cuando se reinicia un equipo. Al reiniciarse se ejecuta la acción correspondiente, como si lo hubiera descubierto por primera vez. Este problema se puede llegar a solventar añadiendo la condición "Tiempo operativo/Tiempo inoperativo" a la acción. Aquí aparece el segundo problema. Si por alguna razón un equipo con la IP 192.168.50.154 se elimina y un tiempo después (horas, días, meses…) esta IP es utilizada por un nuevo equipo, este nuevo equipo no será descubierto por la herramienta. Para solucionar estos problemas se ha combinado el uso de un script en BASH y un programa en Java.

El script se lanza con la acción que se ejecuta cuando se descubre un nuevo equipo y se le pasa como parámetro la IP del equipo descubierto. Este script ejecutará el comando *nmap* al cual se le pasara la opción *–O* (activa la detección de sistema operativo) y la IP recogida como parámetro y la salida del comando será volcada a un archivo llamado "nmap_*IP*.txt" y se enviará por e-mail con el comando *mail*. Al mismo tiempo, el script comprueba si existe el archivo y así, si existe, no ejecutará los comandos *nmap* y *mail*. De esta forma se soluciona el problema existente al reiniciarse un equipo ya que la acción sí que se ejecutará, pero no hará nada al existir el archivo. Ajustar las direcciones de correo remitente (primera dirección) y destinatario (segunda dirección) del comando *mail*.

```
IP=$1

if [ ! -f /home/zabbix/discovered_hosts/nmap_$IP.txt ]; then

        sudo /usr/bin/nmap -O $IP > /home/zabbix/discovered_hosts/nmap_$IP.txt
        mail -r zabbix@dominio.es -s "Nuevo equipo descubierto" zabbix@dominio.es <<<
`cat /home/zabbix/discovered_hosts/nmap_$IP.txt`"

fi
```

El programa en Java se encarga de solventar el segundo problema. El programa se ejecuta en un bucle infinito. Se crea la conexión con la base de datos "zabbix" y se realiza la siguiente consulta (recordar que en la tabla "*interface*" se almacenan todas las IPs de los equipos monitorizados en Zabbix):

```
"select ip from interface;"
```

Posteriormente, se guardan las IPs en una lista. A continuación, se ejecutan 10 iteraciones de un bucle que realizará lo siguiente:

- Se realiza la misma consulta y se almacenan las IPs devueltas por la consulta en otra lista
- Se comparan las dos listas y si alguna IP que estaba en la primera lista no está en la segunda se añade a una tercera lista de IPs marcadas para borrar. Esto ocurre cuando uno o varios equipos han sido borrados de Zabbix
- Se realiza un bucle que recorra la lista de IPs marcadas para borrar. Para cada IP se realizan las siguientes acciones:
    - Obtener el "*dhostid*", que es el id del equipo descubierto (no confundir con el id del equipo, "*hostid*"), de la tabla "*dservices*".
        - `"select dhostid from dservices where ip='"IP_a_borrar"';"`
    - Con el "*dhostid*" se borra la entrada correspondiente a este número en la tabla "*dhosts*". De esta forma, cuando la IP sea usada por un nuevo equipo, este será descubierto de nuevo.
        - `"delete from dhosts where dhostid="+dhostid+";"`
    - Por último, se borra el fichero "nmap_*IP*.txt" para que el script pueda volver a ejecutar el comando *nmap* con el nuevo equipo y que la salida de este comando sea enviado por e-mail. El programa Java deberá ejecutar el siguiente comando Linux para borrar
- Finalmente se espera medio segundo y se vuelve a ejecutar el bucle

Acabadas las 10 iteraciones del bucle, se vuelve a crear la primera lista de IPs y se realiza todo el proceso.

La Figura 22 ilustra las tablas implicadas en este proceso.

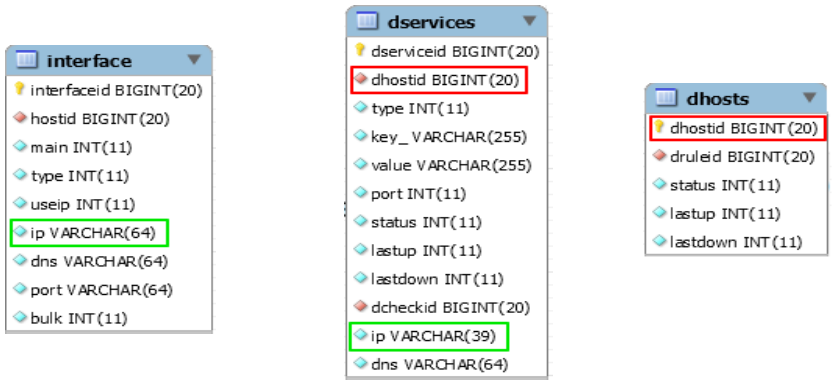


Figura 22: Tablas "interface", "dservices" y "dhosts"

