UNIVERSIDAD DE VALLADOLID

E.T.S.I. TELECOMUNICACIÓN

TRABAJO FIN DE MÁSTER

MÁSTER EN INGENIERÍA DE TELECOMUNICACIÓN

Implementación de un sistema de monitorización activa de servidores

Autor:

D. Diego de las Heras Deza

Tutor:

D. Federico Simmross Wattenberg

Valladolid, 7 de Septiembre de 2015

TÍTULO:	**Implementación de un sistema de monitorización activa de servidores**
AUTOR:	**D. Diego de las Heras Deza**
TUTOR:	**D. Federico Simmross Wattenberg**
DEPARTAMENTO:	**Teoría de la Señal y Comunicaciones e Ingeniería Telemática**

TRIBUNAL

PRESIDENTE:	**D. Carlos Alberola López**
VOCAL:	**D. Manuel Rodríguez Cayetano**
SECRETARIO	**D. Miguel Bote Lorenzo**
SUPLENTE	**D. Eduardo Gómez Sánchez**
SUPLENTE	**Dña. María Jesús Verdú Pérez**
SUPLENTE	**Dña. Luisa María Regueras Santos**

FECHA:	**14 de Septiembre de 2015**
CALIFICACIÓN:	

Resumen

En este trabajo se desarrolla un sistema de monitorización activa de servidores.

Hoy en día, la mayoría de los servicios relacionados con las tecnologías de la información son proporcionados por máquinas que, en definitiva, son propensas a fallos. Se hace necesaria, por tanto, la monitorización y supervisión del correcto funcionamiento de todas las máquinas. Aunque existen multitud de soluciones de monitorización de servidores, en algunos casos es necesario el desarrollo de nuevas herramientas para satisfacer las necesidades específicas de una organización concreta.

Con este TFM se pretende desarrollar un sistema de monitorización activa de servidores adaptado a las necesidades de una filial de la empresa Hewlett-Packard. Para ello se implementará una aplicación capaz de ejecutar comandos en servidores remotos y procesar los resultados para detectar posibles anomalías en su funcionamiento; cuando se produzcan incidencias la aplicación enviará correos electrónicos al administrador. La herramienta también permite visualizar de forma directa y sencilla el estado de todas las máquinas y servicios que están siendo monitorizados para que el administrador sea capaz de detectar los problemas rápidamente y tomar las medidas necesarias para solucionarlos.

Una vez terminado el desarrollo, se ha realizado la validación de la aplicación con el tutor de este TFM en la empresa, dándole acceso a todas las herramientas y probándolas en un entorno de producción. A partir de la realimentación obtenida, y tras las correcciones necesarias, se puede asegurar que se cumplen con las expectativas de la empresa en relación a la herramienta desarrollada.

Palabras clave

Monitorización, servidores, Hewlett-Packard, Python, SQLite.

Abstract

In this work an active server monitoring system is developed.

Nowadays, most services related to technologies of information are provided by machines that, ultimately, are prone to failure. It is necessary, therefore, monitoring and supervising the proper functioning of all machines. Although there are many server monitoring solutions, in some cases is necessary the development of new tools to address the specific needs of a particular organization.

The aim of this work is to develop an active server monitoring system adapted to the needs of a subsidiary of Hewlett-Packard. For this purpose will be implemented an application that can execute commands on remote servers and process the results to detect any anomalies in its operation; when incidents occur, the application sends emails to the administrator. The tool also displays the status of all machines and services being monitored in a simple way for the administrator to be able to detect problems quickly and take necessary measures to solve them.

After completion of the development, the application validation was performed with the tutor of this work in the company, giving him access to all the tools and testing them in a production environment. Based on the feedback obtained, and after the necessary corrections, it can be ensured that the developed application fulfills the expectations of the company in relation to this project.

Keywords

Monitoring, servers, Hewlett-Packard, Python, SQLite.

Agradecimientos:

A mi familia por el apoyo recibido.

A mi tutor, Federico Simmross Wattenberg, por la ayuda otorgada.

A todos aquellos que, de una forma u otra, han contribuido a la realización de este TFM.

Gracias.

“El científico no busca un resultado inmediato.
No espera que sus ideas avanzadas sean fácilmente aceptadas.
Su deber es sentar las bases para los que vendrán, señalar el camino.”
Nikola Tesla

Dedicado a mi familia, a mis amigos y a todo aquel al que pueda resultarle de utilidad el presente trabajo.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

CAPÍTULO 1. INTRODUCCIÓN

En este capítulo se intenta dar una visión global del trabajo desarrollado, destacando las fases del desarrollo, los objetivos y los medios utilizados. Es materia de este capítulo realizar una introducción al lector sobre los sistemas de monitorización de servidores y su necesidad.

1.1 Motivación

Hoy en día las tecnologías de la información cada vez tienen una mayor importancia en la sociedad. Esto es debido en parte a que permiten a las personas obtener gran cantidad de información en cualquier momento y lugar.

La mayoría de los servicios relacionados con las tecnologías de la información son proporcionados por máquinas que, en definitiva, son propensas a fallos. Un fallo en una de estas máquinas puede suponer que el servicio se vea interrumpido y cientos o miles de personas no puedan acceder a un servicio durante un cierto periodo de tiempo. Estos fallos en el servicio se traducen en un descontento de los usuarios y eventualmente en la pérdida de usuarios y en pérdidas económicas para la empresa prestadora de los servicios.

Se hace necesaria, por tanto, la monitorización y supervisión del correcto funcionamiento de todas las máquinas por parte de personal especializado para que los fallos de funcionamiento, que se producirán antes o después, se solucionen en el menor tiempo posible. El problema se hace mayor cuando en una organización existe un número muy elevado de máquinas y los recursos para supervisarlas son limitados.

Los sistemas de monitorización de servidores han ido ganando importancia con el paso de los años [1] y actualmente se han convertido en herramientas de trabajo indispensables para cualquier administrador. Estos sistemas permiten aumentar la productividad [2] automatizando muchas tareas repetitivas y haciendo posible que un grupo reducido de personas pueda supervisar un número elevado de máquinas.

Un sistema de monitorización de servidores puede ayudar tanto en la prevención de problemas futuros como en la identificación de la causa de un problema que ya se ha producido. Dentro de este grupo de herramientas podemos encontrar *software* con funcionalidades muy distintas, desde el análisis e inspección con rutinas automáticas hasta las pruebas físicas para realizar comprobaciones de *hardware*, sin embargo, todos buscan el buen funcionamiento de los sistemas informáticos y reducir la cantidad de errores potenciales. Aunque existen muchos tipos de sistemas de monitorización, todos basan su funcionamiento en la recopilación una gran cantidad de información de las máquinas que monitorizan, como el estado de la conectividad de red, el tiempo de respuesta, la carga del sistema, la ocupación de los medios de almacenamiento, el estado de los servicios que corren en la máquina, los registros históricos, etc., para después, mostrar la información más importante de forma resumida al administrador.

Actualmente existen multitud de soluciones de monitorización de servidores, tanto de *software* libre como comerciales [3]. Si bien estas herramientas son diseñadas con una estructura flexible para que puedan adaptarse a las necesidades de cada organización, en algunos casos es necesario el desarrollo

de nuevas herramientas pasa satisfacer las necesidades específicas de una organización. La necesidad de una herramienta que se ajuste al entorno de trabajo de una filial de la empresa Hewlett-Packard es lo que da pie a la realización de este Trabajo Fin de Máster.

Con este TFM se pretende desarrollar un sistema de monitorización activa de servidores adaptado a las necesidades de una filial de la empresa Hewlett-Packard. El sistema que utilizan actualmente, basado en *scripts* en Bash, tiene importantes limitaciones que impiden la automatización de algunas tareas de monitorización. Las limitaciones más importantes son: no poder monitorizar máquinas que se encuentran dentro de redes privadas y no disponer de una herramienta para visualizar el estado de las máquinas. Esto obliga a la empresa a destinar recursos humanos en tareas que podrían ser automatizadas mediante las herramientas de *software* adecuadas.

Además de cumplir con los requisitos técnicos y funcionales indicados por la empresa, el sistema tiene que ser escalable, seguro y robusto para que pueda ser utilizado en un entorno empresarial. Una vez terminado el desarrollo se espera que el sistema permita aumentar la productividad de la empresa y automatizar ciertas tareas que hasta este momento eran realizadas de forma manual por personal de la empresa.

1.2 Objetivos

El objetivo fundamental de este TFM es **desarrollar un sistema de monitorización activa de servidores adaptado a las necesidades concretas de una filial de la empresa Hewlett-Packard**. Como se ha comentado en el apartado anterior, el sistema que utilizan actualmente tiene algunas limitaciones que impiden la automatización de algunas tareas de monitorización. Con el desarrollo de esta herramienta se pretende solventar estas limitaciones para que la empresa pueda automatizar estas tareas y de esta forma aumentar su productividad destinando los recursos humanos a otras tareas.

Para satisfacer todas las necesidades identificadas por la empresa se plantean los siguientes sub-objetivos:

- Implementación de una herramienta capaz de ejecutar comandos en servidores remotos y procesar los resultados para detectar posibles anomalías en su funcionamiento.
- Implementación de una interfaz *web* que permita visualizar de forma directa y sencilla el estado de todas las máquinas y servicios que están siendo monitorizados para que el administrador sea capaz de detectar los problemas rápidamente y tomar las medidas necesarias para solucionarlos.
- Almacenar la información sensible, como claves privadas y contraseñas, de forma cifrada de manera que esta herramienta no comprometa la seguridad de otras máquinas.
- Almacenar toda la información obtenida de forma que sea posible consultarla en el futuro para poder obtener información adicional sobre el momento en el que empezó a producirse el problema.
- Notificar al administrador los problemas detectados mediante el envío de correos electrónicos. Esto permite que el administrador sea informado de problemas importantes incluso cuando no se encuentra en el puesto de trabajo.
- Detectar máquinas fuera de servicio e identificar problemas en la red mediante la realización de pruebas de conectividad.
- Modificar la configuración de la herramienta tiene que ser sencillo y utilizando el menor número posible de herramientas externas. Para ello, toda la configuración de la herramienta estará almacenada en ficheros de texto plano.
- Contemplar todos los escenarios en los que la herramienta debe trabajar. Los servidores pueden encontrarse dentro de una red privada que no cuente con conectividad con el exterior y la herramienta tiene que ser

capaz de establecer estos túnele de forma transparente para el administrador.

- Desarrollar la herramienta utilizando las tecnologías adecuadas para que pueda ser ejecutada tanto en sistemas GNU/Linux como en entornos Windows mediante el uso de Cygwin [4]. Además, esta herramienta tienen que tener el menor número posible de dependencias externas de forma que el despliegue de la misma sea rápido y no sea necesario realizar muchas modificaciones en la configuración de la máquina anfitriona.
- Utilizar únicamente *software* con licencias compatibles con el uso empresarial.

1.3 Fases y métodos

Para alcanzar los sub-objetivos propuestos en el apartado anterior es necesario seguir una serie de pasos que se detallan a continuación:

- Toma de requisitos propuestos por una filial de la empresa Hewlett-Packard en una serie de reuniones con el tutor de este Trabajo Fin de Máster en la empresa. Estas reuniones se repetirán a lo largo de la fase de diseño y desarrollo para comprobar que se cumplen las expectativas de la empresa.
- Análisis de las distintas tecnologías y herramientas a utilizar, una vez que se ha realizado la primera toma de requisitos. Algunas de tecnologías han sido definidas por el cliente y, por lo tanto, no ha sido necesaria una documentación sobre las alternativas disponibles.
- Diseño de la herramienta encargada de ejecutar los comandos en los servidores remotos. En esta fase también se diseña la base de datos donde se almacenan los resultados de los comandos ejecutados y el formato que siguen los diferentes ficheros de configuración.

- Desarrollo de la herramienta de monitorización activa de servidores, usando las herramientas elegidas en la segunda fase y a partir del diseño realizado en la tercera.
- Diseño y desarrollo de la interfaz *web* que permite visualizar de forma directa y sencilla el estado de todas las máquinas y servicios que están siendo monitorizados.
- Fase de pruebas con el tutor de este Trabajo Fin de Máster en la filial de la empresa Hewlett-Packard, para comprobar que la plataforma se adapta a sus necesidades, cumpliendo con los requisitos facilitados, y corregir los posibles errores de funcionamiento.

1.4 Medios

Para lograr los objetivos propuestos, siguiendo las diferentes fases y métodos ya mencionados se va a hacer uso de los siguientes elementos:

Software:

- *CPython (Python)* [5]: la implementación de referencia del lenguaje de programación interpretado Python. Es un *software* libre y de código abierto que sigue un modelo de desarrollo basado en la comunidad. Python 2 es el lenguaje de programación utilizado para desarrollar el sistema de monitorización.
- *SQLite* [6]: un sistema de bases de datos relacional que está contenido en una biblioteca relativamente pequeña. Es el sistema de almacenamiento elegido para guardar los resultados de los comandos ejecutados.
- *Eclipse* [7]: un entorno de desarrollo integrado utilizado principalmente para el desarrollo de aplicaciones Java pero que puede ser utilizado para el desarrollo de aplicaciones en otros lenguajes de programación mediante el uso de plug-ins.
- *PyDev* [8]: un plug-in para el entorno de desarrollo integrado Eclipse que permite el desarrollo de aplicaciones Python y cuenta con funcionalidades

avanzadas como la refactorización de código, la depuración en una interfaz gráfica y el análisis de código.

- *DB Browser for SQLite* [9]: una herramienta para crear, diseñar, editar y visualizar bases de datos SQLite mediante una interfaz gráfica. Es un *software* libre y de código abierto que está disponible para varios sistemas operativos, incluidos GNU/Linux y Windows.
- *OpenSSH* [10]: un conjunto de herramientas basadas en el protocolo SSH que permiten asegurar las comunicaciones mediante el cifrado del tráfico de red. Es utilizado tanto para la ejecución remota de comandos como para la creación de túneles.
- *Mozilla Firefox* [11]: un navegador *web* libre y de código abierto disponible para varios sistemas operativos, incluidos GNU/Linux y Windows. Utilizado para el desarrollo y las pruebas de la interfaz *web*.
- *Cygwin* [4]: un entorno UNIX-like y una interfaz de línea de comandos para Microsoft Windows. Es utilizado para realizar las pruebas de la herramienta.
- *Ubuntu* [12]: una distribución Linux de propósito general que proporciona un sistema operativo actualizado. Es el sistema utilizado para el desarrollo y las pruebas de la aplicación.
- *Raspbian* [13]: una distribución Linux optimizada para el *hardware* de Raspberry Pi [14]. Es utilizado en una placa Raspberry Pi para realizar algunas pruebas.
- *Microsoft Windows 7* [15]: un sistema operativo desarrollado por Microsoft. Es utilizado para realizar pruebas de la herramienta desarrollada en el entorno Cygwin.

Hardware:

- *Ordenador de sobremesa*:
    - Intel Core2Duo E8400 @ 3.2 GHz
    - 4 GB RAM
    - 512 GB SSD
    - AMD Radeon HD 6450 1GB GDDR3
- *Raspberry Pi model B+* [14]:
    - Broadcom BCM2835 @ 700 MHz
    - 512 MB RAM
    - 16 GB SSD

1.5 Estructura de la memoria

A continuación se define la distribución de capítulos de este documento, especificando la información contenida en cada uno, de forma que sea fácil e intuitivo para el lector localizar la información.

En este primer capítulo se intenta dar una visión global del trabajo desarrollado, destacando las fases del desarrollo, los objetivos y los medios utilizados. Es materia de este capítulo realizar una introducción al lector sobre los sistemas de monitorización de servidores y su necesidad.

En el capítulo segundo se describen con detalle las tecnologías utilizadas en la fase de desarrollo. Desde un punto de vista técnico y científico se explican las características de diferentes lenguajes de programación y lenguajes *web* haciendo énfasis en las funcionalidades que son especialmente útiles para este trabajo. En una segunda parte se realiza un estudio de los distintos tipos de bases de datos y su conveniencia. Además, se justifica por qué se han elegido estas tecnologías para el desarrollo del TFM.

A continuación, en el capítulo tercero, se muestran los requisitos, tanto funcionales como no funcionales, de la aplicación a desarrollar así como el análisis de cada componente de la aplicación. En este capítulo se amplía la información sobre la fase de desarrollo ofrecida en el primer capítulo de esta memoria incluyendo las tareas desarrolladas para la consecución de los objetivos del TFM.

En el capítulo cuarto se detalla el diseño de cada componente de la aplicación. Se profundiza en los pasos que han llevado al diseño final y se describen las partes que forman cada componente de la aplicación.

El siguiente capítulo, capítulo quinto, se incluyen los manuales de instalación y de administrador necesarios para la instalación, configuración y utilización de la aplicación.

En el sexto capítulo se explican las pruebas realizadas con la aplicación tanto en el entorno GNU/Linux como en Windows (Cygwin), así como los resultados obtenidos en cada prueba. La finalidad de este capítulo es mostrar los errores que fueron encontrados en las pruebas y los pasos que fueron llevados a cabo para su resolución.

En el séptimo y último capítulo se exponen las conclusiones extraídas durante la realización del TFM y las posibles líneas de desarrollo futuro, proponiendo nuevas ideas que amplíen las funcionalidades de la aplicación desarrollada.

CAPÍTULO 2. TECNOLOGÍAS BASE

En el capítulo segundo se describen con detalle las tecnologías utilizadas en la fase de desarrollo. Desde un punto de vista técnico y científico se explican las características de diferentes lenguajes de programación y lenguajes *web* haciendo énfasis en las funcionalidades que son especialmente útiles para este trabajo. En una segunda parte se realiza un estudio de los distintos tipos de bases de datos y su conveniencia. Además, se justifica por qué se han elegido estas tecnologías para el desarrollo del TFM.

2.1 Lenguajes de programación

Los lenguajes de programación son una de las principales herramientas que de las que se dispone para describir de forma precisa los datos y las operaciones que debe realizar una computadora para proporcionar unos resultados o servicios.

Hoy en día existe un amplio abanico de posibilidades. Cada lenguaje de programación tiene sus propias características, ventajas y desventajas. Es muy importante encontrar el lenguaje más apropiado para cada tarea, porque éste condicionará la manera de atacar el problema.

La manipulación de texto es uno de los requisitos más remarcables en este proyecto. Por este motivo, es de suma importancia utilizar un lenguaje de programación que disponga de las herramientas necesarias para la simplificación de este tipo de operaciones.

En este apartado se explican los lenguajes de programación que han sido considerados durante el desarrollo del trabajo, siendo Python el lenguaje elegido para la realización del proyecto.

2.1.1 Python

Python [5] es un lenguaje de programación interpretado creado por Guido van Rossum en el año 1991 [16]. Comparado habitualmente con Tcl, Perl, Scheme, Java y Ruby, en la actualidad se desarrolla como un proyecto de código abierto, administrado por la Python Software Foundation [17].

Python permite dividir el programa en módulos reutilizables desde otros programas Python. Viene con una gran colección de módulos estándar que se pueden utilizar como base de los programas (o como ejemplos para empezar a aprender Python). También hay módulos incluidos que proporcionan E/S de ficheros, llamadas al sistema, *sockets* y hasta interfaces a *GUI* (interfaz gráfica de usuario) como Tk, GTK, Qt entre otros.

Python se utiliza como lenguaje de programación interpretado, lo que ahorra un tiempo considerable en el desarrollo del programa, pues no es necesario compilar ni enlazar [16]. El intérprete se puede utilizar de modo interactivo, lo que facilita experimentar con características del lenguaje, escribir programas desechables o probar funciones durante el desarrollo del programa. El principal objetivo que persigue este lenguaje es la facilidad, tanto de lectura, como de diseño.

2.1.1.1 Características y paradigmas

Python es un lenguaje de programación multiparadigma. Esto significa que más que forzar a los programadores a adoptar un estilo particular de

programación, permite varios estilos: programación orientada a objetos, programación estructurada y programación funcional [16]. Otros paradigmas están soportados mediante el uso de extensiones. Python usa tipos de datos dinámicos y "*reference counting*" para el manejo de memoria. Una característica importante de Python es la resolución dinámica de nombres, lo que enlaza un método y un nombre de variable durante la ejecución del programa (también llamado ligadura dinámica de métodos).

Otro objetivo del diseño del lenguaje era la facilidad de extensión. Se pueden escribir fácilmente nuevos módulos en C o C++. Python puede utilizarse como un lenguaje de extensión para módulos y aplicaciones que necesitan de una interfaz programable.

A continuación se listan las características más importantes del lenguaje:

Lenguaje interpretado:

Un lenguaje interpretado es aquel que se ejecuta utilizando un programa intermedio llamado intérprete, en lugar de compilar el código a lenguaje máquina que pueda comprender y ejecutar directamente una computadora. La ventaja de los lenguajes compilados es que la ejecución del código es más rápida. Sin embargo, los lenguajes interpretados son más flexibles y más portables. Python tiene, no obstante, muchas de las características de los lenguajes compilados, por lo que se podría decir que es semi-interpretado. En Python, como en Java y muchos otros lenguajes, el código fuente se traduce a un pseudo-código máquina intermedio llamado *bytecode* la primera vez que se ejecuta, generando archivos *.pyc* o *.pyo* (*bytecode* optimizado), que son los que se ejecutarán en sucesivas ocasiones.

Tipado dinámico:

Se refiere a que no es necesario declarar el tipo de dato que va a contener una determinada variable, sino que su tipo se determinará en tiempo de ejecución

según el tipo del valor al que se asigne, y el tipo de esta variable puede cambiar si se le asigna un valor de otro tipo.

Fuertemente tipado:

No se permite tratar a una variable como si fuera de un tipo distinto al que tiene; es necesario convertir de forma explícita dicha variable al nuevo tipo previamente. Por ejemplo, si tenemos una variable que contiene un texto (variable de tipo cadena o *string*) no podremos tratarla como un número (sumar la cadena "9" y el número 8). En otros lenguajes el tipo de la variable cambiaría para adaptarse al comportamiento esperado, aunque esto es más propenso a errores.

Multiplataforma:

El intérprete de Python está disponible en multitud de plataformas (UNIX, Solaris, Linux, DOS, Windows, OS/2, Mac OS, etc.) por lo que, si no utilizamos librerías específicas de cada plataforma, nuestro programa podrá correr en todos estos sistemas sin grandes cambios.

Orientado a objetos:

La orientación a objetos es un paradigma de programación en el que los conceptos del mundo real relevantes para nuestro problema se trasladan a clases y objetos en nuestro programa. La ejecución del programa consiste en una serie de interacciones entre los objetos. Python también permite la programación imperativa, programación funcional y programación orientada a aspectos.

Modular:

Python permite dividir el programa en módulos reutilizables desde otros programas Python. Dispone de una gran biblioteca estándar, con módulos incluidos que proporcionan E/S de ficheros, llamadas al sistema, sockets y hasta interfaces a *GUI* "*Graphical User Interface*" como Tk, GTK y Qt entre otras. Esto viene de la filosofía "baterías incluidas" ("*batteries included*"). Si hay áreas que son lentas se pueden reemplazar por *plugins* en C o C++, siguiendo la *API* para

extender o empotrar Python en una aplicación. Debido a la gran variedad de herramientas incluidas en la biblioteca estándar combinada con la habilidad de usar lenguajes de bajo nivel como C y C++, los cuales son capaces de interactuar con otras bibliotecas, Python es un lenguaje que combina su clara sintaxis con el poder de otros lenguajes.

Documentado:

Python cuenta con una entusiasta comunidad de desarrolladores y usuarios que mantienen un *wiki*, acogen conferencias internacionales y locales, contribuye a los repositorios de código en línea, etc. Cuenta además con una documentación completa, integrada tanto en el lenguaje como en páginas *web* aparte, y con tutoriales en línea para familiarizarse rápidamente con el lenguaje.

Código abierto:

Python se desarrolla como un proyecto con licencia de código abierto que hace que sea de libre uso y distribución, incluso para su uso comercial. La licencia de Python está administrada por la Python Software Foundation [17], ésta es compatible con la licencia GPL a partir de la versión 2.1.1, e incompatible en ciertas versiones anteriores.

Modo interactivo:

El intérprete de Python estándar incluye un modo interactivo, en el cual se escriben las instrucciones en una especie de *shell*; las expresiones pueden ser introducidas una a una, pudiendo verse el resultado de su evaluación inmediatamente. Esto resulta útil tanto para las personas que se están familiarizando con el lenguaje como también para los programadores más avanzados: se pueden probar porciones de código en el modo interactivo antes de integrarlo como parte de un programa.

Biblioteca estándar:

Python tiene una gran biblioteca estándar, usada para una diversidad de tareas [18]. Esto viene de la filosofía "pilas incluidas" ("*batteries included*") en referencia a los módulos de Python. Los módulos de la biblioteca estándar pueden mejorarse por módulos personalizados escritos tanto en C como en Python. Debido a la gran variedad de herramientas incluidas en la biblioteca estándar, combinada con la habilidad de usar lenguajes de bajo nivel como C y C++, los cuales son capaces de interactuar con otras bibliotecas, Python es un lenguaje que permite realizar todo tipo de tareas.

2.1.2 Perl

Perl [19] (*Practical Extracting and Reporting Language*) es un lenguaje de programación creado por Larry Wall a mediados del año 1987. Se caracteriza por su simplicidad y está compuesto por algunas de las características de otros lenguajes de programación como C, awk, sed, sh o BASIC.

Perl está especialmente pensado para trabajar en la extracción de información de archivos de texto y generar informes a partir del contenido de los ficheros. Además, resulta muy eficiente en la manipulación de números, texto, ficheros, directorios, bases de datos, redes u otros programas. Este gran abanico de posibilidades ha calado en comunidad de programadores y le ha permitido convertirse en uno de los lenguajes de programación más utilizados hoy en día.

Perl es un lenguaje de programación cuya distribución es libre y gratuita, por lo que cualquiera puede utilizarlo y generar programas con él. Perl, también es multiplataforma, es decir, los principales sistemas operativos lo soportan (UNIX, Macintosh y Windows).

Otra característica a destacar es que Perl es un lenguaje de programación interpretado, no se compila antes de utilizarlo, sino que se ejecuta a medida que el código escrito es leído. Para que poder trabajar con Perls solo se es necesario disponer del intérprete de lenguaje y de memoria suficiente, ya que Perl no limita

el tamaño de los datos como en otros lenguajes, sino que utiliza todos los recursos de los que disponga la máquina.

Para finalizar, CPAN [20] (*Comprehensive Perl Archive Network*), el principal sitio online de distribución de material relativo a Perl. En CPAN se pueden encontrar módulos, documentación, ejemplos y en general todo tipo de material relacionado con Perl. CPAN es el lugar de referencia de este lenguaje de programación y constituye una notoria ayuda a la comunidad de programadores.

2.1.3 Discusión sobre los lenguajes de programación

Existen varios motivos que sustentan el porqué del uso de Python como el lenguaje de programación escogido para el desarrollo de este trabajo.

