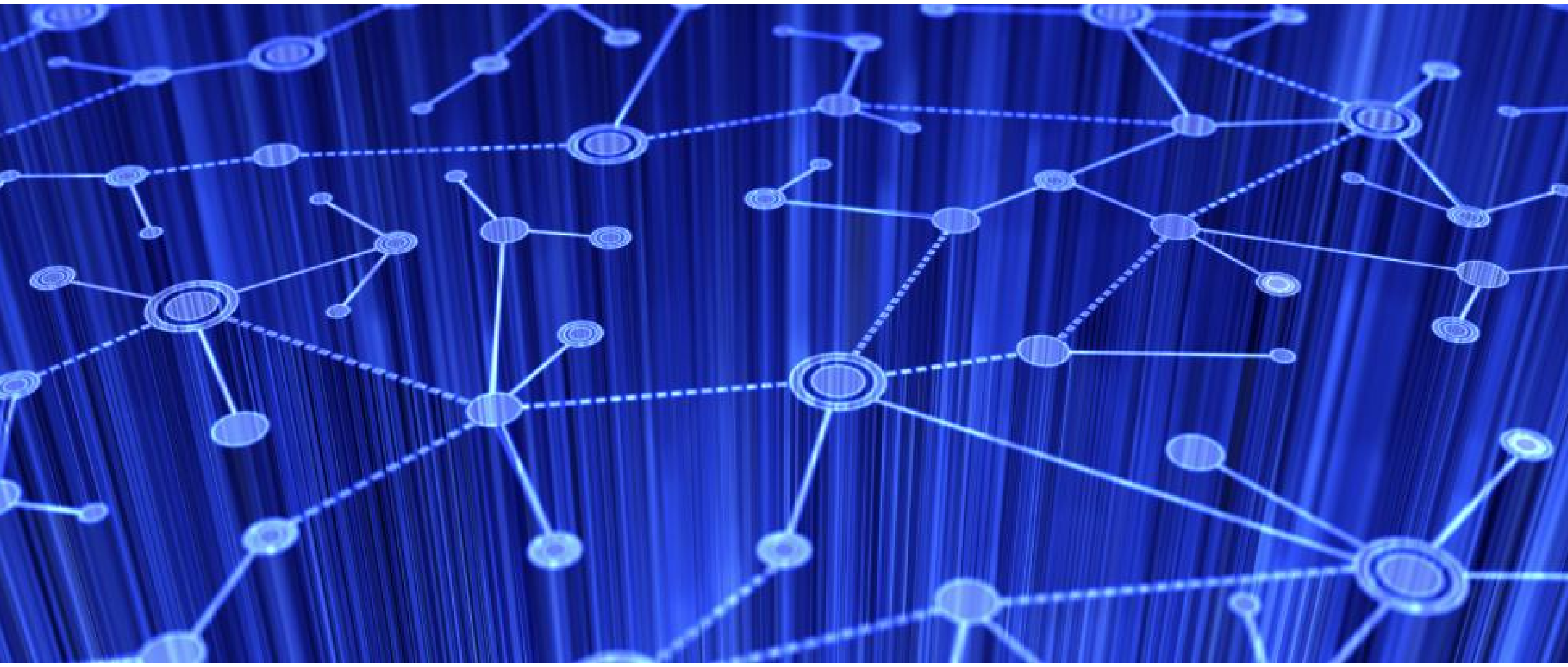
UNIVERSITAT OBERTA DE CATALUNYA



SISTEMA GESTOR DE RECURSOS PARA

UNA ARQUITECTURA GRID

Estudios	Ingeniería Técnica Informática de sistemas 2003
Asignatura	05.138 TFC - Aplicaciones web para el trabajo colaborativo
Curso	febrero 2015 - junio 2015
Consultor	Ferran Prados
Alumno	Alberto Ramírez Fernández

Índice de Contenidos

1. Introducción y objetivos del proyecto

1.1 Marco del proyecto

En un entorno donde los cálculos computacionales cada vez se vuelven más complejos, entre otras cosas por la cantidad de información que se genera a diario en Internet, dotar a nuestras infraestructuras de más poder de procesamiento mediante componentes hardware más potente y caro (lo que se conoce como modelo de escalado vertical) resulta prácticamente inviable para la mayoría de empresas y/o organizaciones.