Con este programa se crea un archivo "ZabbixDiscoveryAdjust.jar" y se sitúa en el directorio */opt/*. La ejecución de este programa debe ser continua por lo que se cada vez que se inicie la máquina se debe ejecutar el programa. Para ello, añadir al fichero */etc/rc.d/rc.local* la siguiente línea:

```
java -jar /opt/ZabbixDiscoveryAdjust.jar
```

CAPÍTULO 5

5 CONCLUSIONES Y TRABAJOS FUTUROS

El presente Trabajo de Fin de Grado se ha realizado en la empresa CIC, situada en el Parque Científico y Tecnológico de Cantabria (PCTCAN), y concretamente en el departamento de sistemas. Al finalizar el contrato en la empresa se realizó una reunión con el gerente del departamento de sistemas y co-director de este proyecto, Roberto Hidalgo, y otras personas del departamento para que les contara con detalle que había hecho en los seis meses de trabajo. Todos quedaron muy satisfechos con el trabajo realizado tras la reunión por lo que me ofrecieron continuar en el departamento para poder llevar el proyecto a producción total, además de realizar otros trabajos relacionados con monitorización.

El hecho de haber realizado el TFG en una empresa me ha ayudado mucho a conocer de primera mano cómo es el entorno empresarial, que obviamente dista mucho de la universidad. Así se han desarrollado competencias trasversales como el trabajo en equipo, la multidisciplinaridad, el aprendizaje autónomo de una gran cantidad de herramientas y sus tecnologías asociadas, etc.

Este TFG es un proyecto de integración, no de desarrollo, por lo que los retos y problemas que plantea su resolución son diferentes. Por un lado, nos enfrentamos a una gran complejidad a la hora de buscar las herramientas necesarias para el cumplimiento de los objetivos. Una vez encontradas se debe estudiar la manera de instalarlas, junto con todas las dependencias asociadas, algunas incluso desconocidas para mi antes de afrontar este proyecto, por lo que he necesitado estudiarlas y aprenderlas.

También cabe destacar que, debido a la existencia de dos herramientas utilizadas en el departamento antes de empezar este proyecto, se ha tenido que encontrar la solución al problema de CIC usando estas dos herramientas.

Con el paquete entero de monitorización, sistema de alarmas e inventario de equipos, quedan satisfactoriamente completados todos los objetivos que se plantearon al comienzo del proyecto, y que fueron definidos en el Capítulo 1, (sección 1.4) de esta memoria. Como consecuencia se resuelve la carencia con la que contaba el departamento de sistemas de CIC y por lo cual se propuso este TFG. En el resto de secciones de este capítulo se describe muy brevemente el trabajo realizado y cómo se ha alcanzado cada uno de estos objetivos.

Asimismo, en este último capítulo se presenta alguna idea que se puede desarrollar como trabajo futuro y de esta forma expandir este proyecto.

5.1 Preparación del paquete de instalación

EL primer paso imprescindible en todo proyecto de integración es disponer de las herramientas necesarias, y poder instalarlas de forma automática en cualquier servidor que lo necesite. Por este motivo, el primer paso del proyecto fue realizar una serie de scripts en lenguaje Shell de Linux para que la instalación de las herramientas de tal forma que las diferentes herramientas utilizadas en el proyecto fueran fácilmente y automáticamente instalables en cualquier momento y lugar. Se realizaron estos scripts para que cualquier persona con un mínimo conocimiento de Shell pueda instalar las herramientas.

Los scripts de instalación están preparados para instalar cada herramienta en una máquina virtual o servidor físico, pero nunca dos en la misma máquina virtual o servidor físico. Esto se debe a que se ideó como un sistema lo menos enlazado posible, de tal forma que, si por ejemplo Zabbix deja de funcionar, la aplicación que recoge logs y el sistema de alarmas de Elastalert seguirá funcionando, a la par que Jira y CMDBuild, con el único inconveniente de que CMDBuild no podrá actualizar sus datos, pero si seguirán siendo accesibles.

No se requirió script de instalación de Elastic ya que es una instalación compleja y su configuración depende de los archivos de logs que vaya a recoger, por lo que la instalación de esta herramienta deberá ser manual (para más información consultar https://www.elastic.co/)

5.2 Sistema de monitorización de máquinas virtuales

Para cumplir con este objetivo se seleccionó la herramienta Zabbix, la cual cumplía con creces el problema de la necesidad de monitorización de máquinas virtuales cuyos sistemas operativos podían variar. Además, esta herramienta se ve muy potente ya que, gracias a sus monitores se puede obtener multitud de datos sobre las máquinas virtuales que están siendo monitorizadas, ya sea para realizar el sistema de alarmas cuando las máquinas no funcionen correctamente como para obtener información de dichas máquinas y volcarla a la CMDB.