Una de las razones principales razones para elegir Python es el hecho de que el cliente utiliza habitualmente otras herramientas desarrolladas en este lenguaje. Al disponer de personal familiarizado con este lenguaje y ya tener instalado el intérprete de Python en gran parte de su equipo, esto facilita mucho las tareas de instalación y de mantenimiento de la herramienta por parte de la empresa.

En la actualidad es utilizado por una amplia comunidad de personas, hecho que permite encontrar en la red de Internet abundante documentación, tutoriales y ejemplos de ayuda.

Su distribución es libre y por tanto es sencillo conseguir todas las herramientas necesarias para empezar a trabajar con este lenguaje.

Otra razón que ha llevado a la elección de este lenguaje es la portabilidad del lenguaje antes mencionada, ya que los *script* escritos en este lenguaje pueden ejecutarse en varias plataformas.

Entrando en consideraciones más concretas, la utilización de Python se justifica en gran medida en el potencial que tiene para trabajar con texto. Python suministra un buen conjunto de operaciones básicas y módulos que permiten trabajar con mayor comodidad con cualquier tipo de texto. Este hecho es

fundamental para este proyecto, puesto que es necesario procesar la salida de los comandos ejecutados en busca de cadenas de texto, expresiones regulares, etc...

Por último, destacar que Python incorpora algunas bibliotecas que han sido de vital importancia para la realización del proyecto. Cabe destacar la biblioteca *sqlite3* [21] que permite realizar operaciones en la base de datos SQLite utilizada en el desarrollo de este trabajo y la biblioteca *SimpleHTTPServer* [22] que permite crear un servidor *web* con las funciones necesarias para la realización de este proyecto.

2.2 Lenguajes *web*

Actualmente existen diferentes lenguajes de programación para desarrollar en la *web*, estos han ido surgiendo debido a las tendencias y necesidades de los creadores de contenido.

Uno de los requisitos que se han marcado para este proyecto ha sido el desarrollo de una interfaz *web* que permita visualizar la información obtenida por la herramienta de monitorización.

En este apartado se explican los lenguajes *web* que han sido utilizados durante el desarrollo del trabajo y los motivos que han llevado a su elección.

2.2.1.1 HTML

HTML (*HyperText Markup Language*, lenguaje de marcas de hipertexto), es un lenguaje de descripción de hipertexto compuesto por un conjunto de comandos, marcas o etiquetas, también llamadas *tags* que permiten definir la estructura lógica de un documento *web* y establecer los atributos del mismo como el color del texto, los contenidos multimedia, hipervínculos, etc. [23]

Se puede resumir en que HTML es un lenguaje que permite crear páginas *web* y para ello hace uso de una serie de comandos o etiquetas que indican o marcan qué contenido se debe mostrar y de qué forma.

Estos comandos van siempre incluidos entre los signos < y > y se insertan en el propio texto que forma el contenido de la página. Especifican las distintas partes de

la página, esto es, su estructura, y su formato. Adicionalmente, dan la posibilidad de insertar contenidos especiales como pueden ser imágenes, vídeos o sonidos, entre otros [23].

2.2.1.2 JavaScript

JavaScript es un lenguaje interpretado basado en *scripts* que se introducen directamente en el código HTML [23]. El código se envía desde el servidor al cliente y es este el encargado de interpretarlo. Debido a que necesita utilizar HTML a través de distintos manejadores de eventos para realizar cualquier acción, JavaScript no puede utilizarse para crear aplicaciones independientes, no entendidas en un contexto *web* o que requieran un navegador.

Las características a destacar, a modo de resumen, de este lenguaje de programación son las siguientes [23]:

- Se trata de un lenguaje interpretado, no necesita compilación y es el propio navegador del cliente el que se encarga de traducir el código.
- Es un lenguaje multiplataforma.
- Es un lenguaje de alto nivel.
- Admite programación estructurada y basada en objetos.
- Puede manejar la mayoría de eventos que pueden tener lugar en una página *web*.
- No necesita un entorno de desarrollo.

Así, según lo visto, JavaScript permite desarrollar aplicaciones que se ejecutan directamente en el cliente (en un navegador) de forma que se pueden realizar ciertas operaciones o decisiones sin necesidad de realizar una conexión con el servidor.

2.2.1.3 CSS

CSS (*Cascading Style Sheets*, hojas de estilo en cascada) es un lenguaje introducido a finales de la década de los noventa que describe cómo los documentos *web* deben ser representados gráficamente [24]. Es muy potente y flexible y está directamente conectado con HTML.

CSS permite generar un patrón de estilo que es aplicable a todos los documentos de una *web*, lo que supone un claro ahorro en el tiempo de desarrollo y en el de mantenimiento de una página. Además, esto permite organizar mejor el contenido al tener separada la estructura de una página de su diseño.

La principal desventaja de las hojas de estilo en cascada es la incompatibilidad que puede haber entre distintos navegadores. Una posible solución, desde el punto de vista del desarrollador de la página, es la de conocer las propiedades que implementa cada versión de navegador, probar diferentes estilos en varios navegadores y ser conscientes de que siempre existirá algún navegador, desde el que se visualice la página desarrollada, que no disponga del soporte adecuado para el estilo implementado en el CSS de la página, siendo el código del HTML el único que podrá interpretar correctamente [24].

2.2.2 Discusión sobre los lenguajes *web*

En cuanto a los lenguajes *web* utilizados para el desarrollo de la interfaz *web* en la que se visualiza el estado de las máquinas existía total libertad de elección. A continuación se exponen algunos de los motivos que han llevado a la elección de estos lenguajes.

Debido a que uno de los requisitos es que la interfaz *web* sea compatible con el mayor número posible de navegadores *web*, se ha optado por utilizar tecnologías maduras como HTML, JavaScript y CSS. Estas tecnologías están soportadas por los navegadores *web* más utilizados y su uso asegura una visualización correcta del contenido.

Otro punto a tener en cuenta es la simplicidad de la interfaz *web* que requiere la aplicación desarrollada. Existen multitud de *frameworks* y bibliotecas que permiten la creación de páginas con diseños vistosos y gran cantidad de contenido, pero su uso se hace innecesario incluso contraproducente en un desarrollo de este tamaño.

Durante el estudio de las diferentes tecnologías se buscó documentación sobre la biblioteca jQuery [25][26] y sobre AJAX [27] pero finalmente no ha sido necesario su uso en el desarrollo de este TFM. La utilización de jQuery o de otras

bibliotecas hubiera añadido nuevas dependencias externas a la aplicación que dificultarían su instalación y portabilidad.

2.3 Bases de datos

Un SGBD *(Sistema Gestor de Bases de Datos)* es un sistema computacional que facilita la gestión de las bases de datos [23].

Una base de datos podría definirse como una colección de datos interrelacionados que son almacenados en un soporte informático. Algunas razones que justifican su uso son su capacidad para almacenar grandes volúmenes de información, la optimización de su gestión, la facilidad para realizar consultas y la exactitud, rapidez y fiabilidad en su administración [23].

Existen diferentes tipos de bases de datos, entre las que se encuentran las de XML, las orientadas a objetos y las relacionales, que serán descritas a continuación.

2.3.1 Bases de datos XML

XML *(eXtensible Markup Language,* lenguaje de marcas extensible*)* es un metalenguaje, es decir, un lenguaje empleado para describir otros lenguajes, que permite a los desarrolladores crear sus propios *tags* o etiquetas personalizadas aportando así funcionalidades que no estaban disponibles en HTML.

Es conocido que una de las grandes características de HTML es su simplicidad, lo que le permite ser usado por un gran número de usuarios. Sin embargo, debido a las necesidades crecientes de los usuarios, surgió la necesidad de crear nuevas etiquetas para hacer que los documentos HTML fueran más atractivos. El W3C [28] *(World Wide Web Consortium)* creó el nuevo estándar XML, cuya primera versión se convirtió en recomendación del W3C en octubre del año 2000.

XML es una versión restringida de SGML *(Standard Generalized Markup Language)* diseñada especialmente para documentos *web*. Por ejemplo, XML

soporta enlaces que apuntan a múltiples documentos, por oposición a los vínculos HTML, que sólo pueden hacer referencia a un único documento de destino.

SGML es un sistema para definir tipos de documentos estructurados y lenguajes de composición con los que representar instancias de dichos tipos de documentos. SGML ha sido la forma estándar, e independiente de los fabricantes, de mantener repositorios de documentación estructurada durante más de una década. SGML permite separar lógicamente en dos partes un documento. Una parte que define la estructura del documento y la otra que contiene el propio texto. SGML es un sistema de gestión documental extremadamente potente, sin embargo no ha sido ampliamente adoptado debido a su inherente complejidad.

XML trata de proporcionar una función similar a la de SGML, pero es menos complejo y, al mismo tiempo, está preparado para funcionar en entornos de red. Significativamente, XML retiene las ventajas principales de SGML: extensibilidad, estructura y validación [29].

Existen dos tipos de bases de datos en cuanto a la forma de almacenar datos XML en ellas:

- *Base de datos XML-enabled*: si el XML no se almacena internamente como XML.
- *Base de datos XML nativa*: si el XML se almacena internamente como XML.

2.3.1.1 Bases de datos XML-enabled

Una base de datos de este tipo emplea un modelo relacional para el centro de datos del modelo de almacenamiento. Esto implica un mapeo entre el modelo de datos XML (jerárquico) y el modelo de datos relacional, o bien almacenar datos XML como un objeto carácter largo.

A su vez, dentro de una base de datos *XML-enabled*, existen dos formas de almacenar los datos XML. La primera opción consiste en almacenar un documento XML como una cadena no *parseada*. El inconveniente de este método,

es que cuando se quiere recuperar parte del documento XML, el programa desarrollado debe recuperar la cadena y efectuar el *parseo* para encontrar la información buscada, por lo que resulta evidente la poca flexibilidad de este método.

La otra opción en bases de datos *XML-enabled*, llamada trituración o descomposición, consiste en desmenuzar un documento XML en partes más pequeñas que serán almacenadas en tablas. Al usar este método, el modelo jerárquico del documento XML es forzado dentro del modelo relacional. Este método tampoco es bueno para la flexibilidad, ya que un cambio en el documento XML no se propaga fácilmente a las tablas correspondientes y además se pueden necesitar crear muchas tablas, ni para el rendimiento, ya que si se necesita obtener el documento XML original de vuelta se tiene que realizar una consulta SQL demasiado costosa, lo cual empeora aún más cuando hay más tablas unidas [30].

2.3.1.2 Bases de datos nativas XML

El segundo tipo de bases de datos analizado usa el modelo de datos XML jerárquico para almacenar y procesar XML internamente. Se emplea el mismo formato para almacenar y para procesar, es decir, no existe mapeo para el modelo relacional y los documentos XML no son almacenados como imágenes. Así, cuando se usan declaraciones *XPath* o *XQuery*, se procesan de forma nativa por el motor sin convertirse a SQL. Este es, precisamente, el motivo por el que este tipo de bases de datos XML se denominan “nativas” [30].

Una base de datos nativa XML define un modelo de datos lógico para cada documento XML y almacena y recupera documentos según ese modelo, el cual debe incluir elementos, atributos y el orden del documento, como mínimo. El documento XML debe ser la unidad de almacenamiento lógico, si bien no está restringido por ningún modelo de almacenamiento físico subyacente [29].

Este tipo de bases de datos XML funcionan según un concepto similar al de las tablas en las bases de datos relacionales. Por esto, al disponer de colecciones

de documentos independientes del esquema, se proporciona a la base de datos una alta flexibilidad y permite desarrollar aplicaciones de manera fácil.

No obstante, existen una serie de desventajas en el uso de este tipo de bases de datos:

- El que las colecciones de documentos sean independientes del esquema puede generar problemas de integridad de los datos.
- Al no existir mecanismo óptimos de agrupación, ordenación y enlazado de la información entre documentos, las posibilidades son muy limitadas en cuanto a realización de consultas.
- Debido a que a que para realizar una actualización en la base de datos se necesita recuperar el documento completo, modificar el dato deseado y volver a introducir todo en dicha base de datos, las actualizaciones son poco eficientes. Este problema se agrava, como cabe esperar, cuanto mayor sea el tamaño del documento.

Por lo tanto, se ha visto en este punto y el anterior que lo sistemas de almacenamiento de datos mediante ficheros XML presentan una serie de problemas. Esto hace necesaria la búsqueda de otro tipo de bases de datos.

2.3.2 Bases de datos orientadas a objetos

Con el auge de la programación orientada a objetos surgió la posibilidad de extender este concepto al de base de datos. Así, se tendría un sistema gestor de bases de datos orientadas a objetos (SGBDO), del inglés OBDMS *(Object DataBase Management System)*. De esta forma, se puede trabajar con objetos complejos, herencias y otras características propias de los lenguajes de programación orientados a objetos.

ODMG [31] *(Object Database Management Group, Grupo de Gestión de Bases de Datos de Objetos)* ha creado un modelo de datos orientado a objetos u ODM *(Object Data Model)* y un lenguaje de consulta orientado a objetos u OQL

(Object Query Languague) normalizados. Estos son el equivalente a la norma SQL en los sistemas de bases de datos relacionales [32].

Sin embargo, según [33], el modelo de base de datos relacional, que se verá en el próximo apartado, se ha establecido hasta ahora como el principal para las aplicaciones de procesado de datos debido, principalmente, a su simplicidad frente a otros modelos.

2.3.3 Bases de datos relacionales

El modelo de datos relacional organiza y representa los datos en forma de tablas o relaciones. El término relación representa una tabla de dos dimensiones formada por filas y columnas de datos. Cada fila de esta tabla contiene un conjunto de valores relacionados entre sí. Dicha tabla tiene un nombre, así como cada una de las columnas que la forman. Estos nombres clarifican el significado de los valores contenidos en la tabla. Adicionalmente, una base de datos relacional tiene los siguientes componentes:

- *Estructura de datos*: se trata de una colección de objetos abstractos formados por datos, dominios, tuplas, atributos y relaciones.
- *Operadores*: permiten manipular las estructuras de datos a partir de unas reglas bien definidas. Estos operadores, además del cambio de esquema, son la unión, diferencia, producto cartesiano, proyección y selección, es decir, los primitivos del álgebra relacional para manipulación de datos.
- *Definiciones de integridad*: conjunto de reglas y conceptos que posibilita expresar qué valores de datos pueden aparecer de forma válida en el esquema. Se incluyen las claves, la posibilidad de tener valores nulos y la reglas de integridad de claves primarias y de integridad referencial.

Integridad de claves primarias:

Las claves pueden usar usadas como identificadores de las tuplas en una relación dada, ya que a cada valor de una clave le corresponde únicamente una tupla y viceversa.

En el modelo relacional solo se puede encontrar una tupla determinada si se conoce el valor de una clave.

Una relación puede disponer de varias claves, aunque se suele emplear siempre la misma como identificador. A esta clave, empleada para identificar una relación, se la denomina clave primaria. El resto de claves se llaman claves secundarias o alternativas.

Como se ha comentado, la clave primaria es la que identifica a una relación por lo que no debe tomar valores nulos para evitar cualquier ambigüedad. A esta condición se la llama regla de integridad de claves primarias o regla de integridad de entidad. Es decir, ningún atributo de una clave primaria podrá valores nulos.

Integridad referencial:

Unas relaciones pueden hacer referencia a otras mediante claves primarias de estas. Adicionalmente al concepto de clave primaria, existe el de clave ajena, que se da en un atributo A de una relación R cuando se requiere que todos los valores de A no nulos existan en la clave primaria de alguna relación que no tiene por qué ser necesariamente distinta de R. Es decir, si A es un conjunto de atributos (A1, A2,… An), la definición anterior es válida si A toma valor nulo cuando alguno de sus componentes sea nulo.

Al identificar como clave ajena a un atributo, es recomendable definir las acciones a realizar en caso de intentar actualizar dicho atributo con valores no válidos. Estas acciones dependerán del significado de los datos [34].

Volviendo a las bases de datos relacionales, estas presentan unas características diferenciadoras frente al resto de modelos como son atomicidad de los valores de los atributos, la no repetición de tuplas, la no ordenación de tuplas y la no ordenación de los atributos. Debido a esto, la forma en que se almacenan los datos en el modelo de datos relacional no importa, contribuyendo esto a una mayor facilidad en el uso de la base de datos por parte del usuario. Al contrario de lo que ocurría en otros modelos de bases de datos vistos anteriormente, se pueden

realizar consultas para recuperar o almacenar información de forma flexible y con poder sobra la administración de la información. Adicionalmente, durante la fase de diseño de una base de datos relacional, se realiza un proceso de normalización mediante el cual cada relación queda descrita en términos de dependencia. Dicho proceso evita la redundancia de datos, lo que a su vez evita problemas al actualizar los datos y permite proteger la integridad de los mismos [35].

2.3.3.1 SQL

SQL (*Structured Query Language*), fue desarrollado originalmente por IBM a partir de los proyectos SEQUEL-XRM y System-R (1974-1977), resultados de la búsqueda de un prototipo de SGBD relacional. Así, a finales de la década de los ochenta surgió SQL, IBM y otras empresas comenzaron a utilizar SQL en sus SGBD relacionales, adquiriendo gran popularidad hasta convertirse en el lenguaje comercial para bases de datos relacionales más utilizado actualmente.

A la hora de denominar los diferentes elementos que se encuentra en la base de datos se emplean los siguientes términos, que difieren de los introducidos anteriormente en el apartado de bases de datos relacionales: se emplea el término tabla en lugar de relación, columna en lugar de atributo y fila en lugar de tupla.

Además, SQL presenta varias características [33]:

- *LMD, Lenguaje de Manipulación de Datos*: permite a los usuarios realizar consultas a la base de datos así como insertar, eliminar y modificar filas de esta.
- *LDD, Lenguaje de Definición de Datos*: posibilita crear, eliminar y modificar definiciones de tablas y vistas. Adicionalmente, se pueden establecer restricciones de integridad en las tablas en el momento de crearlas o posteriormente.
- *Disparadores y restricciones de integridad avanzadas*: siendo los disparadores acciones ejecutadas por el SGBD cuando se realizan modificaciones en la base de datos que cumple con las condiciones establecidas en dichos disparadores.

- *SQL incorporado:* permite ejecutar el código SQL en lenguajes anfitriones como C o Cobol, y SQL dinámico, que permite crear y ejecutar consultas durante el tiempo de ejecución.
- *Ejecución cliente-servidor y posibilidad de acceso a la base de datos de forma remota*: a partir de estas órdenes se controla la forma en que aplicaciones clientes se conectan con servidores de bases de datos de SQL o tienen acceso a los datos de estas bases de datos a través de la red.
- *Gestión de transacciones*: mediante distintas órdenes a partir de las cuales los usuarios controlan el modo en que se ejecutan dichas transacciones.
- *Seguridad*: mediante mecanismos ofrecidos por SQL para el control del acceso de los usuarios a los diferentes datos, tablas y vistas.

2.3.3.2 MySQL

MySQL [36] es un sistema de gestión de bases de datos relacionales rápido y robusto disponible públicamente desde 1996, si bien, su origen data de 1979, año en que comenzó su desarrollo.

El servidor de MySQL controla el acceso a los datos para ofrecer la posibilidad de que múltiples usuarios pueden hacer uso del sistema simultáneamente, permitiendo un acceso rápido al mismo y asegurando que solo los usuarios autorizados puedan tener dicho acceso. Por lo tanto, MySQL es un sistema de servidor multiusuario y multihilo.

Es capaz de trabajar con una buena cantidad de tipos de datos diferentes y hasta 32 índices por tabla, además mantiene un buen nivel de seguridad.

MySQL es portable a la mayoría de sistemas operativos [37] (UNIX, Mac, Solaris, Windows.), abarcando así a un mayor número de usuarios. Pero en realidad su gran aceptación es debida a la considerable cantidad de bibliotecas y herramientas que permiten trabajar con MySQL desde cualquier lenguaje de programación (C, C++, Java, PHP, Perl, Tcl, Phyton, etc.). Este hecho permite

que sus usuarios puedan realizar cualquier tipo de operación utilizando la interfaz adecuada a su lenguaje de programación.

MySQL utiliza SQL que, como ya se ha comentado en el apartado correspondiente, es el estándar de facto en lenguajes de consulta de bases de datos.

Cabe destacar que MySQL se distribuye bajo una licencia de código abierto, aunque también existen licencias comerciales en caso de que sean necesarias.

2.3.3.3 SQLite

SQLite [6] es un sistema de gestión de bases de datos relacional. Se trata de un lenguaje estructurado de consulta para acceso a bases de datos relacionales. El gran potencial que este tipo de bases de datos ofrece es el manejo del álgebra y el cálculo relacional en las consultas para recuperar la información de utilidad para nuestras aplicaciones.

Entrando particularmente en SQLite3 podemos entender SQLite como un proyecto de dominio público, sus primeras versiones datan del año 2000.

Algunas de las características que hacen de SQLite una buena opción para la gestión de bases de datos son:

- Características ACID (*Atomic, Consistent, Isolated and Durable*).
- Derivada de SQL92, se conserva su semántica y estructura.
- Baja necesidad de espacio, muy ligera.
- No necesita dependencias externas.
- Todo el código y documentación es de dominio público.
- Multiplataforma.

2.3.4 Discusión sobre las bases de datos

Una vez presentadas las diferentes tecnologías disponibles, se ha elegido SQLite como la más adecuada para este trabajo.

Por un lado, se han descartado las bases de datos XML debido a los aspectos negativos vistos en el apartado 2.3.1, siendo la dificultad para realizar consultas de forma rápida el principal inconveniente. Por otro lado, las bases de datos orientadas a objetos pueden estar más indicadas para aplicaciones con datos complejos o irregulares donde se tienen patrones previsibles. Por este motivo, se opta por las bases de datos relacionales como mejor opción.

Entre los dos sistemas gestores de bases de datos analizados, MySQL y SQLite, se ha optado por SQLite por los siguientes motivos.

El primero y más importante es que uno de los requisitos de este trabajo es que la aplicación resultante tiene que tener muy pocas dependencias externas. El lenguaje de programación Python incorpora una biblioteca para acceder a este tipo de bases de datos sin instalar dependencias adicionales. Aunque Python también permite acceder a bases de datos MySQL, es necesario instalar un paquete.

Otro motivo es el hecho de no necesitar un servidor de MySQL que se encargue de procesar las consultas. Como se mencionó en el punto 2.3.3.3, toda la funcionalidad de SQLite se encuentra dentro de una biblioteca que contiene todo lo necesario para realizar las consultas. Esta biblioteca viene incluida en la instalación de Python.

SQLite permite tener la base de datos completa contenida en un único fichero. Este puede ser almacenado o compartido entre diferentes máquinas con gran facilidad.

Por último, tanto los contenidos que se van a guardar en la base de datos como las consultas que se van a realizar van a ser sencillos y no son necesarias todas las funcionalidades que proporciona MySQL.

CAPÍTULO 3. ANÁLISIS DE LA APLICACIÓN

En este capítulo se muestran los requisitos, tanto funcionales como no funcionales, de la aplicación a desarrollar así como el análisis de cada componente de la aplicación. En este capítulo se amplía la información sobre la fase de desarrollo ofrecida en el primer capítulo de esta memoria incluyendo las tareas desarrolladas para la consecución de los objetivos del TFM.

3.1 Introducción

En este TFM se pretende desarrollar un sistema de monitorización activa de servidores adaptado a las necesidades de una filial de la empresa Hewlett-Packard.

El sistema que utilizan actualmente, basado en *scripts* en Bash, tiene importantes limitaciones que impiden la automatización de algunas tareas de monitorización.

Las limitaciones más importantes son: no poder monitorizar máquinas que se encuentran dentro de redes privadas y no disponer de una herramienta para

visualizar el estado de las máquinas. Esto obliga a la empresa a destinar recursos humanos en tareas que podrían ser automatizadas mediante las herramientas de *software* adecuadas.

La solución desarrollada en este TFM permitirá solventar estas limitaciones e incluirá algunas características adicionales que serán de gran utilidad para empresa. El esquema de esta solución se puede ver en la **figura 1**.

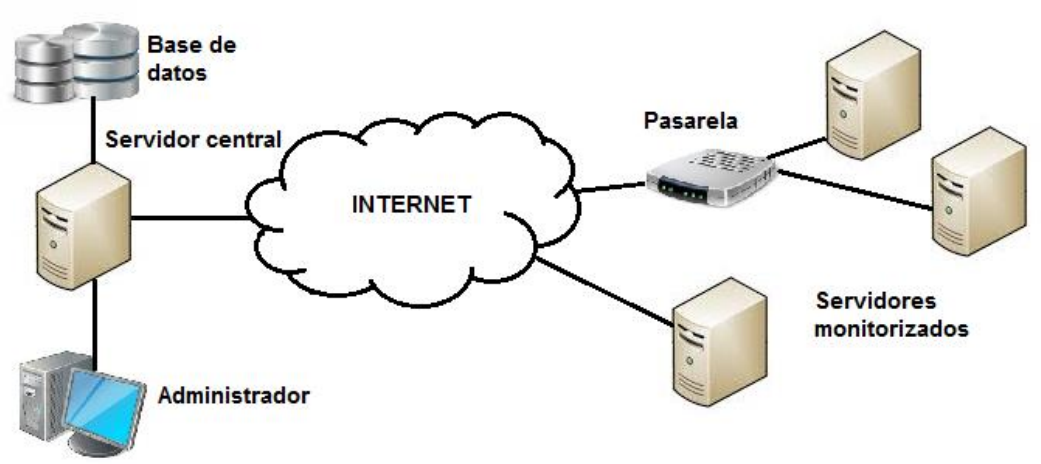


Figura 1. Esquema de la solución desarrollada en este trabajo.

En primer lugar tenemos el Servidor central que es donde se ejecutaran las herramientas desarrolladas en este TFM. Este servidor dispone acceso a la Base de datos donde se almacenan los resultados de los comandos ejecutados en las máquinas monitorizadas. Normalmente la Base de datos se encontrará dentro del Servidor central pero no tiene por qué ser así. El Servidor central debe disponer de acceso a Internet para poder conectarse a los servidores remotos.

Los Servidores monitorizados pueden estar conectados directamente a Internet, o encontrarse dentro de una red privada. En el caso de encontrarse dentro de una red privada es necesario establecer previamente un túnel a través de una Pasarela. La herramienta de monitorización ejecutada en el Servidor central se encarga de realizar las conexiones y de establecer los túneles.

Por último, el Administrador es la persona encargada de utilizar las herramientas desarrolladas para supervisar el correcto funcionamiento de las

máquinas monitorizadas. Como la interfaz que se ha diseñado para visualizar el estado de las máquinas se ha desarrollado utilizando tecnologías *web*, el Administrador puede trabajar conectado al Servidor central directamente o a través de Internet.