Por otro lado, los HPC (High-Performance Computer) o supercomputadores solo están al alcance de unas pocas organizaciones, universidades y empresas que se dedican al ámbito de la investigación. Por esto surgió la necesidad de resolver el problema de otra forma, dando lugar a un nuevo paradigma de computacional consistente en añadir más recursos físicos mediante el incremento de servidores de bajo y medio coste y prestaciones medias, esto se conoce como **escalabilidad horizontal** (Ver figura 1).

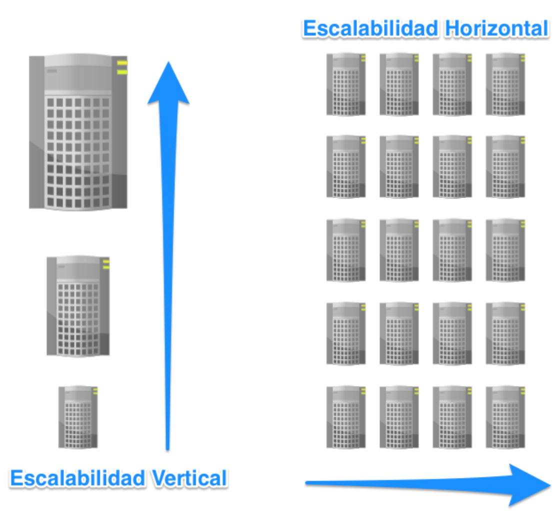


Figura 1

Nace con esto el concepto de computación en GRID, que es un tipo de computación **distribuida** y que trata de utilizar de forma coordinada todo tipo de recursos, desde CPU's hasta memoria o almacenamiento.

Sin embargo, esto plantea una nueva dimensión de problemas prácticamente inexistentes hasta la fecha, como son la gestión de estos recursos y la comunicación entre todas las máquinas que forman el GRID, que pueden estar conectadas entre sí mediante redes tan extensas como la propia Internet.

1.2 Objetivos

Con este proyecto se plantea resolver este problema: **gestión de recursos** de forma **transparente** y **comunicación entre** las **máquinas** que componen el GRID.

Para ello se desarrollará un **sistema web** que permita **añadir, eliminar y listar** estos **recursos** (máquinas que componen el GRID), número de **CPU's** de las mismas, **memoria** y capacidad de **almacenamiento**.

Por otro lado, otro punto importante para el desarrollo de este proyecto es la comunicación entre el sistema gestor y la totalidad de máquinas que conforman el GRID, de forma que el gestor conozca los recursos de cada una de ellas y si se encuentran o no **disponibles** en todo momento.

1.3 Resultados esperados

Con este proyecto se espera facilitar la gestión y administración de dispositivos que se adhieren a una red o sistema GRID, en concreto se espera poder:

- Visualizar de forma sencilla este GRID de ordenadores.
- Conocer la disponibilidad de cada uno de los ordenadores.
- Tener la capacidad de eliminar de la lista dispositivos no disponibles.

2. Estudio de viabilidad

Para hacer posible este proyecto y que se lleve a cabo de la mejor forma posible, cabe destacar que el coste del mismo es casi nulo, ya que para su desarrollo se utiliza por completo software libre (desde servidor web Apache, lenguajes de programación PHP, Golang o Javascript, bases de datos, etc.).

El entorno de desarrollo local, compuesto principalmente por Vagrant, VirtualBox y Ansible en completamente gratuito también. El IDE utilizado para el desarrollo de código en este caso es PHPStorm, del cuál se ha obtenido una licencia académica.
En cuanto al esfuerzo humano, se han usado las horas correspondientes al calendario de la asignatura.

Tecnológicamente no existen impedimentos que hagan este proyecto inviable. Todos los objetivos determinados, que ha de cumplir este proyecto son satisfactible mediante el uso de herramientas comunes, lenguajes de programación como PHP o Golang.

Para la implantación o puesta en producción se ha usado un servidor dedicado comercial, adquirido a la empresa *Kimsufi*, del que ya se disponía previamente, por lo que no supone gasto adicional.

3. Metodología

El proyecto se llevará a cabo mediante un diseño iterativo e incremental, primando así la velocidad con la que se obtiene feedback en un escenario empírico, eliminando largas fases de análisis y diseño muy separadas en el tiempo de la codificación final.

De esta forma se consigue mayor agilidad, una respuesta al cambio en menos tiempo y de forma más precisa. Se tratará de hacer ciclos cortos de diseño y desarrollo para aprender rápidamente, poder mejorar la comunicación y el cambio de dirección en caso de error.