5.3 Sistemas de alarmas

El sistema de alarmas del paquete de monitorización tiene dos partes:

- Por un lado, gracias a Elastalert se puede implementar un sistema de alarmas en función de los datos recogidos de los logs realizado por la herramienta Elastic. Esta herramienta configurada en base a ficheros YAML que hacen consultas a la base de datos Elasticsearch, permite definir reglas de alertas de negocio en función de los datos almacenados en dicha base de datos, que permitirá abrir incidencias en la herramienta de gestión de proyectos Jira, utilizada por CIC a diario.
- Por otro lado, Zabbix cuenta con iniciadores, que son expresiones lógicas que se disparan cuando algún recurso del sistema no funciona como debiera. Esto hace que se pueda implementar un sistema de alarmas basado en incidencias Jira, al igual que con Elastalert.

Con esto descrito anteriormente se consigue implementar el sistema de alarmas, uno de los principales objetivos del proyecto.

5.4 Inventario de equipos

Es realmente importante hoy en día tener un inventario de los servidores físicos y máquinas virtuales que se tienen en la infraestructura TI. Es por esto que uno de los objetivos de este proyecto era buscar una CMDB que cumpliera con este objetivo. Gracias a CMDBuild, una CMDB altamente configurable se cumple este objetivo. Pero se va más allá y se automatiza la creación de datos dentro de la CMDB ya que Zabbix mediante sus monitores es capaz de obtener datos como el nombre del equipo, el sistema operativo que ejecuta, el número de CPUs que tiene, la RAM total, el disco duro, etc, que son los datos que se suelen almacenar en una CMDB (además de otros). Por lo que únicamente hay que definir una unión entre Zabbix y CMDBuild para que esta última coja datos de la primera, los guarde en su base de datos y los muestre a través de su interfaz web.

5.5 Trabajos futuros

Una vez cumplidos los objetivos del proyecto, existen algunos aspectos de importancia que han aparecido a raíz del trabajo desarrollado y que merecen un análisis futuro independiente. A saber, estos aspectos son:

- Tratar de cambiar la interfaz que existe entre Zabbix y CMDBuild ya que esta solo permite coger los datos de la tabla de la base de datos de Zabbix donde se almacenan los datos de interés para la CMDB y volcarlos a una misma clase en la CMDB. Se podría programar un proceso que creara entradas, actualizara y borrara estas mismas en la CMDB de forma automática (al igual que hace la interface actualmente implementada) pero de tal forma que cada grupo que existe en Zabbix se volcara a su clase homóloga en CMDBuild.
- Este producto de monitorización se ha desarrollado en CIC única y exclusivamente para la empresa. Se está pensando en vender este producto a clientes externos. Los scripts de instalación de las herramientas harán esta tarea más sencilla si finalmente se lleva a cabo.
- Cuatro de las cinco herramientas utilizadas en este proyecto son software libre. Se podría sustituir Jira por una aplicación de software libre que hiciera una función similar y por tanto cambiar las interfaces entre Elastalert y Jira y por otra parte entre Zabbix y Jira.

6 BIBLIOGRAFIA

[1] Módulo 1. Ros J., Virtualización Corporativa con VMware, 3ª edición, www.lulu.com, 2009

[2] Cap. 1. Elmasri, R., Navathe, S.B., Fundamentos de Sistemas de Bases de Datos, 5ª; edición, Pearson Education, 2008.

[3] perfil, V. (2016). *Informatica*. [En línea] ¿Qué es MySQL?. Disponible en: http://indira-informatica.blogspot.com.es/2007/09/qu-es-mysql.html [Último acceso 21 Jul. 2016].

[4] Postgresql.org.es. (2016). *Sobre PostgreSQL | www.postgresql.org.es*. [En línea] Disponible en: http://www.postgresql.org.es/sobre_postgresql [Último acceso 21 Jul. 2016].

[5] Es.wikipedia.org. (2016). *NoSQL*. [En línea] Disponible en: https://es.wikipedia.org/wiki/NoSQL#Tabla_Comparativa_de_SGBD_NoSQL.5B4.5D [Último acceso 21 Jul. 2016].

[6] Genbetadev.com. (2014). *Bases de datos NoSQL. Elige la opción que mejor se adapte a tus necesidades*. [En línea] Disponible en: http://www.genbetadev.com/bases-de-datos/bases-de-datos-nosql-elige-la-opcion-que-mejor-se-adapte-a-tus-necesidades [Último acceso 21 Jul. 2016].

[7] Zabbix.com. (2016). *2 What is Zabbix [Zabbix Documentation 3.0]*. [En línea] Disponible en: https://www.zabbix.com/documentation/3.0/manual/introduction/about [Último acceso 25 May. 2016].

[8] Zabbix.com. (2016). *2 Items [Zabbix Documentation 3.0]*. [En línea] Disponible en: https://www.zabbix.com/documentation/3.0/manual/config/items [Último acceso 25 May. 2016].

[9] Zabbix.com. (2016). *3 Triggers [Zabbix Documentation 3.0]*. [En línea] Disponible en: https://www.zabbix.com/documentation/3.0/manual/config/triggers [Último acceso 25 May. 2016].