Como se vio en el capítulo 2, se ha decidido utilizar Python como lenguaje de programación principal para el desarrollo de este trabajo. Aunque el lenguaje de programación Python permite la programación orientada a objetos, se ha decidido utilizar la programación estructurada para la realización de este trabajo. Los motivos que han llevado a la elección de este paradigma de programación han sido el pequeño número de componentes que forman parte de las herramientas desarrolladas y la escasa interacción entre ellos.

3.2 Especificación de requisitos

Se pueden dividir los requisitos en dos grupos: funcionales y no funcionales [38]. Los requisitos funcionales definen las acciones que ha de ser capaz de llevar a cabo la aplicación, de forma que el acceso a la información sea adecuado; mientras que los requisitos no funcionales son aquellos que no forman parte de la funcionalidad principal de la aplicación (fiabilidad, disponibilidad, etc.).

3.2.1 Requisitos funcionales

Antes de comenzar el diseño y desarrollo del sistema de monitorización de servidores se han fijado una serie de requisitos funcionales que tienen que satisfacerse durante la realización del trabajo. A continuación se listan los requisitos funcionales que han sido fijados por el tutor del Trabajo Fin de Máster en la filial de la empresa Hewlett-Packard.

Herramienta de monitorización: implementación de una herramienta capaz de ejecutar comandos en servidores remotos y procesar los resultados para detectar posibles anomalías en su funcionamiento:

- La autentificación en estos servidores puede realizarse de dos formas distintas: mediante clave pública/privada o mediante usuario y contraseña.

- La información sensible debe almacenarse cifrada de forma que esta herramienta no comprometa la seguridad de otras máquinas.
- Los servidores pueden encontrarse dentro de una red privada que no cuente con conectividad con el exterior y, por tanto, la conexión con estas máquinas se realiza a través de un túnel SSH. La herramienta tiene que ser capaz de establecer estos túneles de forma transparente para el administrador.
- Los resultados de estos comandos tienen que almacenarse de forma que sea posible consultarlos en el futuro para poder obtener información adicional sobre el momento en el que empezó a producirse el problema.
- La herramienta tiene que ser capaz de realizar pruebas de conectividad para detectar máquinas fuera de servicio e identificar problemas en la red.
- El sistema tiene que ser capaz de notificar al administrador los problemas detectados mediante el envío de correos electrónicos. Esto permite que el administrador sea informado de problemas importantes incluso cuando no se encuentra en el puesto de trabajo.
- La configuración de la herramienta tiene que ser fácil de modificar y la modificación debe requerir el uso de del menor número posible de herramientas externas.
- En la configuración debe ser posible definir los siguientes parámetros: las máquinas monitorizadas, los servicios monitorizados en cada máquina o grupo de máquinas, las pruebas de conectividad realizadas en cada máquina, la configuración de los túneles en caso de ser necesaria y las notificaciones que recibe el administrador cuando se detecta un problema.

Herramienta de visualización: implementación de una interfaz *web* que permita visualizar de forma directa y sencilla el estado de todas las máquinas y servicios que están siendo monitorizados para que el administrador sea capaz de detectar los problemas rápidamente y tomar las medidas necesarias para solucionarlos.

- La herramienta debe mostrar, para cada máquina monitorizada, la respuesta de los comandos ejecutados de forma visual y textual. Además, se tiene que mostrar la fecha y hora de la última ejecución del comando y el tiempo que tardó en ejecutarse el comando.
- Para cada comando ejecutado debe ser posible visualizar información detallada (toda la información que fue recopilada y almacenada en la base de datos). En esta información debe encontrarse al menos: el comando ejecutado con todos los parámetros, el código de salida del comando, la salida textual del comando, la fecha y hora de ejecución y el tiempo que tardó en ejecutarse.
- La herramienta debe permitir listar todas las ejecuciones anteriores de un mismo comando en una máquina concreta. Esto puede ser de gran utilidad para identificar cuando comenzó a producirse el fallo de un comando en una máquina.
- La interfaz *web* tiene que refrescarse de forma periódica. El tiempo de refresco debe ser definido por el usuario.
- La configuración de la herramienta tiene que ser fácil de modificar y la modificación debe requerir el uso de del menor número posible de herramientas externas.
- En la configuración debe ser posible definir los siguientes parámetros: el puerto en el que escucha el servicio *web* y el intervalo de refresco de la interfaz.

3.2.2 Requisitos no funcionales

En este punto se listan los requisitos no funcionales que el sistema de monitorización de servidores debe cumplir:

- Las herramientas desarrolladas tienen que poder ser ejecutadas tanto en sistemas GNU/Linux como en entornos Windows mediante el uso de Cygwin [4].

- El lenguaje de programación principal, en el que se desarrollarán las herramientas que componen este trabajo, será Python 2 [5].
- Para acceder a las máquinas remotas y establecer los túneles se utilizará la herramienta OpenSSH [10].
- Para cifrar las contraseñas y otra información sensible se utilizará el estándar de cifrado por bloques AES (*Advanced Encryption Standard*) [39].
- Estas herramientas tienen que tener el menor número de dependencias externas posibles de forma que el despliegue de la misma sea rápido y no sea necesario realizar muchas modificaciones en la configuración de la máquina anfitriona.
- La herramienta de visualización de resultados tiene que utilizar tecnologías *web* maduras, de forma que sea compatible con la mayoría de los navegadores *web* disponibles en la actualidad. En concreto, la herramienta tiene que ser totalmente compatible con el navegador Mozilla Firefox [11].
- La configuración de las diferentes herramientas estará almacenada en ficheros de texto plano que puedan ser modificados con cualquier editor de texto.
- Todo el *software* utilizado debe tener una licencia compatible con el uso empresarial.

3.3 Análisis de los componentes de la aplicación

Una vez vistos los requisitos tanto funcionales como no funcionales de la aplicación se pasa a analizar sus funcionalidades.

Para ello se utilizarán los diagramas de casos de uso [40]. Estos muestran la relación entre los actores y los casos de uso del sistema. Representan las funcionalidades que ofrece el sistema en lo que se refiere a su interacción externa. En los diagramas de casos de uso se representa el sistema como una caja rectangular con el nombre en su interior. Los casos de uso están en el interior de la

caja del sistema, y los actores fuera. Cada actor está unido a los casos de uso en los que participa mediante una línea.

En los diagramas de casos de uso, el actor es una persona, un sistema informatizado o una organización, que realiza algún tipo de interacción con el sistema. Se representa mediante una figura humana.

El caso de uso es una descripción de la secuencia de interacciones que se producen entre un actor y el sistema, cuando el actor utiliza el sistema para llevar a cabo una tarea específica. Expresa una unidad coherente de funcionalidad, y se representa en el Diagrama de Casos de Uso mediante una elipse con el nombre del caso de uso en su interior. El nombre del caso de uso debe reflejar la tarea específica que el actor desea llevar a cabo usando el sistema.

Antes de pasar a describir en detalle el análisis de la aplicación, es necesario introducir algunos términos a los que se hace referencia en este capítulo y en los posteriores. El **glosario** es el siguiente:

- *host o servidor:* máquina que se quiere monitorizar y sobre la que se ejecutarán los comandos.
- *pasarela o túnel:* máquina que sirve de nodo intermedio cuando el *host* se encuentra ubicado dentro de una red privada sin conexión con el exterior.
- *grupo o grupo de hosts:* conjunto de máquinas que tienen algo en común que permite agruparlas como una unidad.
- *módulo:* conjunto de *scripts* que se utilizan para definir los comandos que se tienen que ejecutar en las máquinas remotas y el procesado que hay que realizar sobre los resultados del comando para determinar si la ejecución ha sido exitosa o no.
- *tarea:* entidad que relaciona un módulo (contiene el comando) y las máquinas en las que se ejecutará. Además, define el intervalo de tiempo que debe esperar la herramienta entre dos ejecuciones consecutivas del módulo.

- *base de datos:* contiene todos los resultados obtenidos tras la ejecución de los comandos y su procesado posterior.
- *configuración o ficheros de configuración:* contiene todos los parámetros de la aplicación que pueden ser modificados por el administrador. Son ficheros de texto plano que, entre otras cosas, permiten definir los *hosts*, los grupos y las tareas.
- *administrador:* persona encargada de supervisar el estado de las máquinas ayudándose de las diferentes herramientas desarrolladas en este trabajo.

El análisis de la aplicación desarrollada en este trabajo se puede dividir en las siguientes secciones:

- Herramienta de monitorización.
- Herramienta de visualización.
- Herramientas auxiliares.
    - Herramienta de cifrado de contraseñas.
    - Herramienta para probar las notificaciones.
- Base de datos.

3.3.1 Herramienta de monitorización

La herramienta de monitorización es el componente principal de esta aplicación. Es la encargada de realizar las tareas de monitorización, incluyendo el establecimiento de los túneles y las conexiones necesarias con las máquinas remotas, la ejecución de los comandos, el procesado de los resultados obtenidos en la ejecución de los comandos y el almacenamiento de la información en la base de datos para su consulta posterior.

El diagrama de casos de uso en el que se explican las distintas funcionalidades de la herramienta de monitorización se puede ver en la **figura 2**.

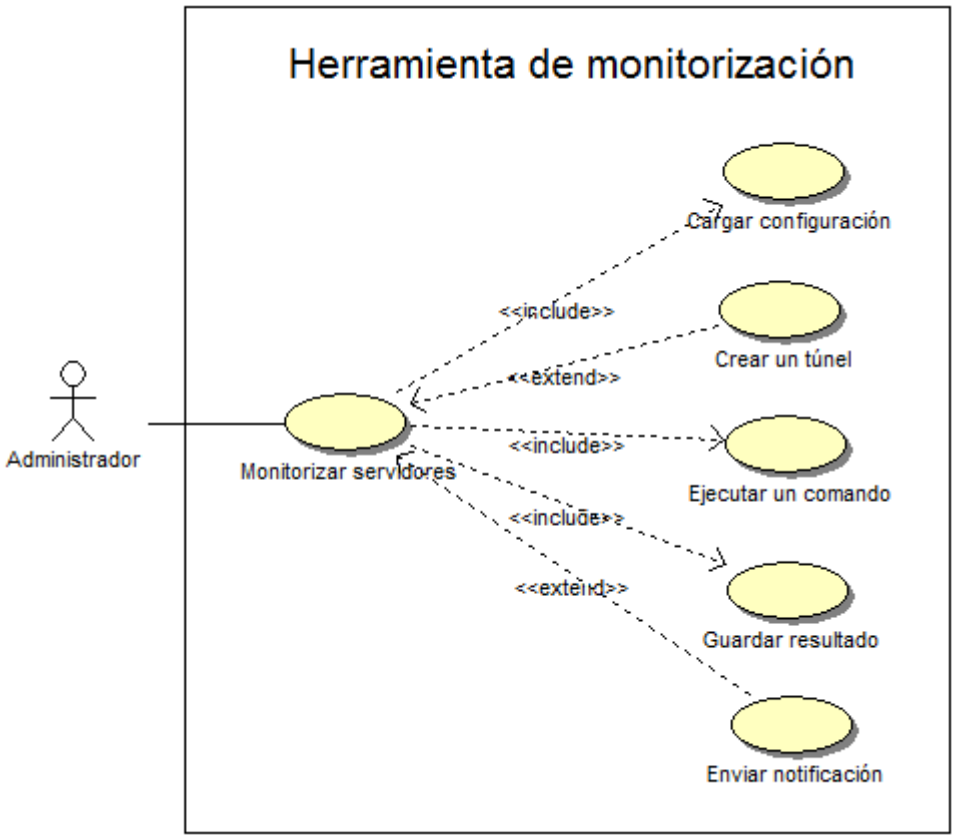


Figura 2. Diagrama de casos de uso de la herramienta de monitorización.

Este diagrama de casos de uso cuenta con un actor y seis casos de uso que se explican a continuación:

Actores:

- *Administrador*: es la persona encargada de supervisar el estado de las máquinas ayudándose de las diferentes herramientas desarrolladas en este trabajo.

Casos de uso:

- *Monitorizar servidores:* realiza la monitorización de todos los servidores que así han sido configurados. Las funcionalidades de este caso de uso se han separado en varios casos de uso más específicos que se explican a continuación.

- *Cargar configuración*: lee los ficheros de configuración y obtiene toda la información necesaria para llevar a cabo las tareas que debe desempeñar la herramienta. En ficheros de configuración se almacenan las máquinas monitorizadas y los comandos a ejecutar en cada una de ellas, entre otras cosas.
- *Crear un túnel*: crea un túnel SSH entre la máquina anfitriona, donde se está ejecutando la herramienta, y el servidor que va a ser monitorizado. Para crear el túnel se establece una conexión con una máquina que hace la función de Pasarela entre los dos extremos. Este caso de uso no es utilizado siempre, sólo en las máquinas que así lo tienen configurado.
- *Ejecutar un comando*: establece una conexión SSH entre la máquina anfitriona, donde se está ejecutando la herramienta, y el servidor que va a ser monitorizado. Esta conexión puede realizarse a través de un túnel si así está configurado. Después de establecida la conexión, se ejecuta el comando y se obtiene la salida del mismo.
- *Guardar el resultado*: procesa la información devuelta por el comando y almacena los resultados en la base de datos. Estos resultados serán leídos más adelante por la herramienta de visualización.
- *Enviar notificación*: envía un correo electrónico al Administrador en caso de detectarse algún error o alguna anomalía en la respuesta de un comando. Este caso de uso no es utilizado siempre, solo cuando se produce un error.

3.3.2 Herramienta de visualización

La herramienta de visualización es otro de los componentes principales de la aplicación. Es la encargada de crear un servidor *web* y de obtener la información almacenada en la base de datos y representarla de una forma conveniente para el administrador.

De un modo más general, permite visualizar de forma directa y sencilla el estado de todas las máquinas y servicios que están siendo monitorizados para que

el administrador sea capaz de detectar los problemas rápidamente y tomar las medidas necesarias para solucionarlos.

El diagrama de casos de uso en el que se explican las distintas funcionalidades de la herramienta de visualización se puede ver en **figura 3**.

Este diagrama de casos de uso cuenta con un actor y seis casos de uso que se explican a continuación:

Actores:

- *Administrador*: es la persona encargada de supervisar el estado de las máquinas ayudándose de las diferentes herramientas desarrolladas en este trabajo. Puede navegar por la interfaz *web* consultando la diferente información.

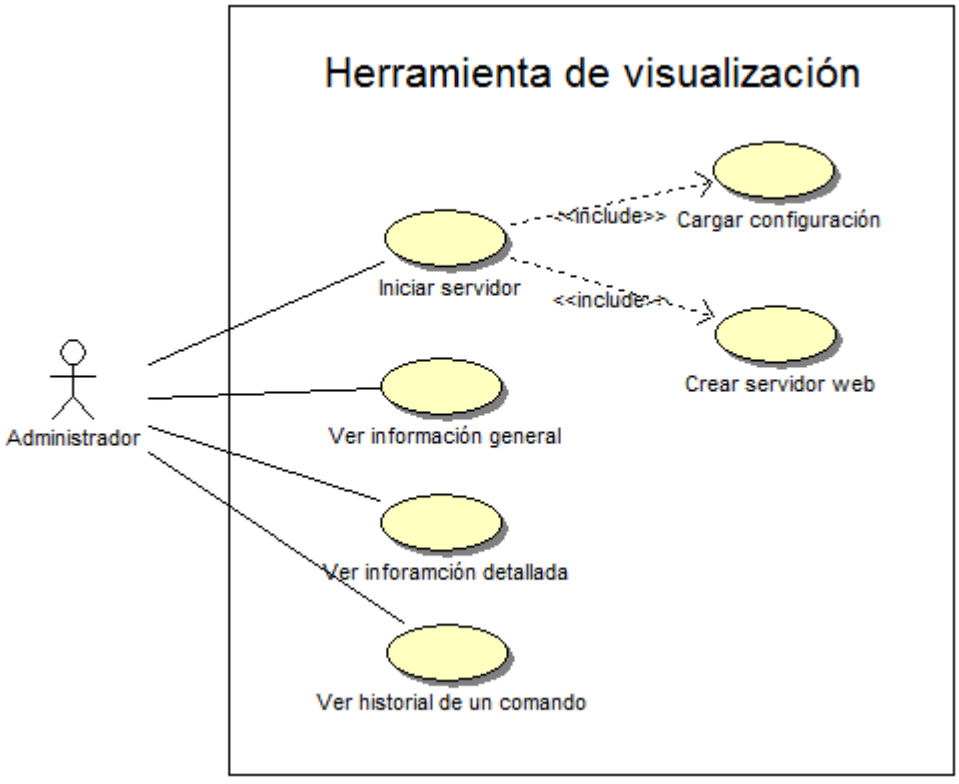


Figura 3. Diagrama de casos de uso de la herramienta de visualización.

Casos de uso:

- *Iniciar servidor:* inicia el servidor *web* que será el encargado de atender todas las peticiones solicitadas por el navegador *web*. Las funcionalidades de este caso de uso se han separado en varios casos de uso más específicos que se explican a continuación.
- *Cargar configuración*: lee los ficheros de configuración y obtiene toda la información necesaria para llevar a cabo las tareas que debe desempeñar la herramienta. En ficheros de configuración se almacenan las máquinas monitorizadas y los comandos a ejecutar en cada una de ellas, entre otras cosas.
- *Crear servidor web*: crea un servidor *web* que escucha en el puerto indicado en la configuración y atiende las peticiones generadas por el navegador *web*.
- *Ver información general*: muestra la información general en la que se puede observar el estado de todas las máquinas monitorizadas de un solo vistazo.
- *Ver información detallada*: muestra la información detallada de la ejecución de un comando en una máquina concreta. Esta información puede ser muy útil para el Administrador en caso de que se hayan producido errores.
- *Ver historial de un comando*: muestra una lista con las últimas ejecuciones de un comando en una máquina concreta. Esta información puede ser muy útil para detectar en que momento un comando empezó a fallar, y por tanto, en que momento empezaron a producirse los problemas.

3.3.3 Herramientas auxiliares

Además de las herramientas de monitorización y visualización, durante el transcurso de este trabajo se ha detectado la necesidad de desarrollar otras dos herramientas auxiliares que son utilizadas durante la configuración de la herramienta de monitorización.

En los siguientes apartados se explica la función de cada una de ellas mediante diagramas de casos de uso.

3.3.3.1 Herramienta de cifrado de contraseñas

La herramienta de cifrado de contraseñas es una herramienta auxiliar que se utiliza para cifrar las contraseñas que tienen que ser almacenadas en los ficheros de configuración.

Si las contraseñas se almacenaran en texto plano, cualquier persona con acceso a los ficheros de configuración podría acceder a las máquinas monitorizadas. Para que esto no suceda, la aplicación desarrollada obliga al Administrador a cifrar las contraseñas con una clave maestra, de forma que aunque alguien tenga acceso a los ficheros de configuración no se comprometa la seguridad de las máquinas monitorizadas.

El diagrama de casos de uso en el que se explican las distintas funcionalidades de la herramienta de cifrado de contraseñas se puede ver en la **figura 4**.

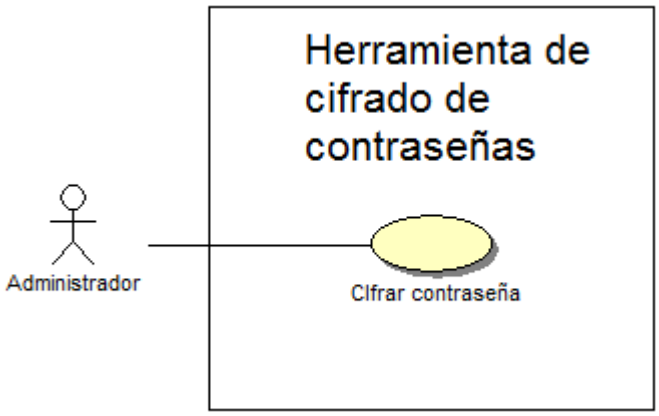


Figura 4. Diagrama de casos de uso de la herramienta de cifrado de contraseñas.

Este diagrama de casos de uso cuenta con un actor y un caso de uso que se explican a continuación:

Actores:

- *Administrador*: es la persona encargada de supervisar el estado de las máquinas ayudándose de las diferentes herramientas desarrolladas en este trabajo. Además, es el encargado de configurar la herramienta de forma adecuada.

Casos de uso:

- *Cifrar contraseña*: solicita al Administrador la contraseña a cifrar y la contraseña de cifrado y a partir de ellas devuelve la contraseña cifrada con AES [39] que debe incluirse en los ficheros de configuración.

3.3.3.2 Herramienta para probar las notificaciones

La herramienta para probar las notificaciones es un herramienta auxiliar que se utiliza para probar que la configuración de las notificaciones es correcta.

Debido a que existen bastantes parámetros de configuración relativos a las notificaciones, entre los que se incluyen los datos del servidor SMTP y los del correo electrónico del Administrador, se ha desarrollado esta herramienta para que el Administrador pueda comprobar que la configuración es correcta y que las notificaciones llegan correctamente a su bandeja de entrada.

El diagrama de casos de uso en el que se explican las distintas funcionalidades de la herramienta para probar las notificaciones se puede ver en la **figura 5**.

Este diagrama de casos de uso cuenta con un actor y tres casos de uso que se explican a continuación:

Actores:

- *Administrador*: es la persona encargada de supervisar el estado de las máquinas ayudándose de las diferentes herramientas desarrolladas en este trabajo.

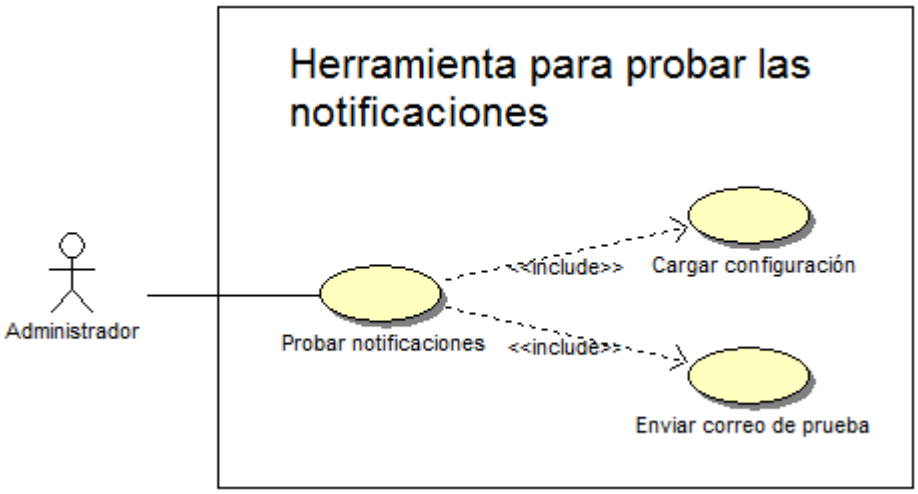


Figura 5. Diagrama de casos de uso de la herramienta para probar las notificaciones.

Casos de uso:

- *Probar notificaciones:* envía un correo electrónico de prueba al Administrador utilizando la información almacenada en la configuración. Las funcionalidades de este caso de uso se han separado en varios casos de uso más específicos que se explican a continuación.
- *Cargar configuración*: lee los ficheros de configuración y obtiene toda la información necesaria para llevar a cabo las tareas que debe desempeñar la herramienta. Los datos relevantes son los relativos a las notificaciones y a la configuración del correo electrónico del Administrador.
- *Enviar correo de prueba*: envía un correo electrónico de prueba al Administrador para que este pueda comprobar que ha configurado correctamente la aplicación.

3.3.4 Base de datos

En la base de datos es donde se almacenan los resultados obtenidos tras la ejecución de los comandos en las máquinas remotas. Estos datos pueden ser

consultados posteriormente por el Administrador utilizando la herramienta de visualización.

El diagrama de casos de uso en el que se explican las diferentes interacciones con la base de datos se puede ver en la **figura 6**.

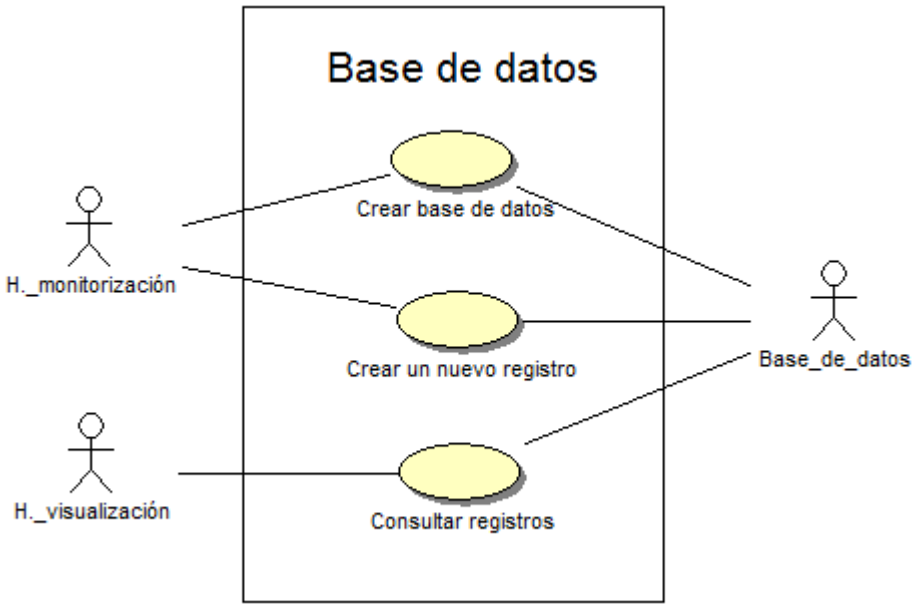


Figura 6. Diagrama de casos de uso de la base de datos.

Este diagrama de casos de uso cuenta con tres actores y tres casos de uso que se explican a continuación:

Actores:

- *H._monitorización*: la herramienta de monitorización es la encargada de ejecutar todas las acciones de forma autónoma sin necesidad de intervención por parte del Administrador.
- *H._visualización*: la herramienta de visualización es en la encargada de cargar la configuración y crear el servidor *web* que atenderá todas las peticiones realizadas por el navegador *web*.
- *Base_de_datos*: contiene todos los resultados obtenidos tras la ejecución de los comandos y su procesado posterior.

Casos de uso:

- *Crear base de datos*: crea una nueva base de datos con las tablas vacías en caso de que la base de datos no exista.
- *Crear un nuevo registro*: crea un nuevo registro en la base de datos con la información obtenida de la ejecución de un comando en una máquina monitorizada.
- *Consultar registros*: consulta los registros de la base de datos que posteriormente serán representarlos en la interfaz *web*. Existen diferentes tipos de consultas que son atendidas por la Base de datos.

CAPÍTULO 4. DISEÑO DE LA APLICACIÓN

En el capítulo cuarto se detalla el diseño de cada componente de la aplicación. Se profundiza en los pasos que han llevado al diseño final y se describen las partes que forman cada componente de la aplicación.