No se realizarán diseños completos desde el principio, como diagramas entidad-relación o diagramas de clases de la aplicación completa. Estos se dividirán por componentes.

4. Planificación

4.1 Plan de trabajo

4.1.1. Especificación y Análisis

- Definir el ámbito del proyecto y sus límites.
- Estudio de mercado y análisis de herramientas similares.
- Definir los casos de uso de la aplicación.
- Primer boceto de los principales componentes que conformarán el proyecto: comunicaciones y flujos de datos y su arquitectura.

4.1.2. Diseño

- Definición de metodologías y herramientas de trabajo. Lenguajes y librerías para el posterior desarrollo.
- Detalle más exhaustivo de cada caso de uso.
- Análisis de aspectos no funcionales como la seguridad de las comunicaciones entre el gestor y las máquinas, el acceso al propio gestor mediante credenciales, rendimiento del script o servicio que se ejecuta en cada una de las máquinas, etc.
- Diseño de arquitectura de la aplicación y el tipo de comunicación entre el gestor y las máquinas del grid.

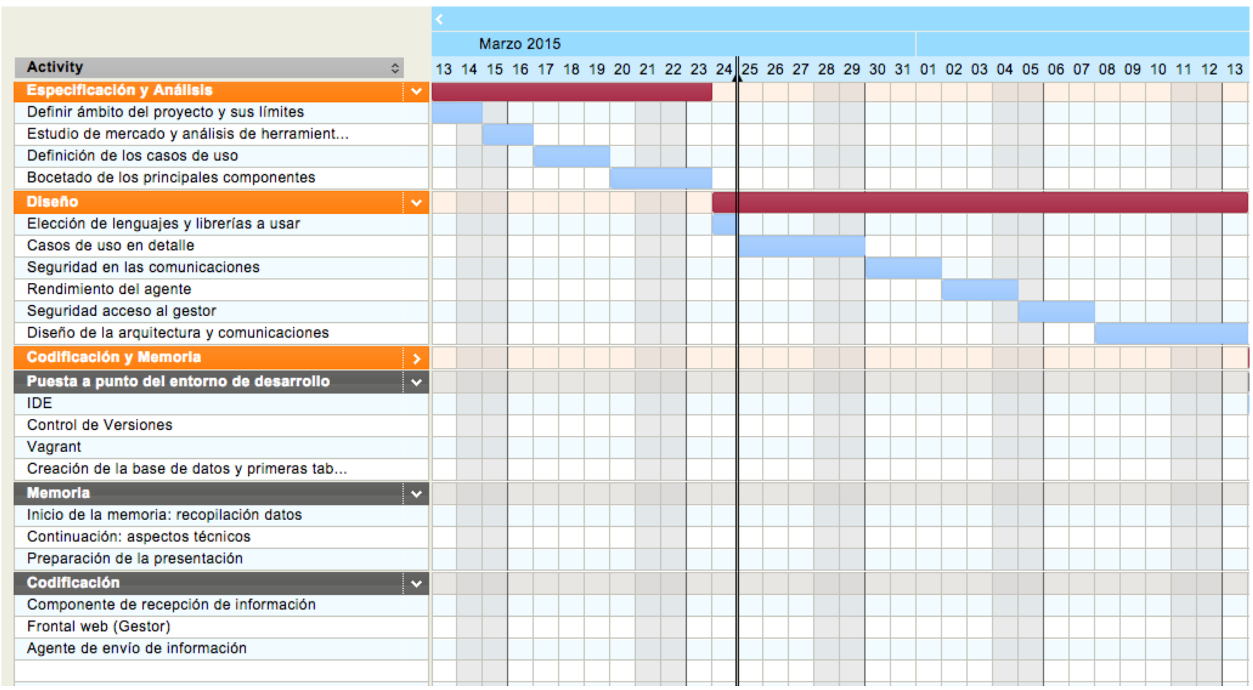
4.1.3. Codificación, Memoria y Presentación