[10] Zabbix.com. (2016). *7 Notifications upon events [Zabbix Documentation 3.0]*. [En línea] Disponible en: https://www.zabbix.com/documentation/3.0/manual/config/notifications [Último acceso 25 May. 2016].

[11] Zabbix.com. (2016). *6 Templates [Zabbix Documentation 3.0]*. [En línea] Disponible en: https://www.zabbix.com/documentation/3.0/manual/config/templates [Último acceso 26 May. 2016].

[12] Zabbix.com. (2016). *1 Network discovery [Zabbix Documentation 3.0]*. [En línea] Disponible en: https://www.zabbix.com/documentation/3.0/manual/discovery/network_discovery [Último acceso 26 May. 2016].

[13] Zabbix.com. (2016). *2 Actions [Zabbix Documentation 3.0]*. [En línea] Disponible en: https://www.zabbix.com/documentation/3.0/manual/config/notifications/action [Último acceso 25 May. 2016].

[14] Zabbix.com. (2016). 9 Users and user groups [Zabbix Documentation 3.0]. [En línea] Disponible en: https://www.zabbix.com/documentation/3.0/manual/config/users_and_usergroups [Último acceso 26 May. 2016].

[15] Zabbix.com. (2016). *2 Permissions [Zabbix Documentation 3.0]*. [En línea] Disponible en: https://www.zabbix.com/documentation/3.0/manual/config/users_and_usergroups/permissions [Último acceso 26 May. 2016].

[16] Zabbix.com. (2016). *2 Server [Zabbix Documentation 3.0]*. [En línea] Disponible en: https://www.zabbix.com/documentation/3.0/manual/concepts/server [Último acceso 26 May. 2016].

[17] Zabbix.com. (2016). *3 Agent [Zabbix Documentation 3.0]*. [En línea] Disponible en: https://www.zabbix.com/documentation/3.0/manual/concepts/agent [Último acceso 26 May. 2016].

[18] srl, T. (2016). *CMDBuild | Database modeling*. [En línea] Cmdbuild.org. Disponible en: http://www.cmdbuild.org/en/progetto/funzionalita/modellazione-database [Último acceso 30 May. 2016].

[19] srl, T. (2016). *CMDBuild | Relation Graph*. [En línea] Cmdbuild.org. Disponible en: http://www.cmdbuild.org/en/progetto/funzionalita/grafo-delle-relazioni [Último acceso 30 May. 2016].

[20] srl, T. (2016). *CMDBuild | Data Management Module*. [En línea] Cmdbuild.org. Disponible en: http://www.cmdbuild.org/en/progetto/funzionalita/modulo-di-gestione [Último acceso 30 May. 2016].

[21] srl, T. (2016). *CMDBuild | Administration Module*. [En línea] Cmdbuild.org. Disponible en: http://www.cmdbuild.org/en/progetto/funzionalita/modulo-di-amministrazione [Último acceso 30 May. 2016].

[22] Es.wikipedia.org. (2016). *Groovy (lenguaje de programación)*. [En línea] Disponible en: https://es.wikipedia.org/wiki/Groovy_(lenguaje_de_programaci%C3%B3n) [Último acceso 5 Sep. 2016].

[23] Anon, (2016). [En línea] Disponible en: https://mysticalpotato.wordpress.com/2014/01/28/groovy-un-lenguaje-de-scripting-que-se-ejecuta-en-la-java-virtual-machine/) [Último acceso 5 Sep. 2016].

[24] srl, T. (2016). *CMDBuild | Interoperability and connectors*. [En línea] Cmdbuild.org. Disponible en: http://www.cmdbuild.org/en/progetto/funzionalita/webservice-e-interoperabilita [Último acceso 30 May. 2016].

[25] srl, T. (2016). *CMDBuild | History*. [En línea] Cmdbuild.org. Disponible en: http://www.cmdbuild.org/en/progetto/funzionalita/storicizzazione [Último acceso 30 May. 2016].

[26] Confluence.atlassian.com. (2016). *What is an Issue - Atlassian Documentation*. [En línea] Disponible en: https://confluence.atlassian.com/jira064/what-is-an-issue-720416138.html [Último acceso 31 May. 2016].

[27] Es.wikipedia.org. (2016). *Interfaz de programación de aplicaciones*. [En línea] Disponible en: https://es.wikipedia.org/wiki/Interfaz_de_programaci%C3%B3n_de_aplicaciones [Último acceso 4 Jul. 2016].

[28] Es.wikipedia.org. (2016). *Representational State Transfer*. [En línea] Disponible en: https://es.wikipedia.org/wiki/Representational_State_Transfer [Último acceso 4 Jul. 2016].

[29] Docs.atlassian.com. (2016). *JIRA 1000.245.0*. [En línea] Disponible en: https://docs.atlassian.com/jira/REST/latest/ [Último acceso 4 Jul. 2016].