4.1 Introducción

Como ya se ha visto en el capítulo 3, la aplicación desarrollada en el este TFM está formada por varias herramientas que proporcionan la funcionalidad completa.

Debido a la extensión de este capítulo se ha explicado el diseño de cada componente en una sección distinta. Los componentes que forman parte de la aplicación son:

- Herramienta de monitorización.
- Herramienta de visualización.
- Herramientas auxiliares.
    - Herramienta de cifrado de contraseñas.
    - Herramienta para probar las notificaciones.
- Base de datos.

Adicionalmente, al final de este capítulo se incluye una sección que explica cómo está organizado el código fuente de la aplicación y los ficheros de configuración. También se incluye un pequeño resumen sobre el contenido de cada fichero.

4.2 Herramienta de monitorización

La herramienta de monitorización es el componente principal de esta aplicación. Es la encargada de realizar las tareas de monitorización, incluyendo el establecimiento de los túneles y las conexiones necesarias con las máquinas remotas, la ejecución de los comandos, el procesado de los resultados obtenidos en la ejecución de los comandos y el almacenamiento de la información en la base de datos para su consulta posterior.

El código fuente se encuentra dentro del fichero *pysmonitor.py* y es el *script* que hay que ejecutar si se quiere iniciar la herramienta de monitorización.

A continuación se detallan cada una de las diferentes tareas que lleva a cabo esta herramienta desde que se inicia hasta que guardan los resultados en la base de datos. Además, se describe en detalle cómo funcionan las tareas, el sistema de módulos y el sistema de notificaciones.

4.2.1 Inicio de la herramienta y ejecución de las tareas

La primera tarea que realiza es comprobar que existen todos los ficheros de configuración y que estos tienen una configuración válida. En caso de que alguno

de ellos no existiera o se encontraran errores se muestra un mensaje de error en la consola indicando el error y en que parte de la configuración se encuentra. Después de mostrar este mensaje, se termina la aplicación puesto que no es posible continuar sin una configuración válida.

Existen cuatro ficheros de configuración que se explican de forma resumida a continuación. Para ver información más detallada y el formato de estos ficheros véase la sección 5.2.1.

- *config.cfg*: configuración general de la aplicación. Algunos de los parámetros son: la ubicación de la base de datos, la configuración de las notificaciones, etc.
- *hosts.cfg*: configuración de las máquinas que van a ser monitorizadas. Para cada máquina se incluye su dirección IP y las credenciales con los que se accede a la máquina.
- *groups.cfg*: configuración de los grupos de máquinas y los túneles. Permite agrupar los *hosts* en grupos y establecer un túnel para cada grupo de *host* en caso de ser necesario. Esto facilita la ejecución de un mismo comando en varias máquinas a la vez.
- *tasks.cfg*: configuración de las tareas. Cada tarea define el módulo (el comando que se ejecuta), el intervalo de tiempo entre ejecuciones y los hosts o grupos en los que se ejecuta el comando.

Una vez cargada la configuración se comprueba la existencia de la base de datos. Si la base de datos definida en la configuración no se encuentra, se pregunta al administrador si quiere crear una base de datos nueva. En caso afirmativo se crea. Si el administrador decide no crear la base de datos se termina la aplicación puesto que no es posible continuar sin una base de datos válida.

El siguiente paso es comprobar si alguna de las máquinas (*hosts*) tiene definida la contraseña en el fichero de configuración *hosts.cfg*. Existen dos formas de autentificarse en una máquina remota con SSH. La forma más segura es

mediante la utilización de un par de claves pública/privada que tiene que ser configurada previamente por el administrador. Otra forma posible es indicar en el fichero *hosts.cfg* la contraseña que ha sido cifrada previamente utilizando la herramienta auxiliar incluida en la aplicación para esta función. Véase la sección 4.4.1.1.

Si alguna de las máquinas tiene definida la contraseña en el fichero *hosts.cfg*, se pregunta al administrador por la clave maestra que se utilizó para cifrar las contraseñas. Una vez descifradas las contraseñas se elimina la clave proporcionada por el administrador de la memoria por motivos de seguridad. Si ninguna de las máquinas tiene definida la autentificación en el fichero *hosts.cfg* se omite este paso.

En este momento la herramienta ya tiene cargada toda la información necesaria para empezar a lanzar tareas. Debido a que la ejecución de algunos comandos (como la comprobación de disco o las copias de seguridad) puede tardar mucho tiempo en ejecutarse, se ha optado por ejecutar cada tarea en un hilo de ejecución en vez de ejecutar las tareas de forma secuencial. En caso de ejecutar las tareas de forma secuencial, un comando que tardase mucho tiempo podría impedir la correcta ejecución de otros comandos.

Como ya se ha comentado, en este momento la herramienta crea un hilo para cada tarea definida en la configuración y se queda esperando a que los hilos terminen. Debido a que los hilos ejecutan las tareas de forma repetitiva cada cierto tiempo, estos no terminan nunca y por lo tanto la herramienta solo termina cuando el administrador decide terminar la ejecución de la herramienta. Durante el lanzamiento de las tareas se establece una pequeña pausa aleatoria que sigue una distribución gaussiana entre en lanzamiento de dos tareas consecutivas para evitar los problemas que puede causar la ejecución de todas las tareas al mismo tiempo, sobre todo si son muchas.

En la **figura 7** se pueden ver los mensajes mostrados en la consola por la aplicación.

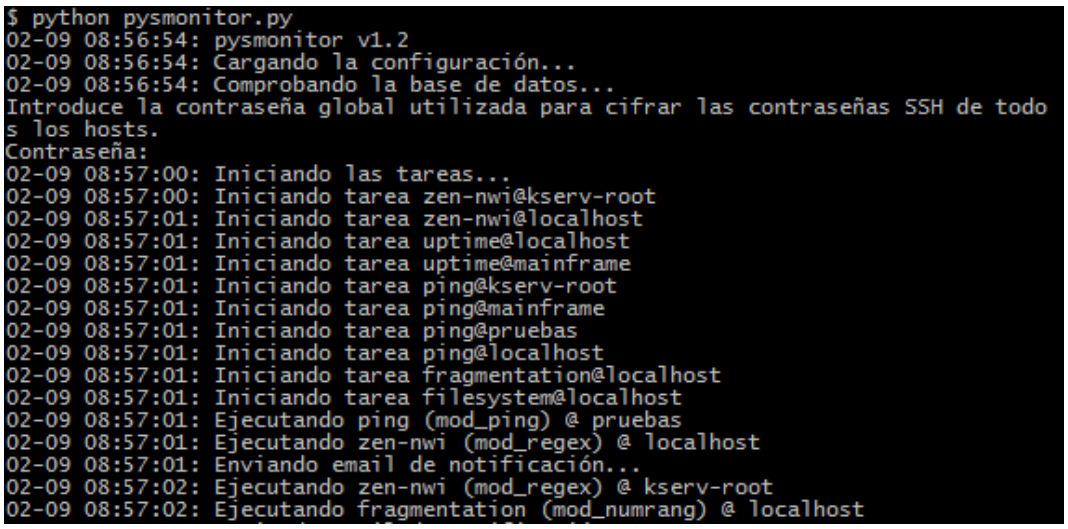


Figura 7. Mensajes mostrados en la consola por la herramienta de monitorización.

4.2.2 Tareas

Cada tarea tiene definido en el fichero de configuración *tasks.cfg* el nombre de la tarea, un módulo (contiene el comando a ejecutar), los parámetros del módulo (si les hay), un intervalo de tiempo entre ejecuciones y las máquinas o grupos de máquinas en las que se ejecutará el módulo.

En el apartado anterior se explicaba la carga en memoria de la configuración de las tareas y la creación un hilo de ejecución para cada tarea. Durante la creación del hilo se pasan como argumentos los parámetros de la tarea.

Lo primero que realiza el hilo de ejecución de una tarea es cargar de forma dinámica el módulo y definir los parámetros del módulo (si les hay).

Después se realizan dos comprobaciones:

- Comprobar si el comando se ejecuta en la máquina local o en la máquina remota. Por ejemplo, el comando "*ping*" se ejecuta en la máquina local mientras que un comando para ver la ocupación de los discos se ejecuta de forma remota.
- Comprobar si es necesario establecer un túnel para conectar con la máquina remota o no. Es típico que en entornos empresariales los

servidores se encuentren dentro de una red privada y el acceso desde el exterior se realice a través de un túnel.

En la **figura 8** se ilustran las cuatro situaciones que se pueden dar y que son contempladas por la aplicación.

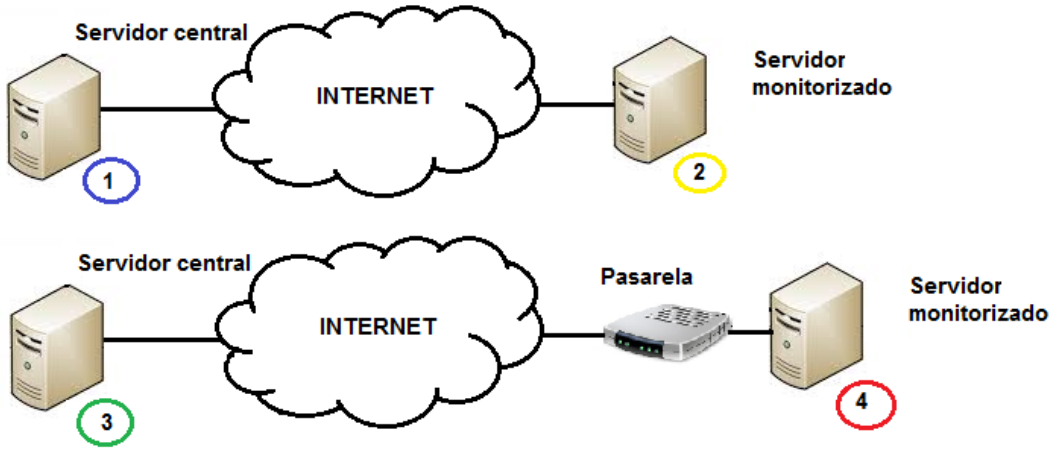


Figura 8. Situaciones que se pueden dar durante la ejecución del comando.

Los casos posibles mostrados en la figura son:

- 1) El comando se ejecuta en el Servidor central y no es necesario establecer un túnel con el Servidor monitorizado. En este caso no se utiliza la herramienta SSH. Un ejemplo práctico para ilustrar este caso es la herramienta "*ping*".
- 2) El comando se ejecuta en el Servidor monitorizado y no es necesario establecer un túnel con el Servidor monitorizado. En este caso se crea una conexión SSH entre las dos máquinas.
- 3) El comando se ejecuta en el Servidor central y es necesario establecer un túnel con el Servidor monitorizado. En este caso se utiliza la herramienta SSH para establecer el túnel entre las dos máquinas. Un ejemplo práctico para ilustrar este caso, al igual que en el caso 1, es la herramienta "*ping*".

- 4) El comando se ejecuta en el Servidor monitorizado y es necesario establecer un túnel con el Servidor monitorizado En este caso se utiliza la herramienta SSH para establecer el túnel entre las dos máquinas y después se crea una conexión SSH entre las dos máquinas.

En este momento se guarda la marca de tiempo UNIX de inicio y se ejecuta el comando en la máquina remota. Se obtiene la salida del comando y el código de salida del comando. También se tienen en cuenta los errores que se pueden producir al establecer el túnel y la conexión con la máquina remota, incluyendo errores de conectividad y de autenticación.

Cuando termina de ejecutarse el comando se guarda la segunda marca de tiempo UNIX que indica el instante en el que finalizó la ejecución. A continuación, se pasa la salida del comando y el código de salida del comando al módulo para que procese la salida. El módulo es el encargado de determinar si el resultado es satisfactorio o se produjo algún error. Como se verá más adelante, los módulos devuelven un código numérico que indica el resultado del comando y un resumen de la respuesta del comando que se mostrará en la herramienta de visualización.

Llegados a este punto se guarda en la base de datos la información de la tarea, la información que se ha recogido de la ejecución del comando y las marcas de tiempo. Una vez guardados los datos, si se produjo algún error en la ejecución del comando, se comprueba la configuración de notificaciones para ver si es necesario enviar un correo electrónico al administrador. Las notificaciones se explican con más detalle en el apartado 4.2.4.

Por último, el hilo de ejecución permanece dormido durante el tiempo necesario para que la siguiente ejecución de la tarea se produzca con el intervalo de tiempo fijado en la configuración.

4.2.3 Módulos

Los módulos son un conjunto de clases que se utilizan para definir los comandos que se tienen que ejecutar en las máquinas remotas y el procesado que hay que realizar sobre los resultados del comando para determinar si la ejecución ha sido exitosa o no.

En los siguientes sub-apartados se explica la estructura de un módulo, las instrucciones para la creación de un módulo nuevo y la lista de módulos incluidos por defecto en la aplicación.

4.2.3.1 Estructura de los módulos

Todos los módulos están escritos en lenguaje Python 2 [5], como el resto de la aplicación, y se encuentran en el directorio *mods*. Por defecto la aplicación incluye unos cuantos módulos de propósito general pero la aplicación está preparada para que el administrador pueda crear sus propios módulos en caso de ser necesario.

Cada módulo tiene que comenzar con el prefijo *mod_* y después el nombre del módulo, que tiene que coincidir con el nombre de la clase definida en su interior.

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# Copyright (c)

import const

# número de argumentos requeridos
mod_nombremod_num_args = 1

# documentación
class mod_nombremod(object):
      def __init__(self, tunnel, arg1):
            self.local_exec = False
            self.command = 'uptime'
            self.param = arg1

      def get_local_exec(self):
            return self.local_exec

      def get_cmd(self):
            return self.command
```

```
    def parse_result(self, cmd_result, cmd_exit):
        if cmd_exit != 0: # error en el comando
            return (const.RES_ERR_CMD, '')

        # lógica del módulo

        return (result, cmd_result)
```

Figura 9. Esqueleto de un módulo.

En la **figura 9** se muestra el esqueleto de un módulo llamado *nombremod* y a continuación se explican las partes más importantes:

- En la constante global *mod_nombremod_num_args* se indica el número de parámetros que acepta el módulo siendo 0 si el módulo no tiene parámetros.
- En el constructor de la clase, *__init__*, se reciben como parámetros *tunnel* (si el comando se tiene que ejecutar a través de un túnel) y los argumentos que necesite el módulo. El número de argumentos tiene que coincidir con lo declarado en la constante *mod_nombremod_num_args*.
- Dentro del constructor se establecen los atributos: *local_exec* (si el comando se tiene que ejecutar en la máquina local o en la máquina remota), *command* (el comando que se va a ejecutar) y los atributos que necesite el módulo.
- El método *parse_result* recibe como parámetros: *cmd_result* (el resultado de la ejecución del comando) y *cmd_exit* (el código de salida del comando).
- Dentro del método *parse_result* se incluyen el código que determina si el resultado del comando ha sido correcto o no y devuelve una tupla que incluye: *result* (el código numérico que indica si el resultado ha sido satisfactorio o hay algún problema) y *cmd_result* (un resumen de la respuesta del comando). En la **figura 10** se pueden ver los valores que puede tomar código numérico, aunque para obtener información detallada se recomienda consultar el fichero *const.py*.

Además de la información presentada, es recomendable ver los comentarios del código fuente de los módulos incluidos en la aplicación para obtener más información.

```
RES_OK              = 0   # ok
RES_WARNING         = 1   # warning
RES_ERROR           = 2   # error
RES_CRITICAL        = 3   # critical
RES_ERR_SSHCON      = 90  # ssh connection error
RES_ERR_SSHAUT      = 91  # ssh authentication error
RES_ERR_TUNCON      = 92  # tunnel connection error
RES_ERR_TUNAUT      = 93  # tunnel authentication error
RES_ERR_CMD         = 97  # command exit code != 0
RES_ERR_MOD         = 98  # mod error
RES_ERR_UNK         = 99  # unknown error
```

Figura 10. Valores que puede tomar el código numérico *results*.

4.2.3.2 Módulos incluidos en la aplicación

La aplicación incluye por defecto algunos módulos de propósito general con dos objetivos. El principal es que el administrador pueda empezar a trabajar con la herramienta lo antes posible sin tener que programar sus propios módulos. El segundo es servir como complemento de la documentación a la hora de programar nuevos módulos.

A continuación se listan los módulos incluidos:

- *mod_filesystem.py*: módulo para monitorizar la ocupación de los diferentes puntos de montaje de una máquina. Utiliza el comando *"df -h"* para obtener la información y permite que el administrador defina los puntos de montaje que quiere monitorizar y cuáles son los diferentes umbrales de alerta.
- *mod_numrang.py*: módulo que permite definir varios niveles de alerta en función del número devuelto en la salida de un comando. Este módulo está pensado para ejecutar un comando que devuelve un resultado numérico y comparar el resultado con unos umbrales.

- *mod_ping.py*: módulo que permite comprobar la conectividad de una máquina y el estado de la red. Utiliza el comando *"ping"* para obtener la información.
- *mod_regex.py*: módulo que comprueba la existencia de una expresión regular en la salida de un comando y devuelve el código de alerta especificado por el administrador.
- *mod_uptime.py*: módulo que permite definir varios niveles de alerta en función del número de días que la máquina lleva encendida. Utiliza el comando *"uptime"* para obtener la información.

Para obtener más información sobre el funcionamiento de los módulos consulte la sección 5.2 y los comentarios en el código fuente de los módulos.

4.2.4 Sistema de notificaciones

El sistema de monitorización, además de guardar el resultado de todos los comandos para su posterior consulta, permite definir las notificaciones que se enviarán por correo electrónico al administrador cuando se produzca algún error.

Como se explicó en el capítulo 1, el sistema de notificaciones es muy importante para que no pasen desapercibidos los errores que ocurran en los servidores monitorizados y éstos sean resueltos lo antes posible.

Para que el sistema de notificaciones funcione, éste tiene que estar activado y correctamente configurado. La configuración se encuentra dentro del fichero *config.cfg* y el significado de los parámetros se puede ver la sección 5.2.1.

Uno de los parámetros que se define dentro de la configuración de notificaciones es el nivel de alerta a partir del cual se envían notificaciones (*alert_level*). Este parámetro puede tomar uno de estos tres valores: *warning*=1, *error*=2 o *critical*=3.

El modo de funcionamiento es sencillo. Una vez que la herramienta de monitorización ha ejecutado un comando en la máquina remota y ha procesado la

respuesta, dispone de un código numérico que indica el resultado del comando. Se compara ese código numérico (en la **figura 10** se pueden ver los valores posibles) con el nivel de alerta definido en la configuración. Si el código numérico es mayor o igual se envía un correo electrónico al administrador.

En la **figura 11** se puede ver el contenido del correo electrónico cuando se recibe una notificación.

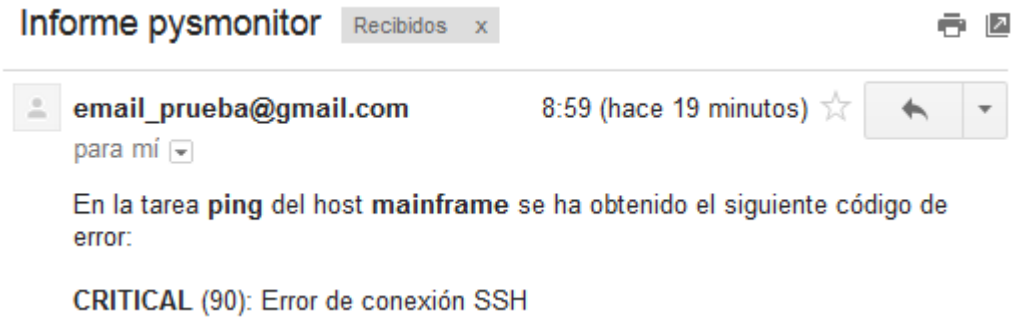


Figura 11. Correo electrónico recibido cuando se produce una notificación.

4.3 Herramienta de visualización

La herramienta de visualización es otro de los componentes principales de la aplicación. Es la encargada de crear un servidor *web* y de obtener la información almacenada en la base de datos y representarla de una forma conveniente para el administrador.

De un modo más general, permite visualizar de forma directa y sencilla el estado de todas las máquinas y servicios que están siendo monitorizados para que el administrador sea capaz de detectar los problemas rápidamente y tomar las medidas necesarias para solucionarlos.

El código fuente se encuentra dentro del fichero *pyserv.py* y es el *script* que hay que ejecutar si se quiere iniciar la herramienta de visualización.

A continuación se detallan cada una de las diferentes tareas que lleva a cabo esta herramienta desde que se inicia hasta que se visualiza la información en un

navegador *web*. Además, se explican las diferentes partes de la interfaz *web* y las funcionalidades que proporciona.

4.3.1 Inicio de la herramienta y creación del servidor *web*

Al igual que la herramienta de monitorización, la primera tarea que realiza es comprobar que existen todos los ficheros de configuración y que estos tienen una configuración válida. En caso de que alguno de ellos no existiera o se encontraran errores se muestra un mensaje de error en la consola indicando el error y en que parte de la configuración se encuentra. Después de mostrar este mensaje, se termina la aplicación porque no es posible continuar sin una configuración válida.

Una vez cargada la configuración, se comprueba la existencia de la base de datos. Si la base de datos definida en la configuración no se encuentra, se termina la aplicación puesto no es posible continuar sin una base de datos válida.

Llegados a este punto se crea el servidor *web* que será el encargado de atender las peticiones generadas por el navegador *web*. Para ello se hace uso de la biblioteca *BaseHTTPServer* [41] que incluye Python 2. Esta biblioteca tiene las clases y las funciones necesarias para crear un servidor *web* sencillo. Existen otras bibliotecas más avanzadas pero con las funcionalidades que proporciona *BaseHTTPServer* es suficiente para el alcance de este trabajo.

En la **figura 12** se puede ver el código fuente utilizado para crear el servidor *web*.

```
from BaseHTTPServer import BaseHTTPRequestHandler, HTTPServer

# clase que recibe las peticiones HTTP
class RequestHandler(BaseHTTPRequestHandler):
      def do_GET(self):
            path, query = (self.path.split('?', 1) + [None])[:2]

            html = u'''<!doctype HTML><html><head>
            <meta charset="UTF-8">
            <title>pysmonitor v%s</title>
            </head>
            <body>
```

```
            aquí va la lógica que genera el código HTML

            </body>
            </html>
            '''

            self.send_response(200)
            self.end_headers()
            self.wfile.write(html.encode("utf-8"))

# iniciamos el servidor
HTTPServer(('', puerto_escucha), RequestHandler).serve_forever()
```

Figura 12. Código fuente utilizado para crear el servidor *web*.

El servidor se mantiene a la escucha de peticiones hasta que termina la aplicación. El puerto en el que escucha el servidor se puede definir en el fichero de configuración *config.cfg*.

En la **figura 13** se pueden ver los mensajes mostrados en la consola por la aplicación:

```
$ python pyserv.py
02-09 08:51:53: pysmonitor (server) v1.2
02-09 08:51:53: Cargando la configuración...
02-09 08:51:53: Comprobando la base de datos...
02-09 08:51:53: Cargando información de las tareas...
02-09 08:51:53: Iniciando el servidor...
02-09 08:51:53: Escuchando en el puerto 8000.
127.0.0.1 - - [02/Sep/2015 08:52:08] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [02/Sep/2015 08:52:13] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [02/Sep/2015 08:52:13] "GET /history?host=mainframe&task=uptime&li
mit=150 HTTP/1.1" 200 -
127.0.0.1 - - [02/Sep/2015 08:52:15] "GET /details?id=2716 HTTP/1.1" 200 -
```

Figura 13. Mensajes mostrados en la consola por la herramienta de visualización.

4.3.2 Interfaz *web*

Como se vio en los requisitos funcionales del capítulo 3, la interfaz tiene que mostrar la información almacenada en la base de datos con mayor o menor detalle dependiendo de la sección de la interfaz en la que estemos.

En esta vista la herramienta muestra, para cada máquina monitorizada, la respuesta de los comandos ejecutados de forma visual y textual. Además, muestra la fecha y hora de la última ejecución del comando y el tiempo que tardó en ejecutarse el comando.

En la **figura 14** se puede ver la vista principal de la interfaz.

Host	Task	Last Check	Duration	Status	Status Information	Details
kserv-root	zen-nwi	02-07-2015 01:34:57	2.4s	OK	Se ha encontrado "ru[n]+ing"	Details
kserv-root	ping	02-07-2015 01:35:58	0.0s	OK	64 bytes from 192.168.1.102: icmp_seq=1 ttl=64 time=0.283 ms	Details
mainframe	uptime	02-07-2015 01:34:58	3.59s	OK	17 days	Details
mainframe	ping	02-07-2015 01:36:01	0.17s	OK	64 bytes from 192.168.1.102: icmp_seq=1 ttl=64 time=0.264 ms	Details
pruebas	ping	02-07-2015 01:35:00	10.0s	CRITICAL	No existe conectividad con el host	Details
localhost	zen-nwi	02-07-2015 01:35:05	0.17s	OK	Se ha encontrado "ru[n]+ing"	Details
localhost	uptime	02-07-2015 01:34:59	0.16s	OK	0 days	Details
localhost	ping	02-07-2015 01:36:02	0.0s	OK	64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.018 ms	Details
localhost	fragmentation	02-07-2015 01:34:59	0.16s	OK	100	Details
localhost	filesystem	02-07-2015 01:35:00	0.17s	OK	/tmp = 1%,	Details

(Actualizando cada 5 segundos)

Figura 14. Vista principal de la interfaz *web*.

Como se puede ver en la figura, cuando la ejecución del comando ha sido correcta el color es verde, mientras que cuando se ha producido un error el color es distinto, siendo naranja para el nivel *warning* y rojo para *critical*.

El administrador puede ordenar la tabla por máquinas (*host*) o por tareas (*task*). Al hacer clic en los enlaces de la columna detalles (*details*) se muestra la información detallada del comando ejecutado (**figura 15**). Al hacer clic en los enlaces de la columna última comprobación (*last check*) se muestra el historial de ejecuciones del comando para esa máquina concreta (**figura 16**). La información de esta tabla es refrescada automáticamente cada cierto tiempo, el intervalo de refresco es fijado en el fichero *config.cfg*.

Para cada comando ejecutado debe ser posible visualizar información detallada (toda la información que fue recopilada y almacenada en la base de datos). La representación de esta información se puede ver en la **figura 15**.

La vista detallada, además de contener la información que aparecía en la vista general, contiene: el comando ejecutado con todos los parámetros, el código de salida del comando, la salida textual del comando, la fecha y hora de ejecución y el tiempo que tardó en ejecutarse.

Host	mainframe
Task	uptime
Last Check	02-07-2015 01:34:58
Duration	3.59s
Status	OK
Status Information	17 days
Command	uptime
Command exit code	0
Command result	01:36:25 up 17 days, 12:48, 0 users, load average: 0,14, 0,16, 0,24

Figura 15. Vista detallada de la interfaz *web*.

Por último, la herramienta debe permitir listar todas las ejecuciones anteriores de un mismo comando en una máquina concreta. Esto puede ser de gran utilidad para identificar cuando comenzó a producirse el fallo de un comando en una máquina. La representación de esta información se puede ver en la **figura 16**.

Host	Task	Last Check	Duration	Status	Status Information	Details
mainframe	uptime	02-07-2015 01:34:58	3.59s	OK	17 days	Details
mainframe	uptime	02-07-2015 01:31:56	2.05s	OK	17 days	Details
mainframe	uptime	01-07-2015 22:54:24	3.1s	OK	17 days	Details
mainframe	uptime	01-07-2015 22:53:30	2.25s	CRITICAL	Error de autentificación tunel SSH	Details
mainframe	uptime	01-07-2015 09:41:37	2.42s	CRITICAL	Error de autentificación tunel SSH	Details
mainframe	uptime	01-07-2015 09:40:59	9.7s	OK	16 days	Details
mainframe	uptime	01-07-2015 08:05:31	7.65s	OK	16 days	Details
mainframe	uptime	01-07-2015 06:34:07	2.22s	OK	16 days	Details

Figura 16. Vista del historial de ejecuciones de un comando en la interfaz *web*.

En la figura se puede apreciar que la información mostrada es casi igual que la mostrada en la vista general pero permite ver cuándo comenzó a producirse un problema y la vista detallada de los comandos cuando se produjo el problema.

4.4 Herramientas auxiliares

Además de las herramientas de monitorización y visualización, durante el transcurso de este trabajo se ha detectado la necesidad de desarrollar otras dos herramientas auxiliares que son utilizadas durante la configuración de la herramienta de monitorización.

En los siguientes apartados se explica la función de cada una de ellas.

4.4.1.1 Herramienta de cifrado de contraseñas

La herramienta de cifrado de contraseñas es una herramienta auxiliar que se utiliza para cifrar las contraseñas que tienen que ser almacenadas en los ficheros de configuración.

Si las contraseñas se almacenaran en texto plano, cualquier persona con acceso a los ficheros de configuración podría acceder a las máquinas monitorizadas. Para que esto no suceda, la aplicación desarrollada obliga al Administrador a cifrar las contraseñas con una clave maestra, de forma que aunque alguien tenga acceso a los ficheros de configuración no se comprometa la seguridad de las máquinas monitorizadas.

El código fuente se encuentra dentro del fichero *pypassword.py* y es el *script* que hay que ejecutar si se quiere iniciar la herramienta de cifrado de contraseñas.

Lo primero que realiza la herramienta es recopilar toda la información que necesita. Solicita al administrador introducir la clave global utilizada para cifrar todas las contraseñas (debe tener al menos 8 caracteres). A continuación solicita la contraseña SSH a cifrar.

Una vez conoce las dos contraseñas, cifra la contraseña SSH con la clave global utilizando AES [39], y por último, codifica el resultado en formato base64 para que el resultado final sea una cadena de caracteres formada por caracteres

ASCII que puedan ser copiados fácilmente. Este resultado es mostrado en la consola y se termina la ejecución de la herramienta.

A partir de aquí es tarea del administrador copiar la contraseña cifrada en el fichero de configuración *hosts.cfg*.

En la **figura 17** se pueden ver los mensajes mostrados en la consola por la aplicación.

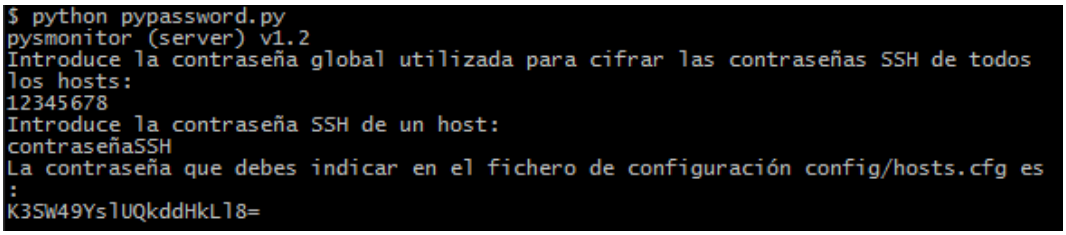


Figura 17. Mensajes mostrados en la consola por la herramienta de cifrado de contraseñas.

4.4.1.2 Herramienta para probar las notificaciones

La herramienta para probar las notificaciones es un herramienta auxiliar que se utiliza para probar que la configuración de las notificaciones es correcta.

Debido a que existen bastantes parámetros de configuración relativos a las notificaciones, entre los que se incluyen los datos del servidor SMTP y los del correo electrónico del Administrador, se ha desarrollado esta herramienta para que el Administrador pueda comprobar que la configuración es correcta y que las notificaciones llegan correctamente a su bandeja de entrada.

El código fuente se encuentra dentro del fichero *pyemailtest.py* y es el *script* que hay que ejecutar si se quiere iniciar la herramienta para probar las notificaciones.

Lo primero que realiza la aplicación al ser lanzada es cargar la configuración referente a las notificaciones y comprobar que están definidos todos los parámetros necesarios.

A continuación intenta enviar un correo electrónico con un mensaje de prueba. Si se produce algún error al intentar conectar con el servidor SMTP o de alguna otra clase se muestra en la consola. Si se envía correctamente se pide al administrador que compruebe su bandeja de entrada.

En la **figura 18** se pueden ver los mensajes mostrados en la consola por la aplicación.

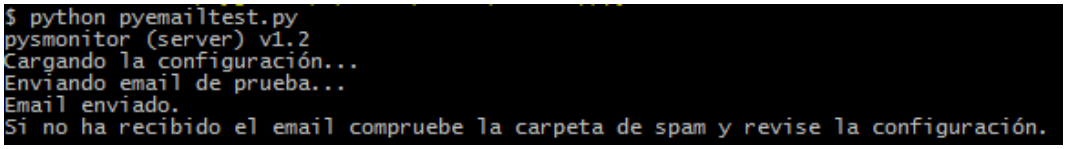


Figura 18. Mensajes mostrados en la consola por la herramienta de monitorización.

4.5 Base de datos

Este apartado está dedicado al diseño de la base de datos. En él se explica la estructura de la base de datos diseñada en este trabajo y los datos que alberga.

Como se discutió en el capítulo 2, se ha decidido utilizar el sistema gestor de bases de datos SQLite [6] debido a que Python [5] trae incorporada una biblioteca para realizar consultas en este tipo de base de datos.

4.5.1 Estructura de la base de datos

La base de datos puede tener cualquier nombre y estar almacenada en cualquier directorio. Por defecto, la base de datos se encuentra en el directorio principal de la aplicación, donde se encuentran las herramientas, y tiene el nombre *db.sqlite*. Tanto el nombre como la ubicación pueden ser modificados en el fichero de configuración *config.cfg*.

La base de datos de este TFM es muy sencilla y contiene únicamente una tabla llamada *"results"*. En esta tabla se almacenarán los resultados obtenidos tras la ejecución de un comando en un servidor remoto. Los campos que contiene esta tabla se pueden ver en la **tabla 1**.

A diferencia de lo que ocurre en otros sistemas gestores de bases de datos, en SQLite no es necesario indicar la longitud máxima de los campos, por ello no se han incluido en la tabla.

La clave primaria es el campo *id*. Este campo no puede ser nulo y tiene que ser único en toda la tabla. Este campo se utilizará para realizar las consultas en la herramienta de visualización cuando se quiera mostrar información detallada sobre el estado de una máquina.

results			
Campo	**Tipo**	**Nulo**	**Extra**
id	INTEGER	No	AUTOINCREMENT
timestamp1	REAL	No	
timestamp2	REAL	No	
host	TEXT	No	
task	TEXT	No	
result	INTEGER	No	
result_summary	TEXT	Sí	
cmd	TEXT	No	
cmd_result	BLOB	Sí	
cmd_exit	INTEGER	No	

Tabla 1. Campos de la tabla *results*.

Además de los campos de la tabla *results*, en la **tabla 1** se puede ver el tipo de dato que almacena el campo, si el campo puede ser nulo o debe contener un valor obligatoriamente y los parámetros especiales que tiene el campo. Si alguno de los campos obligatorios fuera nulo o el valor fuera inválido no se podría procesar adecuadamente la base de datos.

A continuación de detalla el contenido que almacena cada campo:

- *id*: este campo es la clave primaria de la tabla y no puede ser nulo. Se utiliza para referirse a cada entrada de la tabla de forma inequívoca.

Además, es utilizado por la herramienta de visualización para acceder a la información detallada de un informe.

- *timestamp1*: marca de tiempo UNIX que indica la fecha y hora en la cual el comando empezó a ejecutarse en la máquina remota.
- *timestamp2*: marca de tiempo UNIX que indica la fecha y hora en la cual el comando terminó de ejecutarse en la máquina remota. Este campo se utiliza en conjunto con el campo timestamp1 para calcular el tiempo que tardó en ejecutarse el comando.
- *host*: nombre de la máquina en la cual se ejecutó el comando. Debe ser uno de los hosts especificados en el fichero de configuración *hosts.cfg*.
- *task*: nombre de la tarea que se ejecutó en la máquina *host*. Debe ser una de las tareas especificadas en el fichero de configuración *tasks.cfg*.
- *result*: número que indica el resultado de la ejecución del comando. Este número es generado por la herramienta de monitorización después de procesar la respuesta del comando y solo tiene significado en el contexto de la aplicación desarrollada. En la **figura 10** se pueden ver los valores que puede tomar este campo, aunque para obtener información detallada se recomienda consultar el fichero *const.py*.
- *result_summary*: cadena de texto que indica el resumen de la ejecución del comando. Este texto es generado por la herramienta de monitorización después de procesar la respuesta del comando. Este campo es mostrado por la herramienta de visualización y es de gran utilidad, pues contiene la información más relevante de la salida del comando, de esta forma el administrador puede encontrar la información más fácilmente.
- *cmd*: cadena de texto que contiene el comando que fue ejecutado en la máquina remota.
- *cmd_result*: cadena de texto que contiene la salida completa del comando ejecutado en la máquina remota. A partir de este campo se obtiene el resumen que se almacena en el campo *result_summary*.

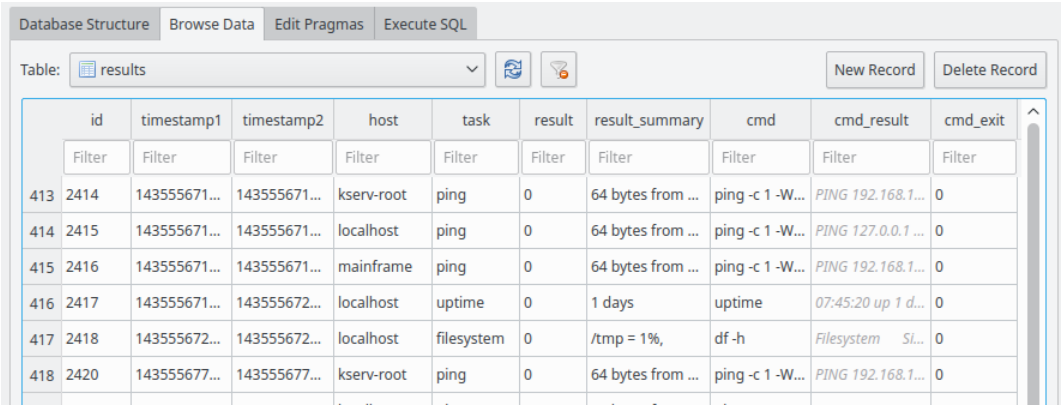
- *cmd_exit*: número que indica el código de salida devuelto por el comando ejecutado en la máquina remota. Este código en conjunto con el campo *cmd_result* nos permite conocer si la ejecución del comando tuvo éxito o se produjo algún error.

Todos los campos de la tabla explicados anteriormente se pueden ver en código de generación de la tabla en la **figura 19**. Este código permite generar la tabla results utilizada en la aplicación.

```
CREATE TABLE `results` (
	`id`			INTEGER PRIMARY KEY AUTOINCREMENT,
	`timestamp1`		REAL NOT NULL,
	`timestamp2`		REAL NOT NULL,
	`host`			TEXT NOT NULL,
	`task`			TEXT NOT NULL,
	`result`		INTEGER NOT NULL,
	`result_summary`	TEXT,
	`cmd`			TEXT NOT NULL,
	`cmd_result`		BLOB,
	`cmd_exit`		INTEGER NOT NULL
);
```

Figura 19. Código para generar la tabla *results*.

En la **figura 20** se puede ver una captura de pantalla de la base de datos con contenido generado durante las pruebas realizadas.



	id	timestamp1	timestamp2	host	task	result	result_summary	cmd	cmd_result	cmd_exit
	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
413	2414	143555671...	143555671...	kserv-root	ping	0	64 bytes from ...	ping -c 1 -W...	PING 192.168.1...	0
414	2415	143555671...	143555671...	localhost	ping	0	64 bytes from ...	ping -c 1 -W...	PING 127.0.0.1 ...	0
415	2416	143555671...	143555671...	mainframe	ping	0	64 bytes from ...	ping -c 1 -W...	PING 192.168.1...	0
416	2417	143555671...	143555672...	localhost	uptime	0	1 days	uptime	07:45:20 up 1 d...	0
417	2418	143555672...	143555672...	localhost	filesystem	0	/tmp = 1%,	df -h	Filesystem Si...	0
418	2420	143555677...	143555677...	kserv-root	ping	0	64 bytes from ...	ping -c 1 -W...	PING 192.168.1...	0

Figura 20. Captura de pantalla de la base de datos.

4.6 Organización del código fuente y de la configuración

En este apartado tiene como objetivo servir para localizar fácilmente dónde se encuentran los diferentes *scripts* y configuraciones de la aplicación dentro del árbol de directorios.

A continuación se muestra una lista de todos los ficheros que forman parte de la aplicación y una breve descripción de la función que realiza cada uno de ellos:

- *pysmonitor.py*: la herramienta de monitorización. Es el *script* más importante de la aplicación y se encarga de establecer las conexiones necesarias y ejecutar los comandos en las máquinas remotas.
- *pyserv.py*: la herramienta de visualización. Contiene el código que crea el servidor *web* y genera la representación *web*.
- *pypassword.py*: la herramienta de cifrado de contraseñas. Un *script* auxiliar que permite cifrar las contraseñas que son guardadas en los ficheros de configuración con AES [39] para mejorar la seguridad. Este *script* hace uso de la biblioteca aes.py.
- *pyemailtest.py*: la herramienta para probar las notificaciones. Este *script* auxiliar es de utilidad para comprobar que la configuración del correo electrónico es válida y funciona correctamente.
- *funct.py*: *script* que contiene las funciones utilizadas por la aplicación.
- *const.py*: *script* que contiene las constantes utilizadas por la aplicación.
- *execssh.py*: clase utilizada para establecer los túneles y ejecutar comandos en las máquinas remotas utilizando la herramienta SSH.
- *execlocal.py*: clase utilizada para ejecutar comandos en la máquina anfitriona.

- *aes.py*: biblioteca que implementa el esquema de cifrado por bloques AES [39]. Es utilizado para cifrar las contraseñas que son almacenadas en los ficheros de configuración.
- *db.sqlite*: base de datos de la aplicación. Es donde se almacenan los resultados obtenidos tras la ejecución de los comandos en las máquinas remotas.
- *mods*: directorio que contiene los módulos que pueden ser utilizados por la herramienta de monitorización.
- *mod_filesystem.py*: módulo para monitorizar la ocupación de los diferentes puntos de montaje de una máquina.
- *mod_numrang.py*: módulo que permite definir varios niveles de alerta en función del número devuelto en la salida de un comando.
- *mod_ping.py*: módulo que permite comprobar la conectividad de una máquina y el estado de la red.
- *mod_regex.py*: módulo que comprueba la existencia de una expresión regular en la salida de un comando y devuelve el código de alerta especificado por el administrador.
- *mod_uptime.py*: módulo que permite definir varios niveles de alerta en función del número de días que la máquina lleva encendida.
- *config*: directorio que contiene los ficheros de configuración.
- *config.cfg*: configuración general de la aplicación. Algunos de los parámetros son: la ubicación de la base de datos, la configuración de las notificaciones, etc.
- *hosts.cfg*: configuración de las máquinas que van a ser monitorizadas. Para cada máquina se incluye su dirección IP y los credenciales con los que se accede a la máquina.
- *groups.cfg*: configuración de los grupos de máquinas y los túneles. Permite agrupar los *hosts* en grupos y establecer un túnel para cada grupo de *host*

en caso de ser necesario. Esto facilita la ejecución de un mismo comando en varias máquinas a la vez.

- *tasks.cfg*: configuración de las tareas. Cada tarea define el módulo (el comando que se ejecuta), el intervalo de tiempo entre ejecuciones y los *hosts* o grupos en los que se ejecuta el comando.

CAPÍTULO 5. MANUALES

En este capítulo se incluyen los manuales de instalación y de administrador. El manual de instalación incluyen las instrucciones para la instalación de todos los programas y bibliotecas necesarias para la correcta ejecución del programa tanto en entornos GNU/Linux (Ubuntu) como en Windows (Cygwin). El manual de administrador incluye instrucciones para configurar las herramientas y las principales funcionalidades que ofrece cada una de ellas.

5.1 Manual de instalación

La herramienta desarrollada en este TFM tiene muy pocas dependencias externas. Solo es necesario tener instalado un intérprete de Python 2 [5] y el cliente de SSH OpenSSH [10]. Estas aplicaciones se encuentran disponibles en la mayoría de los sistemas operativos lo que hace que esta herramienta sea muy portable.

A continuación se detallan los pasos a seguir para instalar la herramienta en un entorno GNU/Linux (Ubuntu) [13] y en un entorno Windows con Cygwin [4].

También es posible instalar la herramienta en otros entornos UNIX diferentes instalando los paquetes que se mencionan en la instalación en GNU/Linux.

5.1.1 Instalación en GNU/Linux (Ubuntu)

Los siguientes pasos son para una instalación en Ubuntu 14.04 [13].

Instalamos los paquetes *openssh-client* y *python2.7* ejecutando los siguientes comandos.

```
apt-get update
apt-get install openssh-client python2.7
```

Nota: Es necesario tener permisos de superusuario para instalar nuevos paquetes. También es posible utilizar la herramienta *sudo* para obtener permisos de superusuario de forma temporal.

Una vez terminada la instalación nos situamos en la ruta donde se encuentra la aplicación, entramos en el directorio que contiene los *scripts* y ejecutamos la herramienta de monitorización. A continuación se muestran los comandos necesarios:

```
cd pysmonitor
python pysmonitor.py
```

Si no se ha producido ningún error, la consola debería mostrar el resultado que se puede ver en la **figura 21**. El mensaje de error referido a que no se encuentra ningún *host* se debe a que la herramienta todavía no está correctamente configurada.

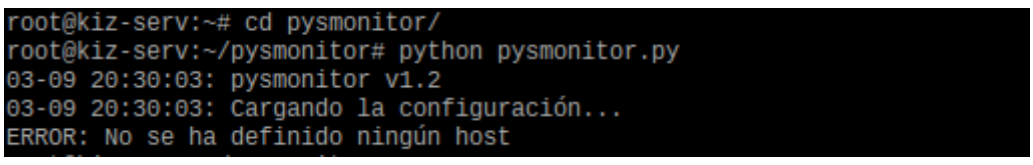


Figura 21. Resultado de la ejecución de la herramienta de monitorización.

5.1.2 Instalación en Windows (Cygwin)

Los siguientes pasos son para una instalación en Windows 7 [15]. Debido a que la herramienta utiliza algunas características que solo están disponibles en sistemas UNIX es necesario utilizar Cygwin [4] para poder ejecutar la herramienta.

El primer paso es descargar la última versión de Cygwin de la página *web* oficial: https://cygwin.com/install.html

Una vez en la página elegimos la versión de 32 bits o de 64 bits de acuerdo a las características de nuestro equipo. Ver la **figura 22**.

Figura 22. Página de descarga de Cygwin.

Una vez descargado el instalador, lo ejecutamos y seguimos el asistente. En las siguientes **figuras** y en los pies de imagen correspondientes se pueden ver las instrucciones a seguir.

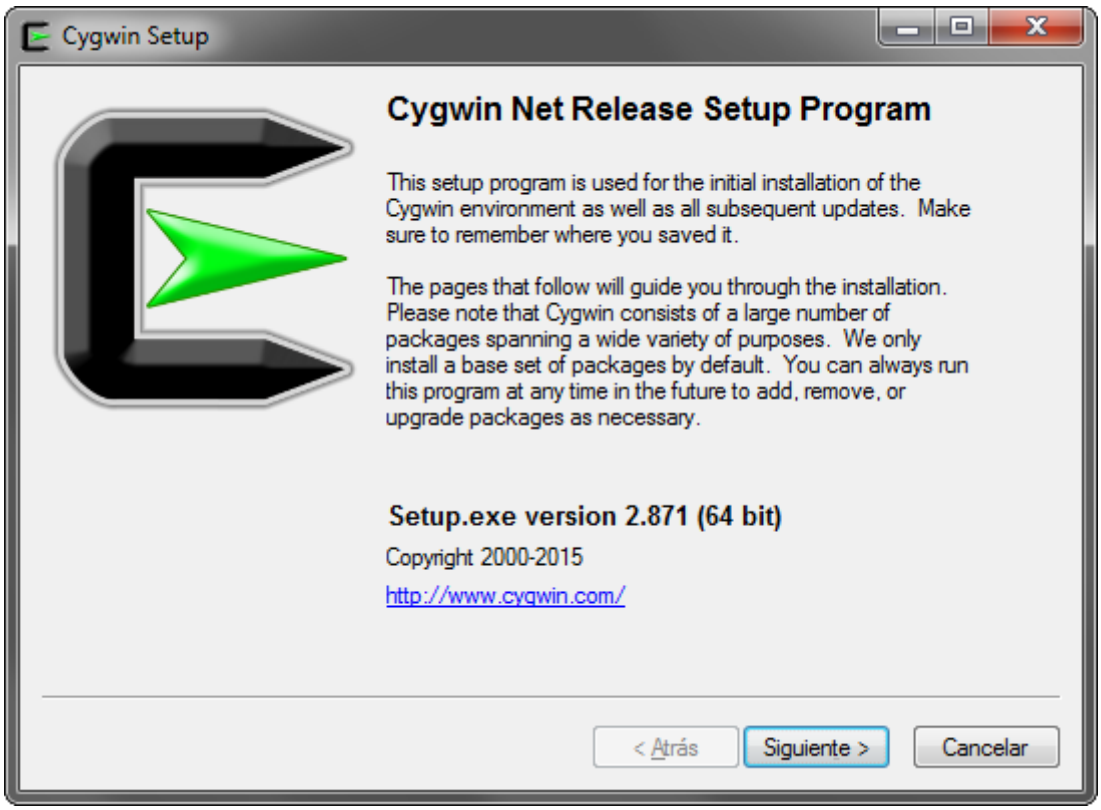


Figura 23. Instalación Paso 1. Notas de instalación.

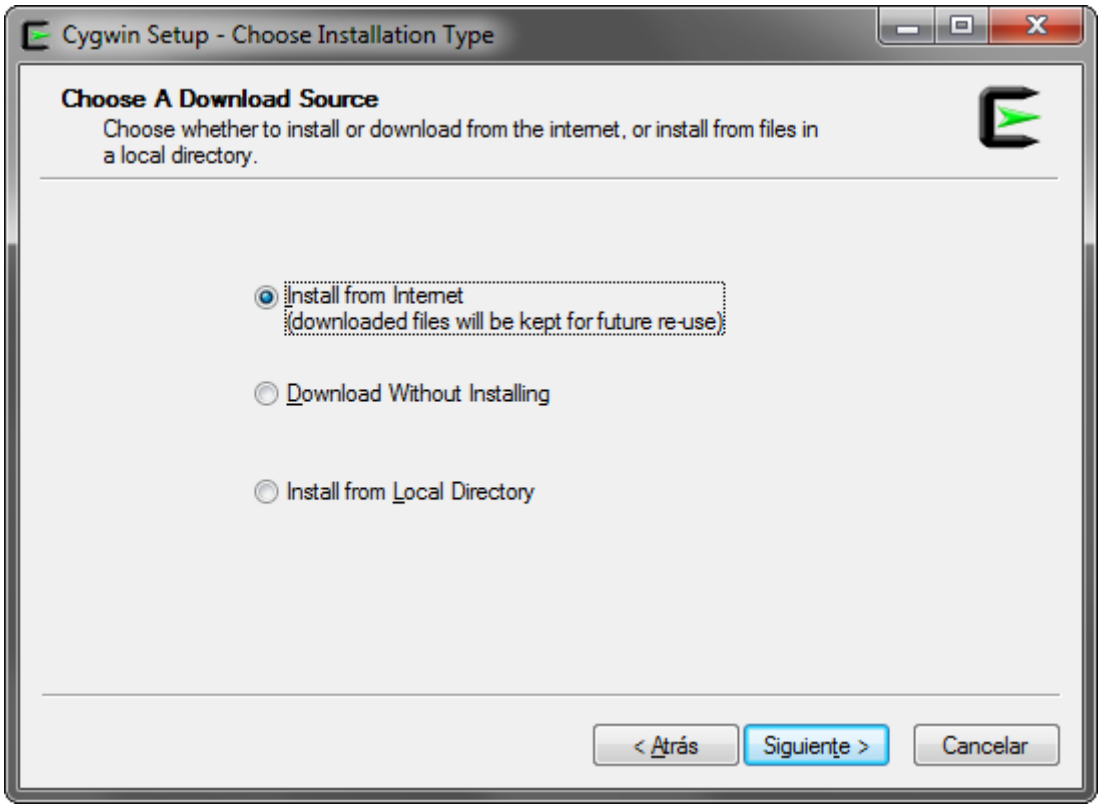


Figura 24. Instalación Paso 2. Descargar los ficheros desde Internet.

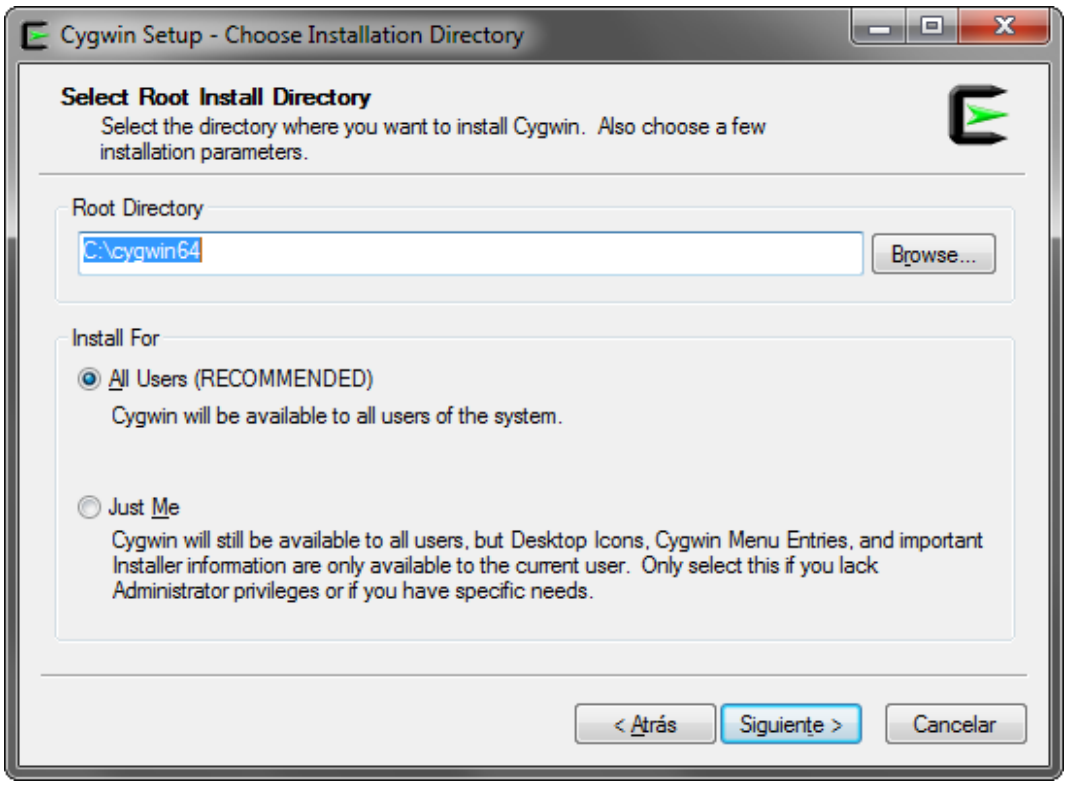


Figura 25. Instalación Paso 3. Directorio de instalación.

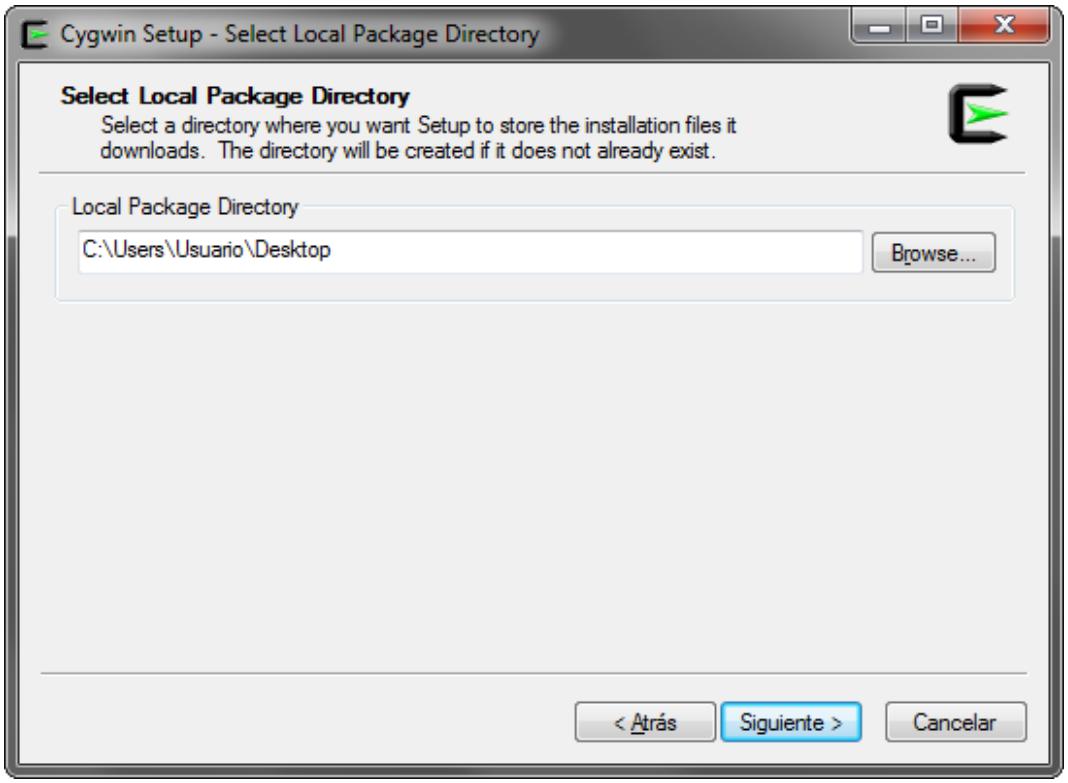


Figura 26. Instalación Paso 4. Directorio temporal para almacenar las descargas.

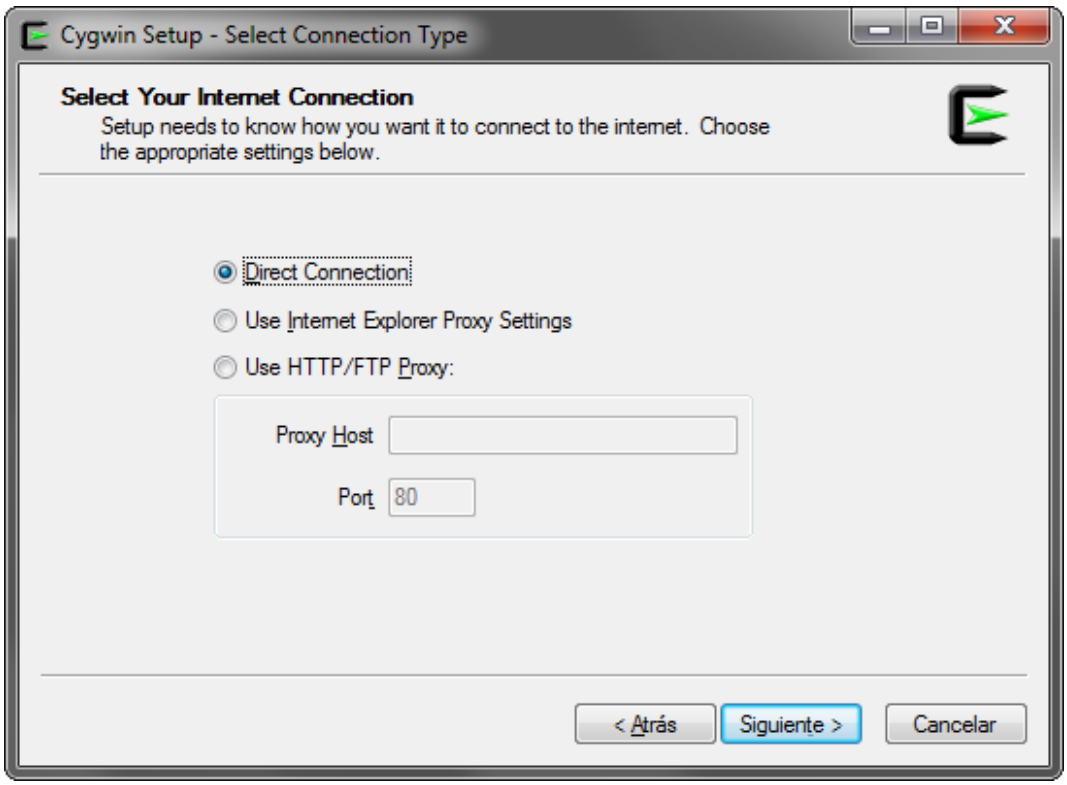


Figura 27. Instalación Paso 5. Descargar con conexión directa a Internet.

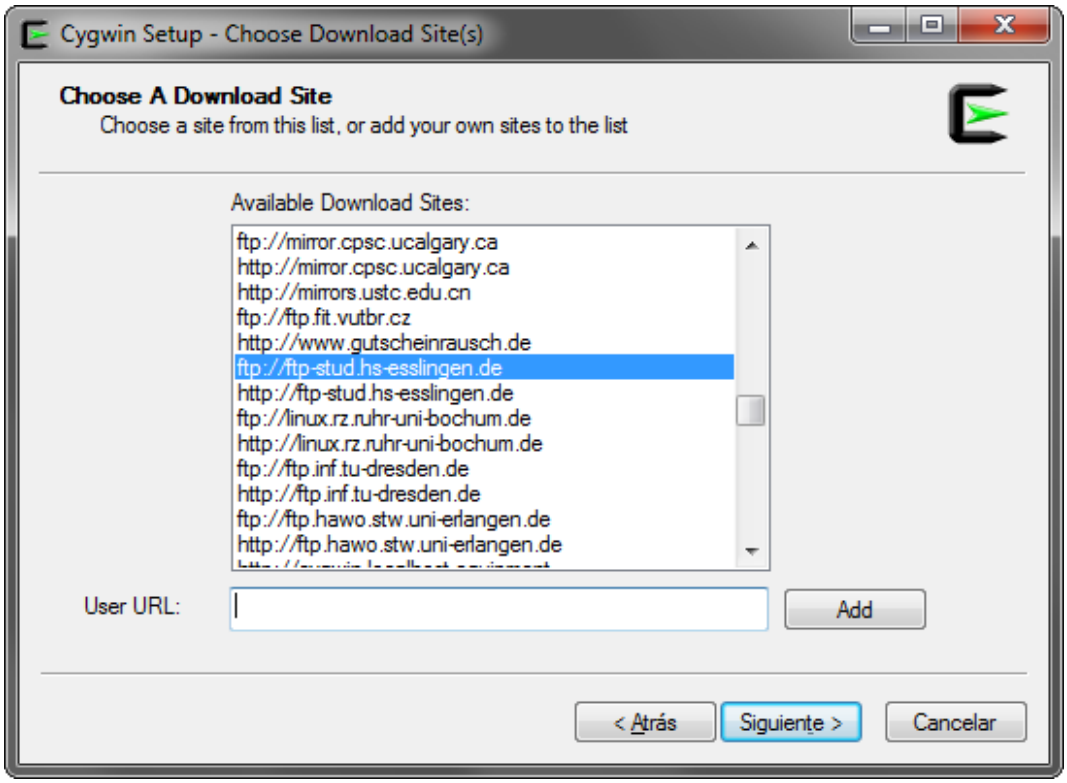


Figura 28. Instalación Paso 6. Servidor de descarga.

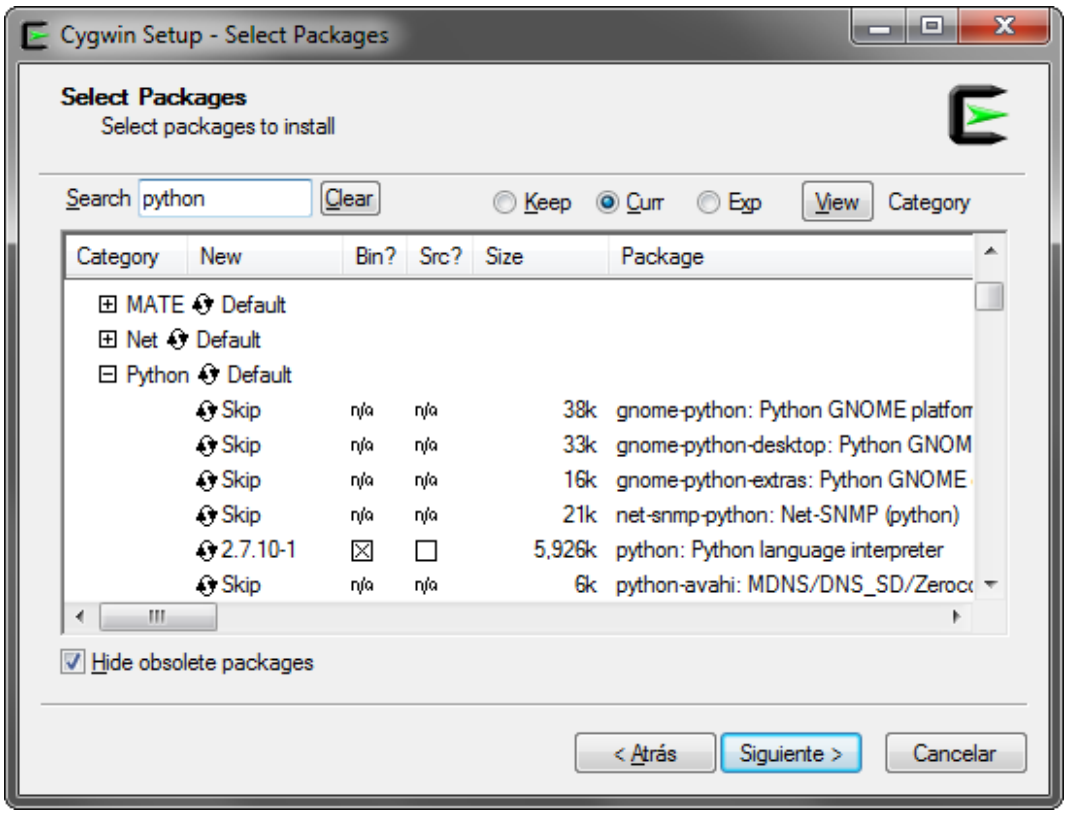


Figura 29. Instalación Paso 7. Seleccionamos el paquete Python.

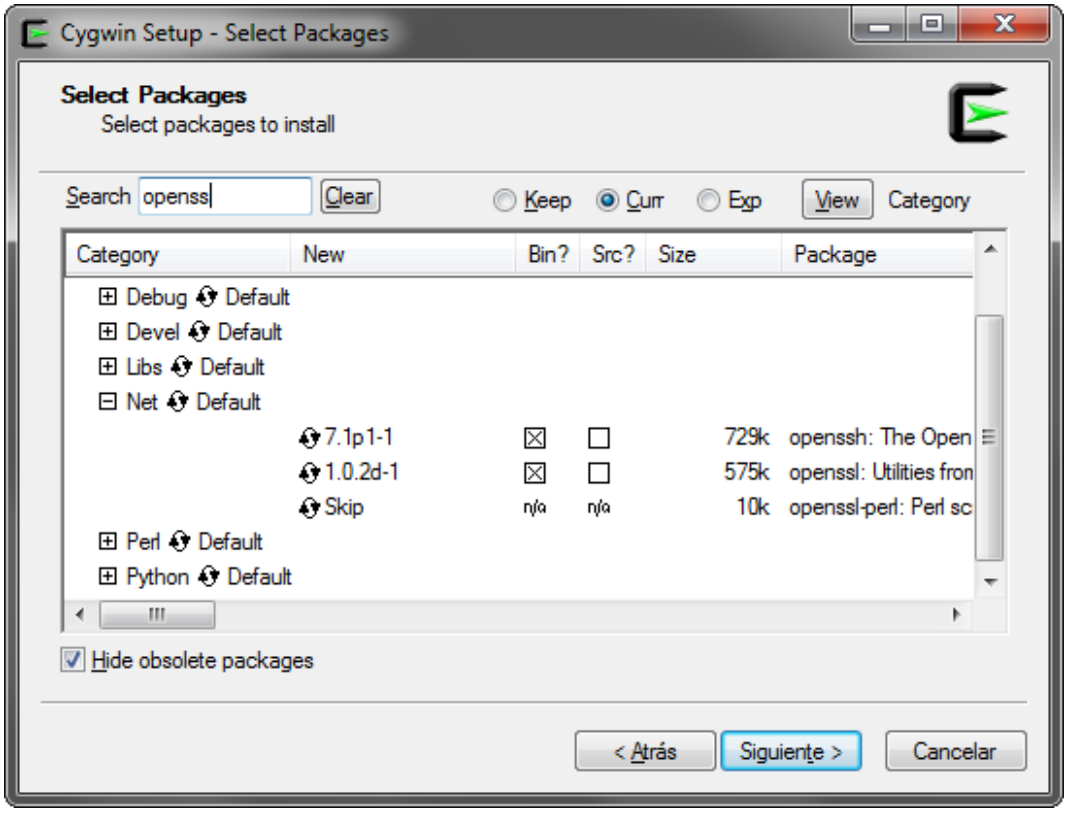


Figura 30. Instalación Paso 8. Seleccionamos el paquete OpenSSH.

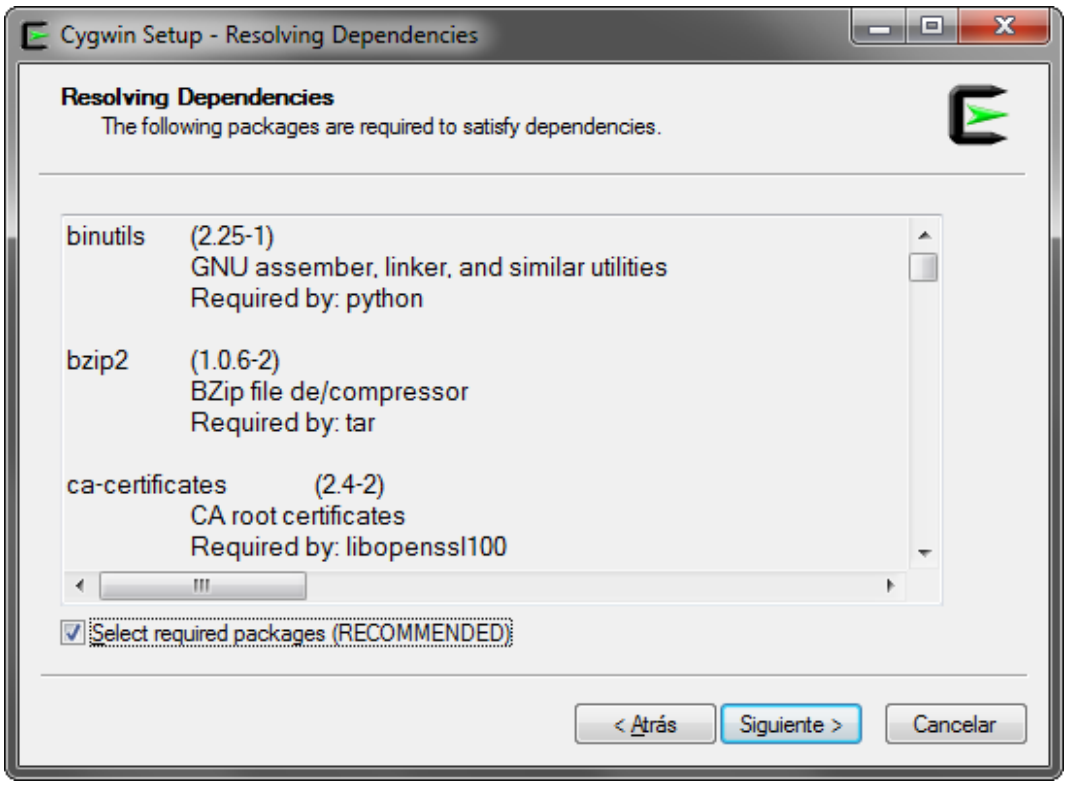


Figura 31. Instalación Paso 9. Confirmamos la descarga de los paquetes.

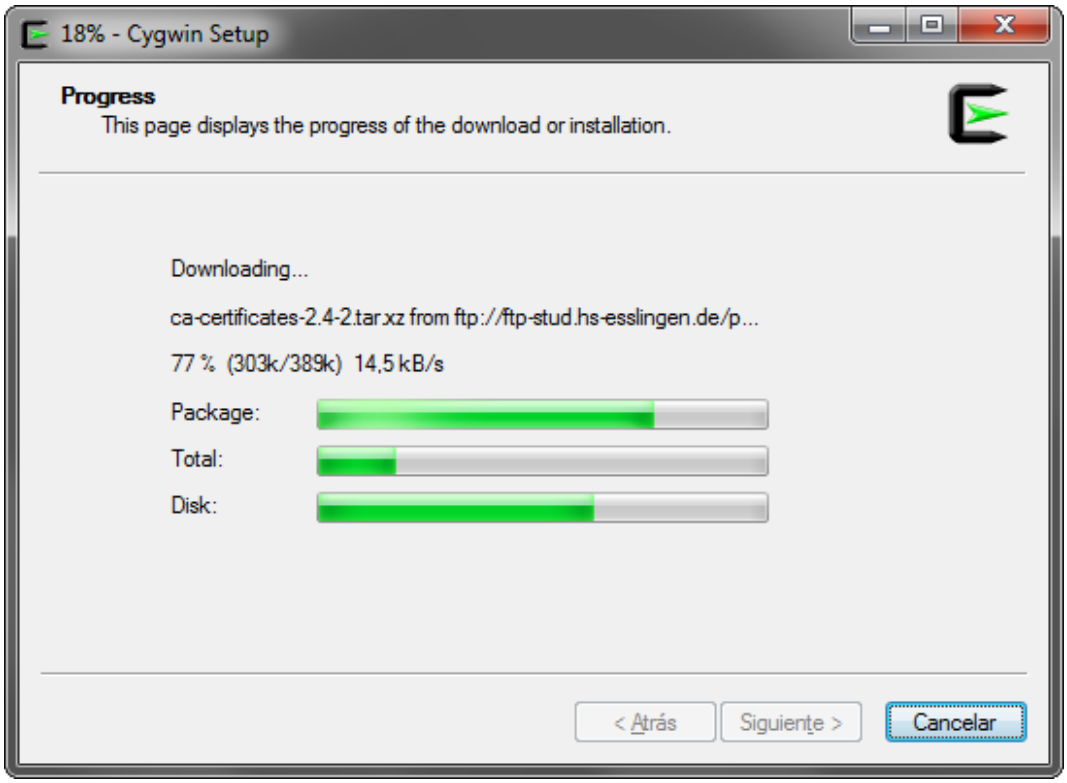


Figura 32. Instalación Paso 10. Esperamos a que termine la descarga.

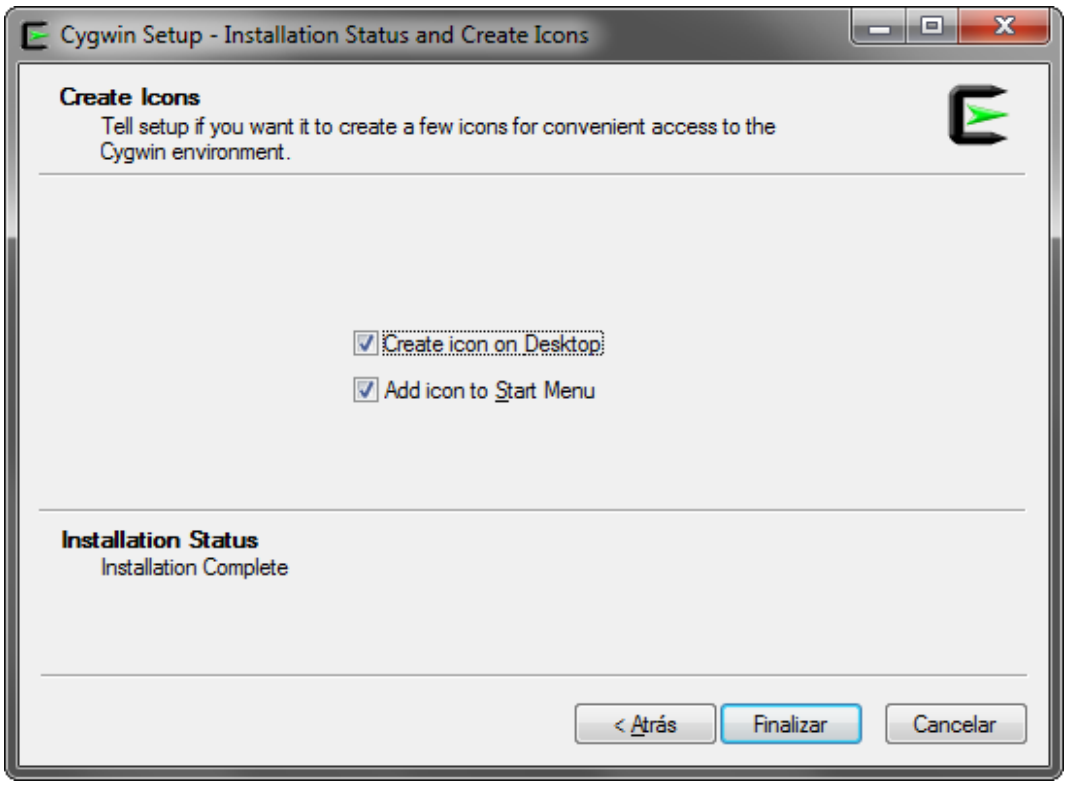


Figura 33. Instalación Paso 11. Terminamos la instalación.

Una vez instalado Cygwin, ejecutamos la terminal de Cygwin desde el menú de inicio.

Con el comando *cd* nos desplazamos hasta el directorio donde está la aplicación desarrollada en este proyecto.

Entramos en el directorio que contiene los *scripts* y ejecutamos la herramienta de monitorización. A continuación se muestran los comandos necesarios:

```
cd pysmonitor
python pysmonitor.py
```

Si no se ha producido ningún error, la consola debería mostrar el resultado que se puede ver en la **figura 34**. El mensaje mostrado es normal y se debe a que la herramienta todavía no está correctamente configurada.

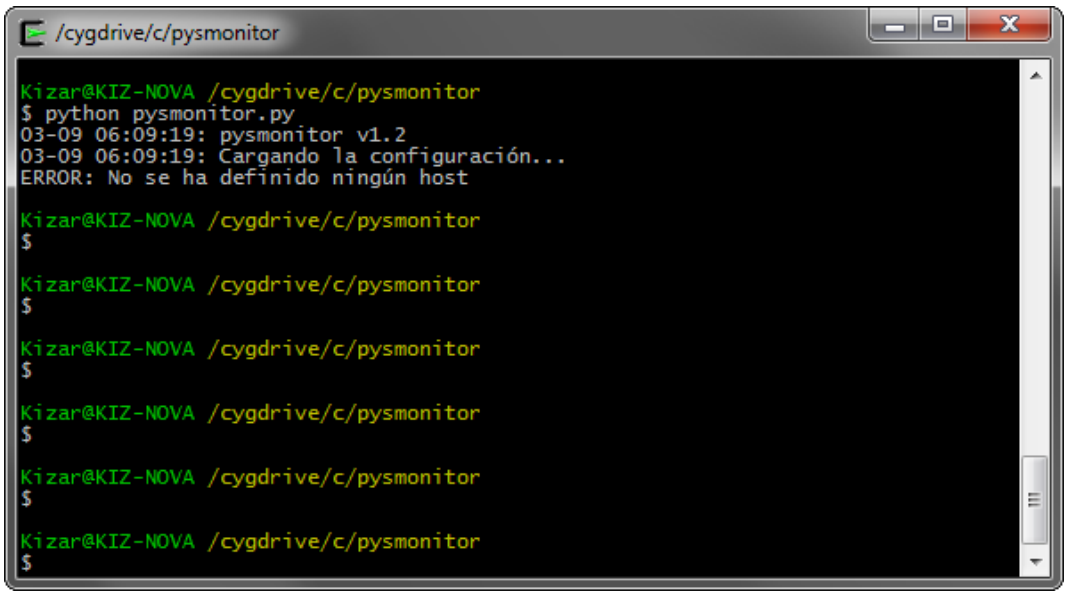


Figura 34. Resultado de la ejecución de la herramienta de monitorización.

5.2 Manual de administrador

Este manual está destinado a los usuarios finales del sistema de monitorización se servidores y tiene como objetivo exponer las opciones de configuración y las posibilidades que ofrece la herramienta.

La información incluida en este manual permite aprender a configurar la aplicación y empezar a utilizar las herramientas que forman parte de ella.

Antes de pasar a explicar el funcionamiento de la aplicación, es necesario introducir algunos términos a los que se hace referencia en este manual. El glosario es el siguiente:

- *host o servidor:* máquina que se quiere monitorizar y sobre la que se ejecutarán los comandos.
- *pasarela o túnel:* máquina que sirve de nodo intermedio cuando el *host* se encuentra ubicado dentro de una red privada sin conexión con el exterior.
- *grupo o grupo de hosts:* conjunto de máquinas que tienen algo en común que permite agruparlas como una unidad.

- *módulo:* conjunto de *scripts* que se utilizan para definir los comandos que se tienen que ejecutar en las máquinas remotas y el procesado que hay que realizar sobre los resultados del comando para determinar si la ejecución ha sido exitosa o no.
- *tarea:* entidad que relaciona un módulo (contiene el comando) y las máquinas en las que se ejecutará. Además, define el intervalo de tiempo que debe esperar la herramienta entre dos ejecuciones consecutivas del módulo.
- *base de datos:* contiene todos los resultados obtenidos tras la ejecución de los comandos y su procesado posterior.
- *configuración o ficheros de configuración:* contienen todos los parámetros de la aplicación que pueden ser modificados por el administrador. Son ficheros de texto plano que, entre otras cosas, permiten definir los *hosts*, los grupos y las tareas.
- *administrador:* es la persona encargada de supervisar el estado de las máquinas ayudándose de las diferentes herramientas desarrolladas en este trabajo.

Debido a que gran parte de la información aquí contenida ya se ha explicado en capítulos anteriores con mayor o menor detalle, se ha evitado duplicar la información y en algunas partes se hace referencia a otra sección de este documento donde es posible ampliar la información.

5.2.1 Configuración de la aplicación

Después de realizar la instalación, véase la sección 5.1, es necesario configurar la aplicación para poder utilizar las herramientas que la forman. En este primer apartado se explican los ficheros de configuración y los parámetros más importantes de cada uno de ellos.

Existen cuatro ficheros de configuración que se encuentran dentro del directorio *config*. A continuación se explica cada uno de ellos de forma resumida y más adelante en detalle.

- *config.cfg*: configuración general de la aplicación. Algunos de los parámetros son: la ubicación de la base de datos, la configuración de las notificaciones, etc.
- *hosts.cfg*: configuración de las máquinas que van a ser monitorizadas. Para cada máquina se incluye su dirección IP y los credenciales con los que se accede a la máquina.
- *groups.cfg*: configuración de los grupos de máquinas y los túneles. Permite agrupar los *hosts* en grupos y establecer un túnel para cada grupo de *host* en caso de ser necesario. Esto facilita la ejecución de un mismo comando en varias máquinas a la vez.
- *tasks.cfg*: configuración de las tareas. Cada tarea define el módulo (el comando que se ejecuta), el intervalo de tiempo entre ejecuciones y los *hosts* o grupos en los que se ejecuta el comando.

El primer fichero que hay que configurar es *config.cfg*. En este fichero se encuentra la configuración general de las herramientas. En la **figura 35** se pueden ver todos los parámetros soportados con su descripción. Además, se puede ver en valor de los parámetros en una configuración válida a modo de ejemplo.