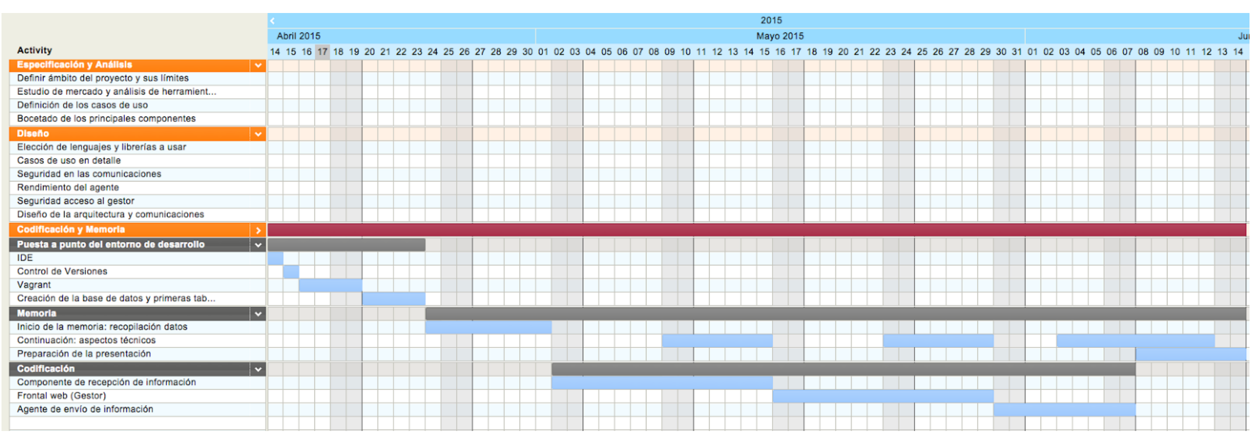
- Puesta a punto del entorno del entorno de trabajo: automatización del entorno de desarrollo, seleccionar IDE, sistema de control de versiones.
- Creación de base de datos y tablas basado en el diagrama entidad/relación de forma iterativa e incremental.
- Inicio de la memoria del proyecto: recopilación de toda la información previamente documentada, clasificación y ordenación de la misma, dando lugar a la primera fase de la memoria.
- Codificación del proyecto por componentes:
    - Componente central de recepción de información de las máquinas. Será encargado de recibir la información proveniente de las máquinas que conforman el GRID.
    - Frontal web para la visualización de los recursos disponibles. Se trata del gestor que permitirá ver qué recursos hay disponibles.

- Servicio cliente encargado de la monitorización de la máquina: instalado en cada una de las máquinas del grid, será el servicio responsable de enviar información periódicamente al componente central de recepción.

- Continuación de la memoria para añadir todo el apartado técnico y de codificación.
- Presentación y defensa.

4.2 Tareas planificadas





PAC1: Plan de trabajo		**9 marzo**
PAC2: Especificación y Análisis	**13 marzo**	**23 marzo**
Definir ámbito del proyecto y sus límites	13 marzo	14 marzo
Estudio de mercado y análisis de herramientas similares	15 marzo	16 marzo
Definición de casos de uso de la aplicación	17 marzo	19 marzo
Bocetado de los principales componentes y sus comunicaciones	20 marzo	23 marzo
PAC3: Diseño	**24 marzo**	**13 abril**
Elección de lenguajes y librerías a usar por cada componente	24 marzo	24 marzo
Desarrollo más exhaustivo de todos los casos de uso, creación de diagramas de actividad y secuencia	25 marzo	29 marzo
Aspectos no funcionales de la aplicación: - Seguridad en las comunicaciones. - Rendimiento del agente instalado en las máquinas - Seguridad para el acceso al gestor	30 marzo	7 abril
Diseño de arquitectura de la aplicación y el tipo de comunicación entre el gestor y las máquinas del grid.	8 abril	13 abril
PAC4: Codificación y Memoria	**14 abril**	**14 junio**
Puesta a punto del entorno de trabajo: - Selección de IDE - Sistema de Control de Versiones - Automatización del entorno con Vagrant - Creación de la Base de Datos y las primeras tablas	14 abril	23 abril
Inicialización de la memoria, recopilación de la documentación escrita hasta el momento.	24 abril	1 mayo
Codificación del componente de recepción de información de las máquinas que componen el grid	2 mayo	15 mayo
Codificación del frontal web para la visualización de los recursos del grid	16 mayo	29 mayo
Codificación del agente que será instalado en cada máquina del grid y enviará información de las mismas al gestor central	30 mayo	7 junio
Continuación de la memoria con los datos técnicos recopilados de la codificación	9 mayo 23 mayo 3 mayo	15 mayo 29 mayo 12 junio
Preparación de la presentación	8 junio	14 junio

5. Marco de trabajo

5.1 Definición del ámbito del proyecto

Este proyecto trata de resolver la gestión transparente de recursos de un sistema GRID mediante una aplicación web gestora. Este gestor web permitirá listar los recursos disponibles en el grid y gestionar de forma limitada estos.