[30] Es.wikipedia.org. (2016). *JSON*. [En línea] Disponible en: https://es.wikipedia.org/wiki/JSON [Último acceso 4 Jul. 2016].

[31] Elmanytas.es. (2014). *Introducción a ELK (Elasticsearch, Logstash y Kibana) (parte 2) | Página web de elmanytas*. [En línea] Disponible en: http://elmanytas.es/?q=node/320 [Último acceso 20 Jul. 2016].

[32] Elastic.co. (2016). *Basic Concepts | Elasticsearch Reference [2.3] | Elastic*. [En línea] Disponible en: https://www.elastic.co/guide/en/elasticsearch/reference/current/_basic_concepts.html [Último acceso 15 Jun. 2016].

[33] Elastic Blog. (2015). *Kibana 4. Literally. | Elastic*. [En línea] Disponible en: https://www.elastic.co/blog/kibana-4-literally [Último acceso 24 Jul. 2016].

[34] Elastalert.readthedocs.io. (2016). *ElastAlert - Easy & Flexible Alerting With ElasticSearch — ElastAlert 0.0.1 documentation*. [En línea] Disponible en: http://elastalert.readthedocs.io/en/latest/elastalert.html [Último acceso 16 Jun. 2016].

[35] Es.wikipedia.org. (2016). YAML. [En línea] Disponible en: https://es.wikipedia.org/wiki/YAML [Último acceso 22 Jun. 2016].

[36] RoseHosting.com Linux VPS Hosting Blog. (2014). *How to install Tomcat 8 on a CentOS 6 VPS*. [En línea] Disponible en: https://www.rosehosting.com/blog/how-to-install-tomcat-8-on-a-centos-6-vps/ [Último acceso 1 Ago. 2016].

[37] Confluence.atlassian.com. (2016). *Connecting JIRA applications to MySQL - Atlassian Documentation*. [En línea] Disponible en: https://confluence.atlassian.com/adminjiraserver071/connecting-jira-applications-to-mysql-802592179.html [Último acceso 1 Ago. 2016].

ANEXOS

Anexo I: Script de instalación de Zabbix

```
1  #!/bin/bash

2  yum -y install http://dev.mysql.com/get/mysql-community-release-el7-
   5.noarch.rpm
3  yum -y install mysql-server
4  systemctl start mysqld
5  systemctl enable mysqld
6  echo "create database zabbix character set utf8 collate utf8_bin;" >
   /tmp/zabbix.sql
7  echo -n "Introduce el password para el usuario zabbix de mysql: "
8  read -s PASS
9  echo -e "\n"
10 echo "grant all privileges on zabbix.* to 'zabbix'@'localhost'
   identified by '$PASS';" >> /tmp/zabbix.sql
11 mysql < zabbix.sql
12 rm -f /tmp/zabbix.sql

13 groupadd zabbix
14 useradd -g zabbix zabbix
15 tar -zxvf zabbix-3.0.1.tar.gz
16 cd zabbix-3.0.1/
17 mysql zabbix < database/mysql/schema.sql
18 mysql zabbix < database/mysql/images.sql
19 mysql zabbix < database/mysql/data.sql
20 yum -y install gcc mysql-devel libxml2-devel net-snmp-devel curl
   libcurl libcurl-devel
21 ./configure --enable-server --enable-agent --with-mysql --enable-ipv6
   --with-net-snmp --with-libcurl --with-libxml2
22 make install
23 sed -i "s/# DBPassword=/DBPassword=$PASS/g"
   /usr/local/etc/zabbix_server.conf
24 sed -i "s/# DBHost=localhost/DBHost=localhost/g"
   /usr/local/etc/zabbix_server.conf
25 echo "zabbix_server" >> /etc/rc.d/rc.local
26 chmod +x /etc/rc.d/rc.local

27 yum -y install httpd php
28 systemctl start httpd
```

```
systemctl enable httpd
sed -i "s/memory_limit.*/memory_limit = 128M/g" /etc/php.ini
sed -i "s/post_max_size.*/post_max_size = 16M/g" /etc/php.ini
sed -i "s/upload_max_filesize.*/upload_max_filesize = 2M/g"
/etc/php.ini
sed -i "s/max_execution_time.*/max_execution_time = 300/g"
/etc/php.ini
sed -i "s/max_input_time.*/max_input_time = 300/g" /etc/php.ini
sed -i "s/session.auto_start.*/session.auto_start = 0/g" /etc/php.ini
sed -i "s/;date.timezone.*/date.timezone = Europe\/Madrid/g"
/etc/php.ini
sed -i "s/;mbstring.func_overload.*/mbstring.func_overload = 0/g"
/etc/php.ini
sed -i
"s/;always_populate_raw_post_data.*/always_populate_raw_post_data = -
1/g" /etc/php.ini
yum -y install php-bcmath php-mysql php-mbstring php-gd php-xml php-
xmlwriter php-xmlreader php-ctype php-session php-gettext
mkdir /var/www/html/zabbix
cd frontends/php/
cp -a . /var/www/html/zabbix/
chown -R apache:apache /var/www/html/zabbix/
sed -i "s/'es_ES'.*/'es_ES' => ['name' => _('Spanish (es_ES)'),
'display' => true],/g" /var/www/html/zabbix/include/locales.inc.php
locale/make_mo.sh

sed -i "s/SELINUX=enforcing/SELINUX=disabled/g" /etc/selinux/config
systemctl stop firewalld
systemctl disable firewalld

wget http://fping.org/dist/fping-3.10.tar.gz
tar -xvf fping-3.10.tar.gz
cd fping-3.10/
./configure --enable-ipv4 --enable-ipv6
make
make install
cd /usr/local/sbin/
chmod 710 fping
chmod ug+s fping
```

```
58 chgrp zabbix fping
59 sed -i "s/#
   FpingLocation=\/usr\/sbin\/fping/FpingLocation=\/usr\/local\/sbin\/fpi
   ng/g" /usr/local/etc/zabbix_server.conf
60 rm -rf fping-3.10/

61 yum install -y nmap
62 echo "zabbix ALL=(ALL) NOPASSWD: /usr/bin/nmap" >> /etc/sudoers
```