```
[global]
# BASE DE DATOS
# ruta de la base de datos
db_path           = ./db.sqlite

# SERVIDOR
# puerto en el que escucha el servidor
server_port       = 8000

# intervalo de actualización de la web en segundos, desactivado=0
update_interval   = 5

# NOTIFICACIONES EMAIL
# para probar que la configuración de notificaciones por email es
correcta puede utilizar la herramienta pyemailtest.py
# el envío de notificaciones por email puede estar activado=true o
```

```
# desactivado=false
email_alerts      = true

# nivel de alerta a partir del cual se envían notificaciones por
email
# el nivel de alerta puede ser warning=1, error=2 y critical=3
# un nivel de alerta warning envía emails para todos los niveles
de error
alert_level       = 3

# configuración de la dirección de correo que envía los emails
sender_email      = emisor@gmail.com
sender_passwd     = siow69bSYsFoSMqtdWXq
sender_smtp       = smtp.mail.com
sender_smtp_port  = 587

# dirección de correo que recibe las notificaciones
reciv_email       = receptor@gmail.com

# asunto del correo electrónico
email_subject     = Informe pysmonitor
```

Figura 35. Configuración de ejemplo del fichero *config.cfg*.

El siguiente fichero que hay que configurar es *hosts.cfg*. En este fichero se encuentra la configuración de las máquinas que van a ser monitorizadas. En la **figura 36** se pueden ver todos los parámetros soportados con su descripción. Además, se puede ver en valor de los parámetros en una configuración válida a modo de ejemplo.

```
[hosts]
# un host en cada línea con el siguiente formato
# hostname  = ip,usuario[,puerto,contraseña]
# el puerto y la contraseña son opcionales
# si la contraseña está definida debe estar cifrada utilizando
# la herramienta pypassword.py

# localhost acceso con contraseña
localhost   = 127.0.0.1, user1, 22, LH6A7w==

# kserv-root acceso con clave pública/privada
kserv-root  = 192.168.1.102, root

pruebas     = 192.168.1.110, user1, 22, eSnLo5F7

# la conexión al servidor mainframe se realiza a través del túnel
t_tunnel, es necesario tener definidos los dos hosts para
utilizarlos en groups.cfg
mainframe   = 192.168.1.105, user2, 22, ST6A7s==

[tunnels]
# un túnel en cada línea con el siguiente formato
# hostname  = ip,usuario[,puerto,contraseña]
```

```
# el puerto y la contraseña son opcionales
# si la contraseña está definida debe estar cifrada utilizando
# la herramienta pypassword.py

t_tunnel    = 192.168.1.111, user3, 22, LH6A7w==
```

Figura 36. Configuración de ejemplo del fichero *hosts.cfg*.

Como se puede ver en los comentarios de la figura anterior, el fichero cuenta con dos secciones con el mismo formato.

En la parte superior, en la sección *hosts*, se deben indicar todas las máquinas que van a ser monitorizadas. Para que la configuración sea válida es necesario que exista al menos un *host* definido.

En la parte inferior, en la sección *tunnels*, se deben indicar todas las máquinas que son utilizadas para establecer los túneles con las máquinas monitorizadas. Esta sección es opcional y no es necesario especificar ningún túnel para poder utilizar las herramientas.

Existen dos formas de indicar la autenticación en las máquinas remotas. La primera y recomendada es utilizar un par de claves pública/privada. Si el usuario decide utilizar esta forma tiene que configurar manualmente la herramienta OpenSSH [10] para utilizar este método. Las herramientas incluidas en esta aplicación leerán la configuración se OpenSSH. En este caso solo es necesario indicar la dirección IP y el usuario en la configuración.

La segunda forma es definiendo la contraseña dentro del fichero de configuración. Por motivos de seguridad esta contraseña está cifrada utilizando AES [39]. Para cifrar las contraseñas es necesario utilizar la herramienta de cifrado de contraseñas (*pypasswd.py*) incluida en esta aplicación. Esta herramienta solicita al usuario la clave maestra utilizada para cifrar todas las contraseñas y la contraseña de SSH en texto plano. Devuelve como resultado la contraseña cifrada que debe incluirse en el fichero *hosts.cfg*. Para más información ver la sección 5.2.1.

El tercer fichero que hay que configurar es *groups.cfg*. En este fichero se encuentra la configuración de los grupos de máquinas y los túneles que es necesario utilizar para conectar a las diferentes máquinas. En la **figura 37** se pueden ver todos los parámetros soportados con su descripción. Además, se puede ver en valor de los parámetros en una configuración válida a modo de ejemplo.

```
# cada grupo tiene la siguiente estructura
# [nombre_grupo]
# tunnel    = (opcional) túnel que se aplica a todos los hosts del
grupo
# hosts     = lista de hosts separados por comas

[casa]
# conexión directa con los servidores del grupo casa
hosts       = kserv-root, localhost

[telefonica]
# conexión al servidor mainframe a través del túnel t_tunnel, es
equivalente ejecutar una tarea en el grupo telefonica o en el host
mainframe
tunnel      = t_tunnel
hosts       = mainframe
```

Figura 37. Configuración de ejemplo del fichero *groups.cfg*.

Como se puede ver en los comentarios de la figura, el fichero debe incluir una sección para cada grupo de máquinas. Dentro de cada grupo es necesario especificar como mínimo un *host* y de forma opcional un túnel, que será utilizado para establecer la conexión con todas las máquinas incluidas en el grupo.

No es necesario configurar ningún grupo para el correcto funcionamiento de la aplicación pero es muy recomendable utilizar grupos cuando el número de máquinas a monitorizar es muy alto o se quiere ejecutar el mismo comando en muchas máquinas distintas.

El último fichero que hay que configurar es *tasks.cfg*. En este fichero se encuentran las tareas que serán ejecutadas por la herramienta. En la **figura 38** se pueden ver todos los parámetros soportados con su descripción. Además, se puede ver en valor de los parámetros en una configuración válida a modo de ejemplo.

```
# cada tarea tiene la siguiente estructura
# [nombre_de_la_tarea]
# mod       = nombre del módulo
# mod_arg1  = primer parámetro del módulo si es necesario
# mod_arg2  = segundo parámetro del módulo si es necesario
# ...
# interval  = intervalo entre ejecuciones de la tarea en segundos
# groups    = lista de grupos separados por comas
# hosts     = lista de hosts separados por comas

[ping]
mod         = mod_ping
# cada minuto
interval    = 60
groups      =
# se ejecuta en todos los hosts
hosts       = *

[uptime]
mod         = mod_uptime
# cada 24h
interval    = 86400
groups      =
hosts       = localhost, mainframe

[filesystem]
mod         = mod_filesystem
# cada hora
interval    = 3600
groups      =
hosts       = localhost

[zen-nwi]
mod         = mod_regex
mod_arg1    = echo "running"
mod_arg2    = ru[n]+ing
mod_arg3    = critical
# cada 10 min
interval    = 600
groups      = casa
hosts       =

[fragmentation]
mod         = mod_numrang
mod_arg1    = echo "100"
mod_arg2    = ok:0-9999, warning:10000-13999, error:14000-19999,
critical:>20000
# cada 24h
interval    = 86400
groups      =
hosts       = localhost
```

Figura 38. Configuración de ejemplo del fichero *tasks.cfg*.

Como se puede ver en la figura anterior, cada tarea se indica en una sección distinta. Una tarea está formada el conjunto de las máquinas en las que se

ejecutará la tarea y un módulo. Los módulos son un conjunto de clases que se utilizan para definir los comandos que se tienen que ejecutar en las máquinas remotas y el procesado que hay que realizar sobre los resultados del comando para determinar si la ejecución ha sido exitosa o no.

Una funcionalidad importante que permite la configuración de la herramienta es especificar comodines en algunos parámetros. Por ejemplo, en la tarea *ping* de la **figura 38** se ha indicado con un asterisco (*) que el comando debe ejecutarse en todas las máquinas que están definidas en el fichero *hosts.cfg*.

La aplicación incluye por defecto algunos módulos de propósito general. Todos los módulos que utiliza la herramienta se encuentran en el directorio *mods*. A continuación se listan los módulos:

- *mod_filesystem.py*: módulo para monitorizar la ocupación de los diferentes puntos de montaje de una máquina. Utiliza el comando *"df -h"* para obtener la información y permite que el administrador defina los puntos de montaje que quiere monitorizar y cuáles son los diferentes umbrales de alerta.
- *mod_numrang.py*: módulo que permite definir varios niveles de alerta en función del número devuelto en la salida de un comando. Este módulo está pensado para ejecutar un comando que devuelve un resultado numérico y comparar el resultado con unos umbrales.
- *mod_ping.py*: módulo que permite comprobar la conectividad de una máquina y el estado de la red. Utiliza el comando *"ping"* para obtener la información.
- *mod_regex.py*: módulo que comprueba la existencia de una expresión regular en la salida de un comando y devuelve el código de alerta especificado por el administrador.
- *mod_uptime.py*: módulo que permite definir varios niveles de alerta en función del número de días que la máquina lleva encendida. Utiliza el comando *"uptime"* para obtener la información.

En la **figura 38** se pueden ver ejemplos de configuración de cada uno de estos módulos, pero es muy recomendable mirar el código fuente de los módulos para entender en profundidad su funcionamiento y el significado de todos los parámetros.

Adicionalmente, es posible desarrollar nuevos módulos adaptados a las necesidades específicas del administrador. Para instrucciones detalladas se recomienda ver la sección 4.2.3 de este documento.

5.2.2 Herramienta de monitorización

La herramienta de monitorización es el componente principal de esta aplicación. Es la encargada de realizar las tareas de monitorización, incluyendo el establecimiento de los túneles y las conexiones necesarias con las máquinas remotas, la ejecución de los comandos, el procesado de los resultados obtenidos en la ejecución de los comandos y el almacenamiento de la información en la base de datos para su consulta posterior.

El código fuente se encuentra dentro del fichero *pysmonitor.py* y es el *script* que hay que ejecutar si se quiere iniciar la herramienta de monitorización.

Cuando se inicia la herramienta, lo primero que hace es comprobar que la configuración sea correcta. Si algún parámetro tiene un valor incorrecto o falta algún parámetro necesario, la herramienta muestra un mensaje indicando la causa del problema. En este momento la aplicación termina puesto que no puede continuar sin una configuración válida.

En la **figura 39** se pueden ver algunos errores que pueden ocurrir al cargar la configuración.

Si la configuración es correcta se comprueba si alguna de las máquinas (*hosts*) tiene definida la contraseña en el fichero de configuración *hosts.cfg*. Si alguna lo tiene, se pregunta al usuario la clave maestra que se utilizó para cifrar las contraseñas (**figura 40**). Si por el contrario, ninguna de las máquinas tiene definida la autentificación en el fichero *hosts.cfg* se omite este paso.

```
Kizar@KIZ-NOVA /cygdrive/c/pysmonitor
$ python pysmonitor.py
04-09 06:21:12: pysmonitor v1.2
04-09 06:21:12: Cargando la configuración...
ERROR: No se ha definido ningún host

Kizar@KIZ-NOVA /cygdrive/c/pysmonitor
$ python pysmonitor.py
04-09 06:21:19: pysmonitor v1.2
04-09 06:21:19: Cargando la configuración...
04-09 06:21:19: Comprobando la base de datos...
04-09 06:21:19: Iniciando las tareas...
ERROR: La tarea ping no tiene definido ningún host o grupo

Kizar@KIZ-NOVA /cygdrive/c/pysmonitor
$
```

Figura 39. Algunos errores al cargar la configuración.

```
Kizar@KIZ-NOVA /cygdrive/c/pysmonitor
$ python pysmonitor.py
04-09 06:24:38: pysmonitor v1.2
04-09 06:24:38: Cargando la configuración...
04-09 06:24:38: Comprobando la base de datos...
Introduce la contraseña global utilizada para cifrar las contraseñas SSH de todo
s los hosts.
Contraseña:
04-09 06:24:49: Iniciando las tareas...
```

Figura 40. La herramienta pregunta la contraseña maestra.

En este momento la herramienta ejecuta todas las tareas programadas y permanece en este estado hasta que el usuario decide terminar la herramienta. En la **figura 41** se pueden un ejemplo de la salida que produce la herramienta en la consola.

```
04-09 06:29:13: Iniciando las tareas...
04-09 06:29:13: Iniciando tarea zen-nwi@kserv-root
04-09 06:29:13: Iniciando tarea zen-nwi@localhost
04-09 06:29:13: Iniciando tarea uptime@localhost
04-09 06:29:13: Iniciando tarea uptime@mainframe
04-09 06:29:13: Iniciando tarea ping@kserv-root
04-09 06:29:13: Iniciando tarea ping@mainframe
04-09 06:29:13: Iniciando tarea ping@pruebas
04-09 06:29:13: Iniciando tarea ping@localhost
04-09 06:29:13: Iniciando tarea fragmentation@localhost
04-09 06:29:13: Iniciando tarea filesystem@localhost
04-09 06:29:13: Ejecutando fragmentation (mod_numrang) @ localhost
04-09 06:29:13: Ejecutando ping (mod_ping) @ kserv-root
04-09 06:29:14: Ejecutando ping (mod_ping) @ t_tunnel => mainframe
04-09 06:29:14: Ejecutando ping (mod_ping) @ pruebas
04-09 06:29:14: Ejecutando zen-nwi (mod_regex) @ localhost
04-09 06:29:15: Ejecutando uptime (mod_uptime) @ t_tunnel => mainframe
04-09 06:29:15: Ejecutando filesystem (mod_filesystem) @ localhost
04-09 06:29:15: Ejecutando uptime (mod_uptime) @ localhost
04-09 06:29:16: Ejecutando zen-nwi (mod_regex) @ kserv-root
04-09 06:29:16: Ejecutando ping (mod_ping) @ localhost
```

Figura 41. Mensajes mostrados en la consola de la herramienta de monitorización.

En cada línea se indica la fecha y hora, la tarea que se ejecutó (y el módulo) y el servidor remoto en el que se ejecutó. En algunas líneas aparecen dos

máquinas al final, la primera es el túnel que hay configurado y la segunda es la máquina en la que se ejecuta el comando.

5.2.3 Herramienta de visualización

La herramienta de visualización es otro de los componentes principales de la aplicación. Es la encargada de crear un servidor *web* y de obtener la información almacenada en la base de datos y representarla de una forma conveniente para el administrador.

De un modo más general, permite visualizar de forma directa y sencilla el estado de todas las máquinas y servicios que están siendo monitorizados para que el administrador sea capaz de detectar los problemas rápidamente y tomar las medidas necesarias para solucionarlos.

El código fuente se encuentra dentro del fichero *pyserv.py* y es el *script* que hay que ejecutar si se quiere iniciar la herramienta de visualización.

Cuando se inicia la herramienta, lo primero que hace es comprobar que la configuración sea correcta. Si algún parámetro tiene un valor incorrecto o falta algún parámetro necesario, la herramienta muestra un mensaje indicando la causa del problema. En este momento la aplicación termina puesto que no puede continuar sin una configuración válida.

A continuación, crea un servidor *web* y se mantiene a la escucha de peticiones hasta que termina la aplicación. El puerto en el que escucha el servidor se puede definir en el fichero de configuración *config.cfg*.

En la **figura 42** se pueden ver los mensajes mostrados en la consola por la aplicación. Se imprime una línea cada vez que se recibe una petición por parte del navegador.