En el contexto actual en el que vivimos, con la creciente necesidad de computación distribuida, este tipo de arquitecturas en GRID cada vez son más frecuentes, de esta forma intentamos proveer de una herramienta que permita gestionar los recursos que componen un sistema GRID.

5.2. Estudio de mercado y análisis de herramientas similares

En la actualidad el uso de sistemas de computación distribuida está muy extendido, Internet y la tremenda cantidad de datos generados a diario, han desembocado en la necesidad por parte muchas empresas a la investigación para alcanzar la máxima eficiencia al mínimo coste.

Nacieron varios movimientos destinados a romper el monopolio de los HPC (High Performance Computers), entre estos movimientos los más destacados son:

Cloud Computing Systems (sistemas de computación en la nube)

Varias empresas *TOP* de internet, con gran cantidad de hardware, se percataron de que existía esta gran demanda por parte de otras empresas de tamaño más pequeños de una gestión flexible, elástica y transparente de hardware. De este modo, empresas como Amazon decidieron “re-vender” parte de su infraestructura como negocio secundario,

dando lugar a un nuevo paradigma, continuó Google, quien dio una vuelta más a este paradigma y luego se sumaron otros grandes como Microsoft.

- **IaaS (Infrastructure as a Service):** consiste en poner a disposición de forma transparente hardware, sin necesidad de que los clientes se preocupen por aspectos puramente físicos. El principal comercializador de IaaS es Amazon, con AWS (Amazon Web Services) y sus principales productos son S3 (Simple Storage Service) que ofrece almacenamiento en la nube y EC2 (Elastic Compute Cloud) que pone a disposición de usuarios servidores en diferentes configuraciones para que éstos puedan instalar el software al gusto.

- **PaaS (Platform as a Service):** este modelo de gestión de recursos en *la nube* consiste en ofrecer, no solo recursos físicos sino, una plataforma de software donde los clientes pueden ejecutar su software propietario. Los mayores comercializadores de PaaS son **Google** con **App Engine** y **Microsoft** con **Azure**.

Computación Grid

La computación en grid nace como metáfora de que acceder a la potencia de ordenadores sea tan sencillo como acceder a la red eléctrica. Con la investigación en entornos educativos nacieron múltiples sistemas grid que permitían la gestión de maquinaria menos potente (como estaciones de trabajo de las universidades y centros educativos) de forma unificada, tal que se consiguiera una revolución en la computación distribuida. Por primera vez se consigue unificar potencia de procesamiento, almacenamiento o monitorización de forma transparente pero a la vez distribuida en diferentes ordenadores y/o dispositivos electrónicos. Aparecen sistemas para la gestión de redes grid:

- Globus: se presenta como estándar de facto para la gestión de recursos, descubrimiento y monitorización y almacenamiento de datos.
- Sun Grid Engine: gestión de recursos y asignador de tareas (Scheduler).
- Ganglia: monitorización de recursos.

6. Requisitos del sistema

6.1. Límites del proyecto

Una vez definido el contexto del proyecto, cabe definir sus límites y hasta dónde abarca el mismo. Para ello se ha elaborado la siguiente lista de puntos:

- Crear una aplicación web para gestionar los recursos del grid.
- La aplicación web debe permitir listar estos recursos y su disponibilidad.
- Por cada máquina agregada al grid se ha de poder visualizar CPU, Memoria y Disco de que disponen.
- La aplicación web debe poder eliminar recursos.
- Los usuarios podrán instalar el agente en sus ordenadores para comunicarse con el grid.
- El agente enviará datos periódicamente a la aplicación web gestora.
- El acceso a la aplicación web será restringido a usuarios mediante identificador y contraseña.

6.2. Principales componentes que conforman el sistema

6.2.1. Comunicaciones

La instalación de un agente en máquinas "clientes" que envían información moderadamente sensible, como IP, hostname y hardware, a través de un canal inseguro como lo es Internet, plantea una problemática, hay necesidad de cifrar esta comunicación para impedir el uso indebido mediante robo de información por sniffing de red, man in the middle u otras técnicas similares. Para ello, el agente deberá, o bien cifrar el mensaje a enviar o bien establecer un canal seguro, como pueden ser conexiones TSL o SSL.