Anexo II: Script de instalación de CMDBuild

```
1  cd /opt/
2  wget --no-cookies --no-check-certificate --header "Cookie:
   gpw_e24=http%3A%2F%2Fwww.oracle.com%2F; oraclelicense=accept-
   securebackup-cookie" "http://download.oracle.com/otn-
   pub/java/jdk/8u72-b15/jdk-8u72-linux-x64.tar.gz"
3  tar xzfv jdk-8u72-linux-x64.tar.gz
4  alternatives --install /usr/bin/java java /opt/jdk1.8.0_72/bin/java 2
5  alternatives --config java
6  alternatives --install /usr/bin/jar jar /opt/jdk1.8.0_72/bin/jar 2
7  alternatives --install /usr/bin/javac javac /opt/jdk1.8.0_72/bin/javac
   2
8  alternatives --set jar /opt/jdk1.8.0_72/bin/jar
9  alternatives --set javac /opt/jdk1.8.0_72/bin/javac
10 rm -f jdk-8u72-linux-x64.tar.gz

11 yum -y install httpd
12 echo "ServerName localhost:80" >> /etc/httpd/conf/httpd.conf
13 systemctl start httpd
14 systemctl enable httpd

15 systemctl stop firewalld
16 systemctl disable firewalld
17 sed -i "s/SELINUX=enforcing/SELINUX=disabled/g" /etc/selinux/config

18 cd /opt/
19 useradd -r tomcat8 --shell /bin/false
20 wget http://archive.apache.org/dist/tomcat/tomcat-
   8/v8.0.33/bin/apache-tomcat-8.0.33.tar.gz
21 tar zxfv apache-tomcat-8.0.33.tar.gz
22 ln -s /opt/apache-tomcat-8.0.33 /opt/tomcat-latest
23 chown -hR tomcat8: /opt/tomcat-latest /opt/apache-tomcat-8.0.33
24 rm -f apache-tomcat-8.0.33.tar.gz
25 cp tomcat8 /etc/init.d/tomcat8
26 chmod +x /etc/init.d/tomcat8
27 chkconfig --add tomcat8
28 service tomcat8 start
29 chkconfig tomcat8 on
30 rm -f /opt/apache-tomcat-8.0.33/conf/tomcat-users.xml
```

```
31 cp tomcat-users.xml /opt/apache-tomcat-8.0.33/conf/
32 cat virtualhost >> /etc/httpd/conf/httpd.conf

33 rpm -iUvh http://yum.postgresql.org/9.3/redhat/rhel-7-x86_64/pgdg-
   centos93-9.3-1.noarch.rpm
34 yum -y install postgresql93 postgresql93-server postgresql93-contrib
   postgresql93-libs
35 /usr/pgsql-9.3/bin/postgresql93-setup initdb
36 systemctl start postgresql-9.3
37 systemctl enable postgresql-9.3
38 # Añadir línea al fichero pg_hba.conf
39 echo -n "Introduce la IP de la máquina: "
40 read IP
41 echo -e "\n"
42 echo "host all all $IP/32 trust" >>
   /var/lib/pgsql/9.3/data/pg_hba.conf
43 # Descomentar dos líneas del fichero postgresql.conf y cambiar valor
   de una de ellas
44 sed -i "s/#listen_addresses = 'localhost'/listen_addresses = '*'/g"
   /var/lib/pgsql/9.3/data/postgresql.conf
45 sed -i "s/#port = 5432/port = 5432/g"
   /var/lib/pgsql/9.3/data/postgresql.conf

46 unzip cmdbuild-2.4.0.zip
47 sed -i "s/<max-file-size>52428800<\/max-file-size>/<max-file-
   size>96944114<\/max-file-size>/g" /opt/tomcat-
   latest/webapps/manager/WEB-INF/web.xml
48 sed -i "s/<max-request-size>52428800<\/max-request-size>/<max-request-
   size>96944114<\/max-request-size>/g" /opt/tomcat-
   latest/webapps/manager/WEB-INF/web.xml
49 cp cmdbuild-2.4.0/extras/tomcat-libs/5.5/dbcp-6.0.45.jar /opt/tomcat-
   latest/lib/
50 cp cmdbuild-2.4.0/extras/tomcat-libs/5.5/postgresql-9.1-901.jdbc4.jar
   /opt/tomcat-latest/lib/
51 wget https://dev.mysql.com/get/Downloads/Connector-J/mysql-connector-
   java-5.0.8.tar.gz
52 tar -xvzf mysql-connector-java-5.0.8.tar.gz
53 cp mysql-connector-java-5.0.8/mysql-connector-java-5.0.8-bin.jar
   /opt/tomcat-latest/lib/
```

```
54 rm -rf mysql-connector-java-5.0.8*
55 chown tomcat8:tomcat8 /opt/tomcat-latest/lib/*
56 cp cmdbuild-2.4.0/cmdbuild-2.4.0.war /opt/tomcat-
   latest/webapps/cmdbuild.war
```