```
$ python pyserv.py
02-09 08:51:53: pysmonitor (server) v1.2
02-09 08:51:53: Cargando la configuración...
02-09 08:51:53: Comprobando la base de datos...
02-09 08:51:53: Cargando información de las tareas...
02-09 08:51:53: Iniciando el servidor...
02-09 08:51:53: Escuchando en el puerto 8000.
127.0.0.1 - - [02/Sep/2015 08:52:08] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [02/Sep/2015 08:52:13] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [02/Sep/2015 08:52:13] "GET /history?host=mainframe&task=uptime&li
mit=150 HTTP/1.1" 200 -
127.0.0.1 - - [02/Sep/2015 08:52:15] "GET /details?id=2716 HTTP/1.1" 200 -
```

Figura 42. Mensajes mostrados en la consola por la herramienta de visualización.

Si nos encontramos en la misma máquina en la que se está ejecutando la herramienta de visualización, para acceder a la interfaz *web* hay que abrir la URL http://localhost:8000/ con cualquier navegador *web* (8000 es el puerto definido por defecto en la configuración).

En la **figura 43** se puede ver la vista principal de la interfaz.

Host	Task	Last Check	Duration	Status	Status Information	Details
kserv-root	zen-nwi	02-07-2015 01:34:57	2.4s	OK	Se ha encontrado "ru[n]+ing"	Details
kserv-root	ping	02-07-2015 01:35:58	0.0s	OK	64 bytes from 192.168.1.102: icmp_seq=1 ttl=64 time=0.283 ms	Details
mainframe	uptime	02-07-2015 01:34:58	3.59s	OK	17 days	Details
mainframe	ping	02-07-2015 01:36:01	0.17s	OK	64 bytes from 192.168.1.102: icmp_seq=1 ttl=64 time=0.264 ms	Details
pruebas	ping	02-07-2015 01:35:00	10.0s	CRITICAL	No existe conectividad con el host	Details
localhost	zen-nwi	02-07-2015 01:35:05	0.17s	OK	Se ha encontrado "ru[n]+ing"	Details
localhost	uptime	02-07-2015 01:34:59	0.16s	OK	0 days	Details
localhost	ping	02-07-2015 01:36:02	0.0s	OK	64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.018 ms	Details
localhost	fragmentation	02-07-2015 01:34:59	0.16s	OK	100	Details
localhost	filesystem	02-07-2015 01:35:00	0.17s	OK	/tmp = 1%,	Details

(Actualizando cada 5 segundos)

Figura 43. Vista principal de la interfaz *web*.

En esta vista la herramienta muestra, para cada máquina monitorizada, la respuesta de los comandos ejecutados de forma visual y textual. Además, muestra la fecha y hora de la última ejecución del comando y el tiempo que tardó en ejecutarse el comando.

Como se puede ver en la figura, cuando la ejecución del comando ha sido correcta el color es verde, mientras que cuando se ha producido un error el color es distinto, siendo naranja para el nivel *warning* y rojo para *critical*.

Es posible ordenar la tabla por máquinas (*host*) o por tareas (*task*). Al hacer clic en los enlaces de la columna detalles (*details*) se muestra la información detallada del comando ejecutado (**figura 44**). Al hacer clic en los enlaces de la columna última comprobación (*last check*) se muestra el historial de ejecuciones del comando para esa máquina concreta (**figura 45**). La información de esta tabla es refrescada automáticamente cada cierto tiempo, el intervalo de refresco es fijado en el fichero *config.cfg*.

Para cada comando ejecutado es posible visualizar información detallada (toda la información que fue recopilada y almacenada en la base de datos). La representación de esta información se puede ver en la **figura 44**.

La vista detallada, además de contener la información que aparecía en la vista general, contiene: el comando ejecutado con todos los parámetros, el código de salida del comando, la salida textual del comando, la fecha y hora de ejecución y el tiempo que tardó en ejecutarse.

Host	mainframe
Task	uptime
Last Check	02-07-2015 01:34:58
Duration	3.59s
Status	OK
Status Information	17 days
Command	uptime
Command exit code	0
Command result	01:36:25 up 17 days, 12:48, 0 users, load average: 0,14, 0,16, 0,24

Figura 44. Vista detallada de la interfaz *web*.

Por último, la herramienta permite listar todas las ejecuciones anteriores de un mismo comando en una máquina concreta. Esto puede ser de gran utilidad para identificar cuando comenzó a producirse el fallo de un comando en una máquina. La representación de esta información se puede ver en la **figura 45**.

Host	Task	Last Check	Duration	Status	Status Information	Details
mainframe	uptime	02-07-2015 01:34:58	3.59s	OK	17 days	Details
mainframe	uptime	02-07-2015 01:31:56	2.05s	OK	17 days	Details
mainframe	uptime	01-07-2015 22:54:24	3.1s	OK	17 days	Details
mainframe	uptime	01-07-2015 22:53:30	2.25s	CRITICAL	Error de autentificación tunel SSH	Details
mainframe	uptime	01-07-2015 09:41:37	2.42s	CRITICAL	Error de autentificación tunel SSH	Details
mainframe	uptime	01-07-2015 09:40:59	9.7s	OK	16 days	Details
mainframe	uptime	01-07-2015 08:05:31	7.65s	OK	16 days	Details
mainframe	uptime	01-07-2015 06:34:07	2.22s	OK	16 days	Details

Figura 45. Vista del historial de ejecuciones de un comando en la interfaz *web*.

En la figura se puede apreciar que la información mostrada es casi igual que la mostrada en la vista general pero permite ver cuándo comenzó a producirse un problema y la vista detallada de los comandos cuando se produjo el problema.

CAPÍTULO 6. PRUEBAS

En este capítulo se explican las pruebas realizadas con la aplicación tanto en el entorno GNU/Linux como en Windows (Cygwin), así como los resultados obtenidos en cada prueba. La finalidad de este capítulo es mostrar los errores que fueron encontrados en las pruebas y los pasos que fueron llevados a cabo para su resolución.

6.1 Problemas detectados

En esta sección se detallan los problemas más importantes que se detectaron durante la realización de este trabajo y las soluciones implementadas para solucionar estos problemas.

6.1.1 Problemas detectados en GNU/Linux

Debido a que la aplicación fue desarrollada en un entorno GNU/Linux, estos fueron los primeros problemas detectados.

Problema: ejecución secuencial de las tareas programadas

Síntomas: la ejecución de algunas tareas retrasa la ejecución de las siguientes tareas programadas.

Causas: la aplicación lanzaba las tareas de forma secuencial, esperando a la finalización de la tarea anterior antes de lanzar la siguiente. Cuando una tarea, como la realización de un *backup* del disco duro, tardaba mucho tiempo en completarse, impedía que las siguientes tareas se ejecutaran a tiempo.

Soluciones consideradas:

a) Limitar el tiempo que puede tardar una tarea en completarse de forma que la siguiente tarea siempre se ejecute a tiempo. No es posible implementar esta solución puesto que sería imposible ejecutar algunos comandos en las máquinas monitorizadas. Esto es un requisito funcional de la aplicación desarrollada.

Solución final: ejecutar cada tarea en un hilo de ejecución distinto, de esta forma es posible ejecutar varias tareas de forma concurrente. Con esta solución se puede lanzar una nueva tarea aunque la tarea anterior no haya terminado y, por tanto, una tarea no puede bloquear la ejecución de las otras tareas.

Problema: ejecución de muchas tareas a la vez al arrancar la aplicación

Síntomas: cuando se inicia la herramienta de monitorización se observa un consumo elevado de memoria RAM, aumenta mucho el uso de la CPU y aumenta considerablemente el envío de paquetes a la red.

Causas: cuando se inicia la herramienta de monitorización se lanzan rápidamente todas las tareas programadas en cada máquina. Esto puede causar el establecimiento de cientos de conexiones de red al mismo tiempo, produciendo los efectos no deseados ya mencionados.

Soluciones consideradas:

a) Limitar el número de tareas que pueden ser ejecutadas a la vez. No es posible implementar esta solución puesto que podría darse el caso de que todas las tareas ejecutadas tardaran mucho tiempo en terminar

retrasando la ejecución de las siguientes tareas. Es de capital importancia que todas las tareas se ejecuten a tiempo.

Solución final: establecer una pequeña pausa entre la ejecución de dos tareas que siga una distribución gaussiana al ejecutar las tareas por primera vez. Con ello conseguimos que las tareas se ejecuten con una separación temporal de forma que se reduce el número de tareas que son ejecutadas a la vez, puesto que algunas tareas acaban antes de que se terminen de lanzar todas las tareas.

6.1.2 Problemas detectados en Windows (Cygwin)

Después del desarrollo de la aplicación en un entorno GNU/Linux y solucionar satisfactoriamente todos los problemas encontrados, se realizaron las pruebas de la aplicación en Cygwin. Durante la realización de estas pruebas se encontraron nuevos problemas que no existían en GNU/Linux.

Problema: no funciona la herramienta OpenSSH

Síntomas: cuando la herramienta desarrollada lanza la aplicación OpenSSH para establecer una conexión con una máquina remota se produce un fallo que impide establecer la conexión.

Causas: al instalar el entorno Cygwin, por defecto no existe el directorio *home* del usuario. Al lanzar el comando *ssh* desde la terminal se muestra un error en la consola que indica que no se puede leer la configuración porque no existe el directorio home del usuario.

Soluciones consideradas:

a) Incluir en el manual de instalación los pasos necesarios para crear el directorio *home* del usuario después de la instalación de Cygwin. Esta solución no es adecuada porque aunque es sencillo crear el directorio *home* de forma manual, requiere un paso más en la configuración que puede ser automatizado.

b) Al iniciar la herramienta, comprobar si nos encontramos en un entorno Cygwin y, si el directorio *home* del usuario no existe, pedir al usuario que lo cree manualmente. Al igual que la solución anterior, se requiere un paso más en la instalación de la aplicación que puede ser evitado.

Solución final: al iniciar la herramienta, comprobar si nos encontramos en un entorno Cygwin y, si el directorio *home* del usuario no existe, la herramienta lo crea automáticamente. Se ha comprobado que no son necesarios permisos de superusuario para crear este directorio.

Problema: errores al crear los hilos de ejecución

Síntomas: al ejecutar la herramienta con la misma configuración que funcionaba en el entorno GNU/Linux, algunas tareas nunca terminan de ejecutarse en la máquina remota.

Causas: debido a que en Windows no existe la llamada al sistema *fork* utilizada para crear los hilos de ejecución de las diferentes tareas, Cygwin emula esta llamada al sistema creando diferentes procesos [42], pero existen varios problemas que no pueden ser solucionados, produciendo errores en la creación de algunos hilos [42].

Soluciones consideradas:

a) Al iniciar la herramienta, comprobar si nos encontramos en un entorno Cygwin y en caso afirmativo crear lanzar las tareas de forma secuencial en vez de ejecutar cada tarea en un hilo de ejecución. Como se vio en la sección de problemas detectados en GNU/Linux, esta solución no es aceptable puesto que si una tarea tarda mucho tiempo en completarse se retrasa la ejecución de las tareas posteriores.

Solución final: al iniciar la herramienta, comprobar si nos encontramos en un entorno Cygwin y en caso afirmativo lanzar cada tarea en nuevo proceso en vez de en un nuevo hilo de ejecución. Si bien esta solución no es perfecta debido a que aumenta el consumo de memoria RAM, permite cumplir los requisitos

establecidos y obtener las mismas funcionalidades que las conseguidas en el entorno GNU/Linux.

Problema: no funciona la herramienta ping

Síntomas: cuando la herramienta desarrollada ejecuta el comando *ping* para comprobar la conectividad con las máquinas monitorizadas el comando produce un error.

Causas: la herramienta *ping* incluida en Cygwin necesita permisos de Administrador para poder ejecutarse [43].

Soluciones consideradas:

a) Incluir en el manual de administrador los pasos necesarios para ejecutar la terminal de Cygwin con permisos de Administrador. Esta solución no es adecuada porque es posible que el usuario que va a ejecutar la herramienta no tenga permisos de administrador.

Solución final: utilizar el comando *ping* de Windows en lugar del comando *ping* incluido en Cygwin. Si bien esta solución implica pasar al comando *ping* diferentes parámetros dependiendo del entorno de ejecución y también procesar la salida del comando de forma distinta, se han obtenido buenos resultados en todas las pruebas realizadas.

6.2 Pruebas de fiabilidad

Con el objetivo de comprobar el correcto funcionamiento de la aplicación desarrollada, se han realizado una serie de pruebas tratando de abarcar el mayor número posible de situaciones de error.

A continuación se incluye una lista con las pruebas realizadas:

- Se configura incorrectamente un *host* (incluyendo una dirección IP incorrecta, un nombre de usuario incorrecto, un puerto de SSH incorrecto y una contraseña incorrecta).

- Se configura incorrectamente un grupo de *hosts* (incluyendo un grupo sin ningún *host*, el nombre de un *host* que no existe y un túnel que no está definido en la configuración).
- Se configura incorrectamente una tarea (incluyendo un *host* que no existe, un grupo de *hosts* que no existe, un intervalo de tiempo incorrecto, un módulo que no existe y un módulo sin los parámetros necesarios para su correcto funcionamiento).
- Se crea un módulo nuevo sin los métodos necesarios para que pueda ejecutarse correctamente (ver sección 4.2.3.1).
- Se configuran incorrectamente las notificaciones (incluyendo una dirección de correo electrónico que no existe, un servidor SMTP que no existe y unas credenciales del servidor SMTP incorrectas).
- Se configura incorrectamente la clave pública/privada de autenticación de una máquina monitorizada.
- Se configura incorrectamente la contraseña de autenticación de una máquina monitorizada en el fichero de configuración.
- Se configura incorrectamente un túnel en una máquina monitorizada que no lo necesita.
- Se configura incorrectamente una máquina monitorizada no incluyendo el túnel necesario para poder establecer la conexión.
- Se desconecta la conexión a Internet de la máquina en la que se está ejecutando la herramienta de monitorización.
- Se ejecuta un comando que no está instalado en la máquina remota.
- Se ejecuta la herramienta sin una base de datos válida.
- Se configura un número muy alto de tareas y de máquinas a monitorizar.

Después de la realización de estas pruebas y de solucionar todos los errores que se detectaron, la herramienta debería funcionar correctamente en la mayoría de las situaciones que se van a producir en un entorno de trabajo real.

6.3 Pruebas de rendimiento

La aplicación ha sido sometida a unas pruebas de rendimiento para conocer el uso de recursos bajo unas condiciones normales de trabajo. Aunque el uso de pocos recursos no era un requisito fundamental, puesto que la herramienta se ejecutará en una máquina con gran cantidad de recursos, es deseable que la aplicación resultante sea escalable y consuma el menor número de recursos posible.

6.3.1 Condiciones experimentales

Las pruebas se han realizado en un ordenador de sobremesa con las siguientes características:

- Intel Core2Duo E8400 @ 3.2 GHz
- 4 GB RAM
- 512 GB SSD

Se han realizado dos pruebas distintas en las que se ha medido el consumo de memoria y la utilización de CPU por la herramienta desarrollada. Para realizar las mediciones se ha utilizado la herramienta *top* [44].

Una primera prueba se ha realizado ejecutando 100 tareas en una máquina remota sin configurar un túnel. Cada tarea consistía en ejecutar el comando "*uptime*" cada 60 segundos.

La segunda prueba se ha realizado bajo las mismas condiciones pero ejecutando 300 tareas.

Para cada una de las pruebas se ha recopilado información de la herramienta *top* cada segundo durante varias horas y, tras analizar los resultados, se observa

que el consumo de recursos sigue un patrón que se repite cada pocos segundos. Se muestran, pues, los resultados obtenidos durante los primeros 150 segundos, tiempo suficiente para que se pueda apreciar la repetición del patrón.

6.3.2 Resultados de las pruebas

En la **figura 46** se puede ver el uso de CPU de la herramienta en las dos pruebas realizadas. Se muestran los resultados obtenidos durante los primeros 150 segundos.

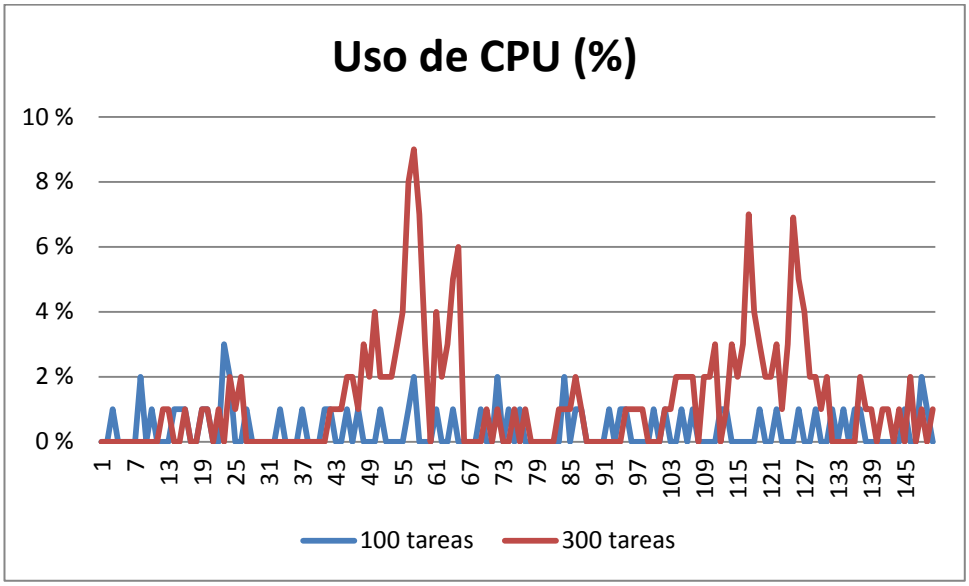


Figura 46. Resultados del uso de CPU.

Se muestra en color azul los resultados de la prueba realizada ejecutando 100 tareas y en color rojo la prueba realizada ejecutando 300 tareas.

Se puede ver que el uso de CPU de la herramienta en ambos casos es moderado siendo ligeramente superior cuando se ejecutan más tareas. En este caso concreto, los picos producidos en la línea roja son debidos a que se están ejecutando varias tareas de forma simultánea.

En la **figura 47** se puede ver el uso de memoria de la herramienta en las dos pruebas realizadas. Al igual que en la gráfica anterior, se muestran los resultados obtenidos durante los primeros 150 segundos.

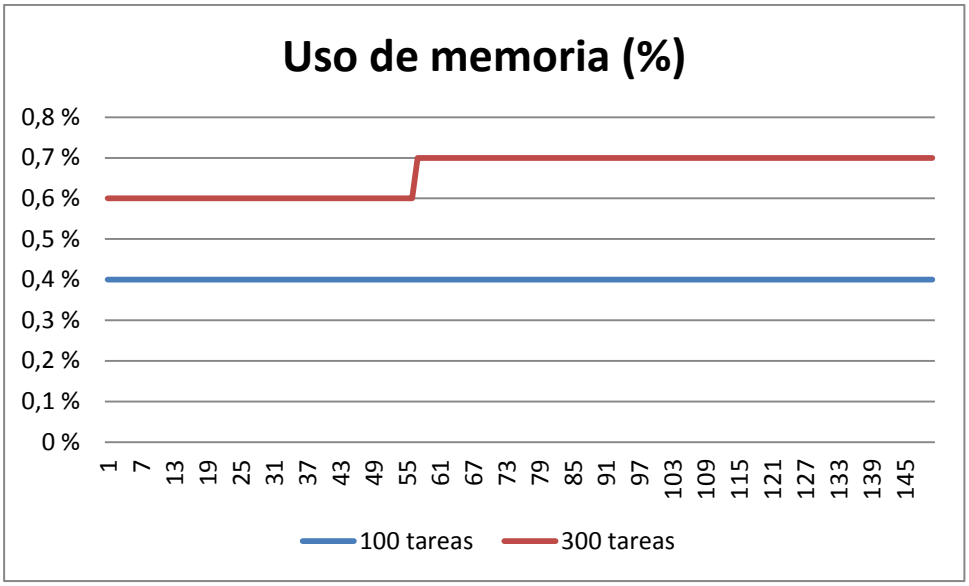


Figura 47. Resultados del uso de memoria.

Se muestra en color azul los resultados de la prueba realizada ejecutando 100 tareas y en color rojo la prueba realizada ejecutando 300 tareas.

Se puede ver que el uso de memoria de la herramienta en ambos casos es bajo siendo ligeramente superior cuando se ejecutan más tareas. En este caso concreto, en la línea roja se ve un pequeño aumento en el uso de memoria que coincide con el aumento del uso de CPU en la **figura 46**. Esto es debido a que se están ejecutando varias tareas de forma simultánea. Una vez pasado el pico de uso de CPU no se libera la memoria inmediatamente debido a que Python se encarga de la gestión de la memoria automáticamente a través de un recolector de basura [45]. Esta memoria se libera automáticamente en caso de ser necesario.

CAPÍTULO 7. CONCLUSIONES Y LÍNEAS FUTURAS

En este último capítulo se exponen las conclusiones extraídas durante la realización del TFM y las posibles líneas de desarrollo futuro, proponiendo nuevas ideas que amplíen las funcionalidades de la aplicación desarrollada.

7.1 Conclusiones

En este TFM se ha desarrollado un sistema de monitorización activa de servidores adaptado a las necesidades de una filial de la empresa Hewlett-Packard. El sistema que utilizaban hasta entonces, basado en *scripts* en Bash, tenía importantes limitaciones que impedían la automatización de algunas tareas de monitorización. Esto obligaba a la empresa a destinar recursos humanos en tareas que podían ser automatizadas mediante las herramientas de *software* adecuadas.

El sistema resultante es una aplicación capaz de ejecutar comandos en servidores remotos y procesar los resultados para detectar posibles anomalías en su funcionamiento; cuando se producen incidencias la aplicación envía correos electrónicos al administrador. La herramienta también permite visualizar de forma directa y sencilla el estado de todas las máquinas y servicios que están siendo

monitorizados para que el administrador sea capaz de detectar los problemas rápidamente y tomar las medidas necesarias para solucionarlos.

En este trabajo se han analizado diferentes alternativas en cuanto a lenguajes de programación, lenguajes *web* y bases de datos, comparándolas y seleccionando las que mejores resultados pueden ofrecer para la realización de este trabajo. Finalmente se ha empleado el lenguaje de programación Python para el desarrollo de las herramientas y se ha optado por SQLite como sistema de gestión de bases de datos. Además, se ha utilizado HTML, JavaScript y CSS para el desarrollo de la interfaz *web*.

Una vez se han tenido claros los requisitos funcionales y no funcionales que debía cumplir la aplicación, se tomaron diferentes decisiones de diseño. Una de las más importantes fue la separación de la aplicación en cuatro herramientas distintas: herramienta de monitorización, herramienta de visualización y dos herramientas auxiliares, para cifrar las contraseñas y para probar en envío de notificaciones por correo electrónico. También se diseñó la base de datos y el formato de los ficheros de configuración.

Después de que la aplicación ha sido desarrollada, se ha procedido a realizar una revisión de la misma con el tutor de este TFM en la empresa, dándole acceso a todas las herramientas. A partir de la realimentación obtenida, y tras las correcciones necesarias, se puede asegurar que se cumplen con las expectativas de la empresa para con la herramienta desarrollada.

Por último, se han realizado dos manuales, un manual de instalación para diferentes plataformas y un manual de administrador, con el fin de que queden patentes todas las posibilidades que permite el sistema y se pueda asegurar que todos los usuarios saben explotarlas o disponer de una ayuda adicional en caso de duda. El manual de administrador de la aplicación desarrollada está destinado a los trabajadores de la filial de Hewlett-Packard y otro personal relacionado que vaya a hacer uso de ella.

Del trabajo realizado se concluye que las herramientas desarrolladas cumplen con todos los requisitos especificados por la empresa y estas solventan las distintas problemáticas que dieron lugar a la realización de este TFM. Esto permite aportar una serie de beneficios a la empresa, como son la mejora en la eficiencia del empleo de recursos humanos por parte de la empresa y la solución de las limitaciones de su actual herramienta de trabajo.

A título personal, creo que la realización de este proyecto ha sido una experiencia enriquecedora, que me ha permitido conocer y llevar a la práctica las diferentes fases necesarias en el desarrollo de un proyecto real de *software* para una empresa. Cabe destacar el hecho de que para realizar el proyecto se ha partido de cero, esto es, no había ninguna base sobre la que trabajar. Por esto, era necesaria una fase previa de toma de requisitos en la que tuve que reunirme con el tutor del TFM en la empresa. Adicionalmente, estas reuniones se repitieron a lo largo de todo el proceso de desarrollo del trabajo, por lo cual había una realimentación continua y se iba ajustando, cada vez más, la plataforma a las funcionalidades que buscaba el cliente. Una de las mayores dificultades fue decidir cuáles eran las funcionalidades más importantes, entre las que requería el cliente, de forma que no nos excediéramos mucho en la duración estimada que debía tener el proyecto. Creo que el trabajo realizado engloba varias facetas que un ingeniero debe dominar, además de las técnicas, como son poder tratar con el cliente, tener capacidad analítica para extraer unos requisitos viables a partir de las peticiones del cliente, saber realizar un diseño factible y eficiente a partir de estos y, sobre todo, saber organizarse para la consecución de unos objetivos.

7.2 Líneas futuras

Conforme se ha ido desarrollando la aplicación, han ido surgiendo posibles vías de continuación o mejora del trabajo que, bien por no ser objetivos principales, o bien por el planteamiento concreto del proyecto, no ha sido posible realizarlas en el presente trabajo. Se plantean aquí, por tanto, estos posibles desarrollos o líneas de análisis futuras.

En primer lugar, se podría mejorar el sistema de notificaciones. El sistema de notificaciones que implementa la aplicación actualmente solo permite enviar notificaciones cuando se detecta algún problema en los servidores monitorizados. Sería posible extender este sistema permitiendo el envío de informes diarios o semanales con el estado de todas las máquinas. Para ampliar las funcionalidades, otra posible mejora sería el envío de correos electrónicos a varias personas, actualmente solo permite especificar una única dirección de correo electrónico en la configuración.

Otra posible mejora de la aplicación estaría relacionada con la interfaz *web*. El sistema actual permite ver la información más importante, pero sería interesante desarrollar nuevas funcionalidades para esta interfaz. Se podrían incluir gráficas donde se muestre el tiempo que tardan en responder las máquinas, estadísticas sobre el número de incidencias detectadas cada mes, etc. También sería útil poder modificar la configuración de la aplicación desde la interfaz *web*, esto permitiría modificar la configuración de forma remota.

Por último, se propone el desarrollo de nuevos módulos para la aplicación. En la aplicación actual se han incluido cinco módulos de propósito general que permiten realizar muchas de las tareas a las que se enfrenta un administrador, pero, la aplicación no está limitada solo a esos módulos, es posible desarrollar nuevos módulos de forma sencilla. En el capítulo 4 se explica en detalle el contenido de cada módulo y las instrucciones para crear un nuevo módulo. Estas instrucciones podrían ser muy útiles si alguien decide continuar el desarrollo de este punto.

BIBLIOGRAFÍA

[1] Datacenter Dynamics. *Los servidores han sido una "caja negra" en el data center.* [Consultado el 7 de septiembre de 2015]. http://www.datacenterdynamics.es/focus/archive/2013/07/los-servidores-han-sido-una-%E2%80%9Ccaja-negra%E2%80%9D-en-el-data-center

[2] Microsoft Corporation. *Performance and Reliability Monitoring.* [Consultado el 7 de septiembre de 2015]. https://msdn.microsoft.com/en-us/library/bb742467.aspx

[3] Monitis. *11 Top Server Management & Monitoring Software.* [Consultado el 7 de septiembre de 2015]. http://www.monitis.com/blog/2011/02/22/11-top-server-management-monitoring-software/

[4] Red Hat Cygwin. *Cygwin Product.* [Consultado el 7 de septiembre de 2015]. https://cygwin.com/

[5] Python Software Foundation. *CPython.* [Consultado el 7 de septiembre de 2015]. https://www.python.org/

[6] SQLite Consortium. *SQLite.* [Consultado el 7 de septiembre de 2015]. https://www.sqlite.org/

[7] The Eclipse Foundation. *Eclipse.* [Consultado el 7 de septiembre de 2015]. https://eclipse.org/

[8] Brainwy Software Ltda. *PyDev.* [Consultado el 7 de septiembre de 2015]. http://www.pydev.org/

[9] DB Browser for SQLite Team. *DB Browser for SQLite.* [Consultado el 7 de septiembre de 2015]. http://sqlitebrowser.org/

[10] OpenBSD. *OpenSSH.* [Consultado el 7 de septiembre de 2015]. http://www.openssh.com/

[11] Mozilla Foundation. *Mozilla Firefox.* [Consultado el 7 de septiembre de 2015]. https://www.mozilla.org

[12] Canonical Ltd. *Ubuntu.* [Consultado el 7 de septiembre de 2015]. http://www.ubuntu.com/

[13] Raspbian Team. *Raspbian.* [Consultado el 7 de septiembre de 2015]. https://www.raspbian.org/FrontPage

[14] Raspberry Pi Foundation. *Raspberry Pi.* [Consultado el 7 de septiembre de 2015]. https://www.raspberrypi.org/

[15] Microsoft Corporation. *Microsoft Windows.* [Consultado el 7 de septiembre de 2015]. https://www.microsoft.com/es-es/windows

[16] Arturo Fernández Montoro. *Python 3 al descubierto.* RC Libros, 2013. ISBN: 8493945048, 9788493945046.

[17] The Python Software Foundation. *The Python Software Foundation (PSF).* [Consultado el 7 de septiembre de 2015]. https://www.python.org/psf/

[18] Guido van Rossum *"Guía de aprendizaje de Python."* FredL.Drake, Jr., editor. Python Software Foundation. Release 2.4.la0-11Septiembre, 2005. [Consultado el 7 de septiembre de 2015]. http://pyspanishdoc.sourceforge.net/tut/tut.html

[19] Perl.org. *The Perl Programming Language.* [Consultado el 7 de septiembre de 2015]. https://www.perl.org/

[20] Perl.org. *Comprehensive Perl Archive Network.* [Consultado el 7 de septiembre de 2015]. https://www.perl.org/cpan.html

[21] Python Software Foundation. *sqlite3 — DB-API 2.0 interface for SQLite databases.* [Consultado el 7 de septiembre de 2015]. https://docs.python.org/2/library/sqlite3.html

[22] Python Software Foundation. *SimpleHTTPServer — Simple HTTP request handler.* [Consultado el 7 de septiembre de 2015]. https://docs.python.org/2/library/simplehttpserver.html

[23] Cobo, A., Gómez, P., Pérez, D. y Rocha, R. (2003). *PHP y MySQL. Tecnologías para el desarrollo de aplicaciones web.* ISBN: 84-7978-706-6, 0-07-123151-X.

[24] J. Carlos-Orós. (2002). *Diseño de páginas web interactivas con JavaScript y CSS.* (3ª edición). Madrid: RA-MA.

[25] The jQuery Foundation. *jQuery.* [Consultado el 7 de septiembre de 2015]. https://jquery.com/

[26] J. Franklin. (2013). *Beginning jQuery.* Apress. ISBN: 978-1-4302-4932-0 (Print) 978-1-4302-4933-7

[27] M. Firtman. (2008). *AJAX: Web 2.0 para profesionales.* Marcombo S.A. ISBN 9788426714824

[28] W3C.org. *World Wide Web Consortium (W3C).* [Consultado el 7 de septiembre de 2015]. http://www.w3.org/

[29] Connolly, T., Begg, C., Pearson y Wesley, A. (2005). *Sistemas de bases de datos: un enfoque práctica para diseño, implementación y gestión.* ISBN: 8478290753, 9788478290758

[30] Chong, R., Hakes, I. y Ahuja, R. (2009). *Conociendo el DB2-Express-C.* (3ª edición).

[31] ODBMS.org. *Object Data Management Group.* [Consultado el 7 de septiembre de 2015]. http://www.odbms.org/odmg-standard/

[32] Ramakrishnan, R., Gehrke, J., Derstadt, J., Selikoff, S. y Zhu, L. (2003). *Database Management System.* (Third Edition). McGRAW-HILL. ISBN: 0-07-246563-8-ISBN 0-07-115110-9 (ISE).

[33] Silberschatz, A. y Korth, H. (2002). *Fundamentos de bases de datos.* (4ª edición). McGRAW-HILL. ISBN: 0-07-228363-7, 84-481-3654-3.

[34] Rivero, E., Martínez, L. y Alonso, I. (2005). *Bases de datos relacionales: fundamentos y diseño lógico.* (1ª edición). Universidad Pontifica Comillas de Madrid. ISBN: 8484681726, ISBN-13: 9788484681724.

[35] M. Antón, I. De la Torre, P. Gutiérrez, F. J. Díaz, M. Martínez, D. González, J. F. & D. Boto. (2010). *Sistema de acceso inalámbrico para la gestión de historiales clínicos electrónicos de pacientes con discapacidad cognitiva.* Actas de la Conferencia IADIS Ibero Americana WWW/Internet 2010, Algarve (Portugal). ISBN: 978-972-8939-34-2.

[36] MySQL.org. *MySQL.* [Consultado el 7 de septiembre de 2015]. https://www.mysql.com/

[37] Welling, L. y Thompson, L. (2003). *PHP and MySQL Web Development.* (Second Edition). Sams Publishing.

[38] Ian Sommerville, María Isabel Alfonso Galipienso. *Ingeniería del software.* Pearson Educación, 2005. ISBN: 8478290745, 9788478290741

[39] United States National Institute of Standards and Technology (NIST). Federal Information Processing Standards Publication 197. *"Announcing the ADVANCED ENCRYPTION STANDARD (AES)*" [Consultado el 7 de septiembre de 2015]. http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

[40] Martin Fowler, Kendall Scott. *UML gota a gota.* Pearson Educación, 1999. ISBN: 9684443641, 9789684443648.

[41] The Python Software Foundation. *BaseHTTPServer — Basic HTTP server.* [Consultado el 7 de septiembre de 2015]. https://docs.python.org/2/library/basehttpserver.html

[42] Red Hat Cygwin. *How do I fix fork() failures?* [Consultado el 7 de septiembre de 2015]. https://www.cygwin.com/faq.html#faq.using.fixing-fork-failures

[43] Red Hat Cygwin. *cygwin: not administrator account?* [Consultado el 7 de septiembre de 2015]. https://www.cygwin.com/ml/cygwin/2012-06/msg00548.html

[44] Die.net. *Manual del comando top.* [Consultado el 7 de septiembre de 2015]. http://linux.die.net/man/1/top

[45] The Python Software Foundation. Garba*ge Collector interface.* [Consultado el 7 de septiembre de 2015]. https://docs.python.org/2/library/gc.html