6.2.2. Flujos de datos y su arquitectura

Descripción de la figura mostrada a continuación: el **agente** se instalará en N máquinas diferentes, todas ellas de personas que quieran compartir sus recursos en el grid, o máquinas de centros educativos y universidades. Estos clientes crearán un canal seguro por el que viajarán los paquetes de información. Cada paquete de información podrá contar con la IP de la máquina cliente, la memoria disponible, cpu, etc.

Esta información llegará al componente de **recepción de información**, encargado de almacenarla en una base de datos centralizada. El sentido por el que este componente se separa de la aplicación web es debido a que se pretende que sea fácilmente escalable, es decir, si el número de máquinas clientes que corren el agente crece exponencialmente, la cantidad de información que el receptor de información recibirá será enorme, al ser un componente sencillo, es fácil de replicar y en ningún momento penalizará a los usuarios que estén accediendo a la web gestora del grid (previsiblemente con mucho menos demanda).

Por otro lado, existirá un número de visitantes que estarán accediendo a la aplicación **web de gestión del grid**, donde podrán visualizar listado de recursos y administrarlos. Esta información es accesible a través de la base de datos centralizada, donde el receptor de información guarda los datos. El número de visitantes es previsiblemente menor, por lo que tener esta aplicación desacoplada del receptor, permitirá una instalación sencilla en un servidor con no mucha demanda de procesamiento.

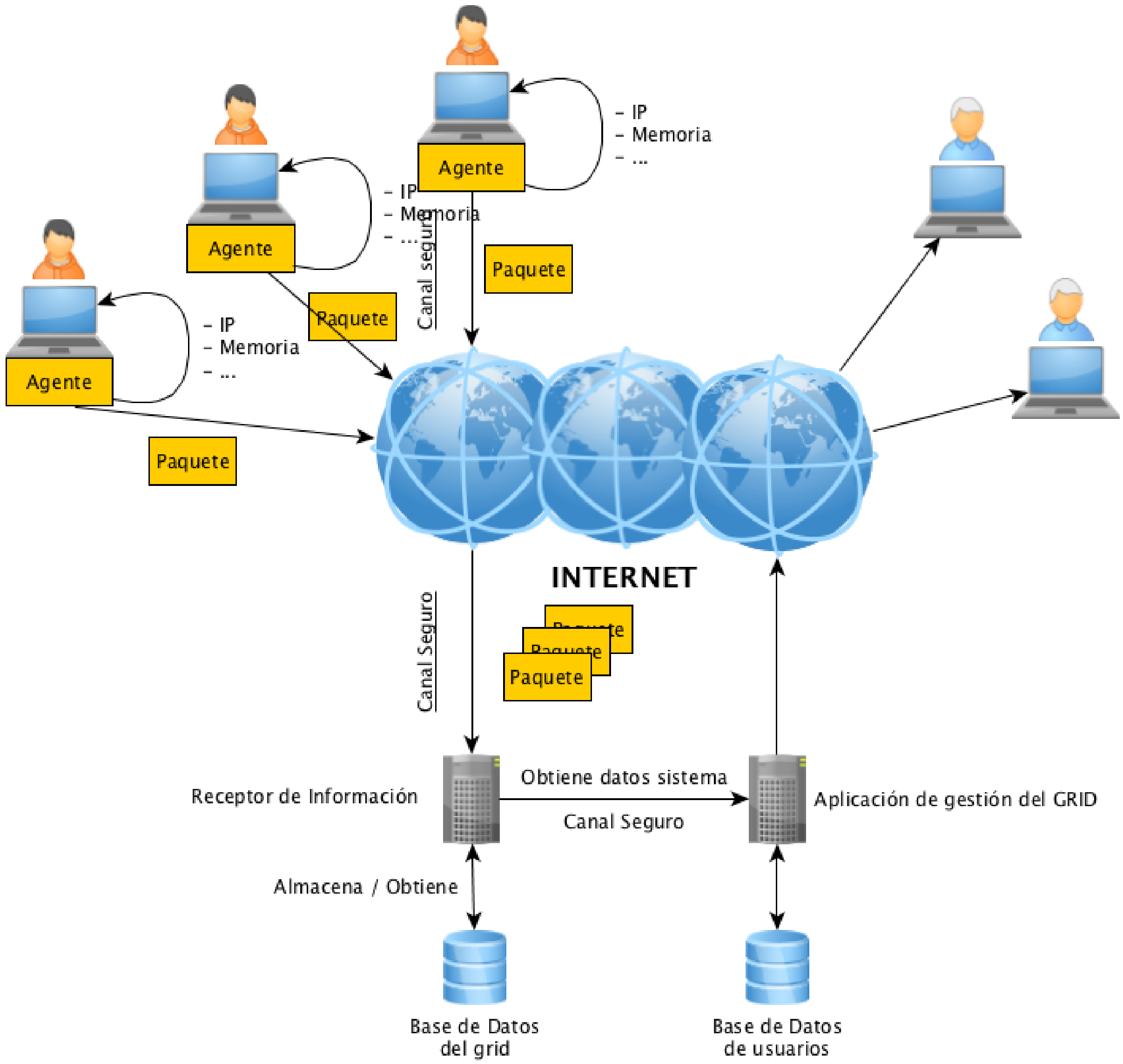


Figura 2.