Anexo III: Script de instalación de Jira

```
1  cd /opt/
2  wget --no-cookies --no-check-certificate --header "Cookie:
   gpw_e24=http%3A%2F%2Fwww.oracle.com%2F; oraclelicense=accept-
   securebackup-cookie" "http://download.oracle.com/otn-
   pub/java/jdk/8u72-b15/jdk-8u72-linux-x64.tar.gz"
3  tar xzfv jdk-8u72-linux-x64.tar.gz
4  alternatives --install /usr/bin/java java /opt/jdk1.8.0_72/bin/java 2
5  alternatives --config java
6  alternatives --install /usr/bin/jar jar /opt/jdk1.8.0_72/bin/jar 2
7  alternatives --install /usr/bin/javac javac /opt/jdk1.8.0_72/bin/javac
   2
8  alternatives --set jar /opt/jdk1.8.0_72/bin/jar
9  alternatives --set javac /opt/jdk1.8.0_72/bin/javac
10 rm -f jdk-8u72-linux-x64.tar.gz
11 echo JAVA_HOME="/opt/jdk1.8.0_72" >> /etc/environment

12 systemctl stop firewalld
13 systemctl disable firewalld
14 sed -i "s/SELINUX=enforcing/SELINUX=disabled/g" /etc/selinux/config

15 yum -y install http://dev.mysql.com/get/mysql-community-release-el7-
   5.noarch.rpm
16 yum -y install mysql-server
17 systemctl start mysqld
18 systemctl enable mysqld
19 echo "CREATE DATABASE jira CHARACTER SET utf8 COLLATE utf8_bin;" >
   jira.sql
20 echo -n "Introduce el password para el usuario jira de mysql: "
21 read -s PASS
22 echo -e "\n"
23 echo "GRANT SELECT,INSERT,UPDATE,DELETE,CREATE,DROP,ALTER,INDEX on
   jirad.* TO 'jira'@'localhost' IDENTIFIED BY '$PASS';" >> jira.sql
24 echo "flush privileges;" >> jira.sql
25 echo -e "Pulsa intro si el usuario root de mysql no tiene password \n"
26 mysql < jira.sql
27 rm -f jira.sql
```

```
28 wget
   https://www.atlassian.com/software/jira/downloads/binary/atlassian-
   jira-6.4.13.tar.gz
29 tar -xvzf atlassian-jira-6.4.13.tar.gz –C /opt
30 mkdir /opt/jirahome
31 sed -i "s/jira.home.*/jira.home = \/opt\/jirahome/g" /opt/atlassian-
   jira-6.4.13-standalone/atlassian-jira/WEB-INF/classes/jira-
   application.properties
32 wget https://dev.mysql.com/get/Downloads/Connector-J/mysql-connector-
   java-5.0.8.tar.gz
33 tar -xvzf mysql-connector-java-5.0.8.tar.gz
34 cp mysql-connector-java-5.0.8/mysql-connector-java-5.0.8-bin.jar
   /opt/atlassian-jira-6.4.13-standalone/lib/
35 rm -rf mysql-connector-java-5.0.8*
36 cp dbconfig.xml /opt/jirahome/
37 sed -i
   "s/<password>password<\/password>/<password>$PASS<\/password>/g"
   /opt/jirahome/dbconfig.xml
38 sed -i "s/<Connector port=\"8080\"/<Connector port=\"80\"/g"
   /opt/atlassian-jira-6.4.13-standalone/conf/server.xml
39 cd /opt/atlassian-jira-6.4.13-standalone/
40 bin/start-jira.sh
41 echo “/opt/atlassian-jira-6.4.13-standalone/bin/start-jira.sh” >>
   /etc/rc.d/rc.local
42 chmod +x /etc/rc.d/rc.local
```