Nota: *la figura 2 difiere de la imagen original, presentada en la PAC2, debido a que finalmente el modelo de comunicaciones entre la web y el receptor cambió, ahora es comunicación http (en el diseño original, la web accedía directamente a la base de datos del grid, causando una inconsistencia en la propiedad de los datos). Además, se ha añadido una base de datos propia al grid, para la gestión de usuarios.*

6.3. Definición de los casos de uso

6.3.1. Actores Implicados

Los actores implicados en este proyecto son:

- **Usuarios anónimos:** podrán descargar el agente y registrarse únicamente.
- **Usuarios registrados:** acceso mediante sus credenciales (usuario y contraseña) a operaciones de lectura, como listar los recursos disponibles.
- **Administradores:** a las funcionalidades de usuario registrado añaden la posibilidad de realizar operaciones de gestión, como eliminación de recursos.
- **Proveedores de recursos:** serán usuarios que quieran incluir su ordenador al grid, mediante la instalación de un agente.

6.3.2. Casos de Uso

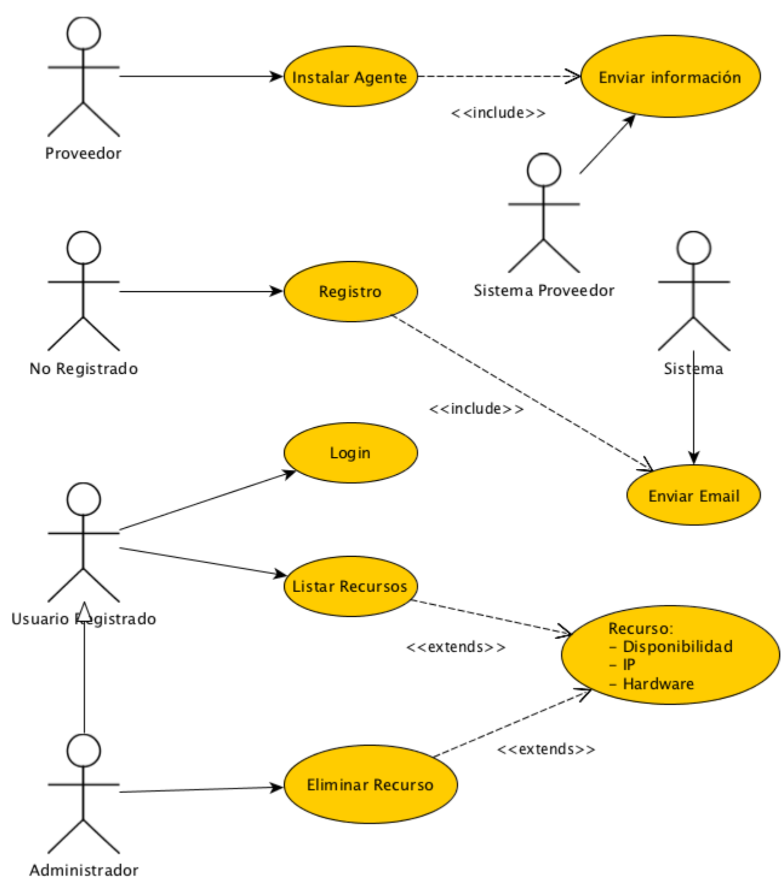


Figura 3.

6.3.3. Ficha de casos de uso

Nombre	Instalar agente
Descripción	Los usuarios que quieran incluir su máquina como recurso en el grid, pueden instalar un agente.
Precondición	-
Secuencia principal	00 Usuario descarga agente. 01 Usuario instala agente.
Errores / Alternativas	-
Postcondición	-

Nombre	Enviar información
Descripción	El agente instalado en la máquina del proveedor de recursos, enviará de forma periódica información al sistema.
Precondición	"Instalar Agente"
Secuencia Principal	00 Abrir un canal seguro con el servidor. 01 Recopilar información del sistema. 02 Enviar periódicamente esos datos.
Errores / Alternativas	-
Postcondición	-

Nombre	Registro de nuevo usuario
Descripción	Usuarios no registrados pueden registrarse en el sistema para acceder al mismo.
Precondición	Usuario no está logueado.