Anexo IV: Script de instalación de Elastalert

```
1  yum -y install git python gcc python-devel libevent-devel
2  cd /opt/
3  git clone https://github.com/Yelp/elastalert.git
4  cd elastalert/
5  curl https://bootstrap.pypa.io/ez_setup.py -o - | python
6  python setup.py install
7  clear
8  elastalert-create-index
```

Anexo V: Script de inicialización del servicio Tomcat 8

Nombre del fichero: tomcat8 [36]

```
#!/bin/bash

export JAVA_HOME=/opt/jdk1.8.0_72/
export JAVA_OPTS="-Dfile.encoding=UTF-8 \
  -Dnet.sf.ehcache.skipUpdateCheck=true \
  -XX:+UseConcMarkSweepGC \
  -XX:+CMSClassUnloadingEnabled \
  -XX:+UseParNewGC \
  -XX:MaxPermSize=128m \
  -Xms512m -Xmx512m"
export PATH=$JAVA_HOME/bin:$PATH
TOMCAT_HOME=/opt/tomcat-latest
SHUTDOWN_WAIT=5

tomcat_pid() {
  echo `ps aux | grep org.apache.catalina.startup.Bootstrap | grep -v grep |
awk '{ print $2 }'`
}

start() {
  pid=$(tomcat_pid)
  if [ -n "$pid" ]
  then
    echo "Tomcat is already running (pid: $pid)"
  else
    # Start tomcat
    echo "Starting tomcat"
    ulimit -n 100000
    umask 007
    /bin/su -p -s /bin/sh root $TOMCAT_HOME/bin/startup.sh
  fi


  return 0
}

stop() {
  pid=$(tomcat_pid)
  if [ -n "$pid" ]
  then
    echo "Stoping Tomcat"
    /bin/su -p -s /bin/sh root $TOMCAT_HOME/bin/shutdown.sh

    let kwait=$SHUTDOWN_WAIT
    count=0;
    until [ `ps -p $pid | grep -c $pid` = '0' ] || [ $count -gt $kwait ]
    do
      echo -n -e "\nwaiting for processes to exit";
      sleep 1
      let count=$count+1;
    done
```

```
    if [ $count -gt $kwait ]; then
      echo -n -e "\nkilling processes which didn't stop after $SHUTDOWN_WAIT
seconds"
      kill -9 $pid
    fi
  else
    echo "Tomcat is not running"
  fi

  return 0
}

case $1 in
start)
  start
;;
stop)
  stop
;;
restart)
  stop
  start
;;
status)
  pid=$(tomcat_pid)
  if [ -n "$pid" ]
  then
    echo "Tomcat is running with pid: $pid"
  else
    echo "Tomcat is not running"
  fi
;;
esac
exit 0
```

Anexo VI: Virtualhost de Apache para CMDBuild

Nombre del fichero: virtualhost

```
<VirtualHost *:80>
	ServerName localhost

	ProxyRequests Off
	ProxyPreserveHost On

	ErrorLog /var/log/httpd/tomcat.error.log
	CustomLog /var/log/httpd/tomcat.log combined

	<Proxy *>
		Order deny,allow
		Allow from all
	</Proxy>

	ProxyPass / ajp://localhost:8009/
	ProxyPassReverse / ajp://localhost:8009/
</VirtualHost>
```

Anexo VII: Fichero de conexión entre Jira y MySQL

Nombre del fichero: dbconfig.xml [37]

```
<?xml version="1.0" encoding="UTF-8"?>

<jira-database-config>
  <name>defaultDS</name>
  <delegator-name>default</delegator-name>
  <database-type>mysql</database-type>/
  <jdbc-datasource>

<url>jdbc:mysql://localhost:3306/jiradb?useUnicode=true&characterEncoding=UTF
8&sessionVariables=storage_engine=InnoDB</url>
    <driver-class>com.mysql.jdbc.Driver</driver-class>
    <username>jira</username>
    <password>password</password>
    <pool-min-size>20</pool-min-size>
    <pool-max-size>20</pool-max-size>
    <pool-max-wait>30000</pool-max-wait>
    <pool-max-idle>20</pool-max-idle>
    <pool-remove-abandoned>true</pool-remove-abandoned>

    <pool-remove-abandoned-timeout>300</pool-remove-abandoned-timeout>
    <validation-query>select 1</validation-query>
    <min-evictable-idle-time-millis>60000</min-evictable-idle-time-millis>
    <time-between-eviction-runs-millis>300000</time-between-eviction-runs-
millis>

    <pool-test-on-borrow>false</pool-test-on-borrow>
    <pool-test-while-idle>true</pool-test-while-idle>
    <validation-query-timeout>3</validation-query-timeout>
  </jdbc-datasource>
</jira-database-config>
```