Secuencia Principal	00 El usuario abre la página principal 01 Pulsa en botón registro. 02 Rellena el formulario y lo envía
Errores / Alternativas	
Postcondición	El usuario debe recibir un email de confirmación

Nombre	Enviar email
Descripción	El servidor procesará el registro de un usuario y le enviará un email para confirmar la cuenta de correo electrónico.
Precondición	“El usuario se ha registrado”
Secuencia Principal	00 ver último usuario registrado 01 preparar un email de confirmación 02 enviar el email
Errores / Alternativas	
Postcondición	

Nombre	Login
Descripción	Los usuarios que tengan cuenta de acceso podrán identificarse en el sistema para poder interactuar con él.
Precondición	Usuario ha sido registrado
Secuencia Principal	01 Usuario va a la página principal 02 Usuario pulsa botón de login 03 Usuario rellena el formulario y lo envía
Errores / Alternativas	Error en los datos de login
Postcondición	El usuario recibe OK del sistema y le permite entrar.

Nombre	Listar recursos
Descripción	Un usuario puede listar los recursos registrados en el sistema, ver su disponibilidad, hardware e IP.
Precondición	El usuario está identificado
Secuencia Principal	01 El usuario entra en la página principal
Errores / Alternativas	
Postcondición	

Nombre	Eliminar recurso
Descripción	Los Administradores del sistema podrán eliminar recursos cuando estos no tengan disponibilidad en un tiempo determinado.
Precondición	- El usuario es administrador. - El recurso no ha tenido disponibilidad en un periodo de tiempo determinado
Secuencia Principal	00 El administrador accede a la lista de recursos 01 El administrador identifica un recurso no disponible 02 El administrador pulsa el botón eliminar
Errores / Alternativas	
Postcondición	El recurso desaparece el listado

7. Estudios y decisiones

Para la realización del proyecto se han usado las siguientes tecnologías y herramientas:

- Lenguaje principal: PHP, diseñado puramente para la web, potente y flexible, fácil de utilizar pero a la vez que permite hacer cosas complejas.
- Servidor Web: Apache, especialmente pensado para interactuar con PHP, es uno de los stack más famosos a nivel mundial y que más webs de hoy en día utilizan.
- Para la base de datos se han usado dos estrategias diferentes, cada una intentando resolver un problema diferente:
    - Recolector de información: para garantizar alta disponibilidad y tolerancia a errores de red, se ha usado **MongoDB**. Con esta base de datos sin esquema, podemos guardar diferente información dependiendo de cada sistema. Además, permite guardar grandes colecciones de datos sin penalizar en rendimiento (como es el caso de los healthcheck temporales).
    - Web Gestora del Grid: En este caso, para guardar la información de usuarios registrados, se ha escogido **MySQL**, porque nos permite guardar sobre todo **consistencia**, que es la mayor propiedad que queremos guardar tratándose de datos de usuario. Además, el volumen de datos es mucho menor que en el caso del recolector de información, por lo que no habrá problemas de rendimiento en este caso con MySQL.
- Desarrollo del Agente: **Golang**, por varios motivos: ante todo, permite una interacción muy directa con el sistema operativo, lo que facilita la toma de datos, además permite de forma sencilla operar con múltiples hilos concurrentes que, junto con el sistema de bucle infinito (avanzado con respecto a otros lenguajes), hacen muy sencillo recopilar información del sistema y realizar envíos periódicos. Golang también permite compilación para varios sistemas operativos, por lo que podríamos tener con mínimo esfuerzo el mismo agente compilado para Linux y OS X. Por último, cabe destacar el gran rendimiento de este lenguaje en general.

- La automatización para correr y aprovisionar una máquina virtual, de forma que el trabajo diario sea mucho más sencillo y portable, se lleva a cabo con Vagrant (definición de máquina virtual mediante un solo fichero de texto) mediante VirtualBox y Ansible, que usando sintaxis YAML, permite definir todo el aprovisionamiento de la máquina, desde configuraciones, paquetes instalados hasta incluso coordinar el deploy a producción.
- Para la parte de cliente, se usa Javascript, con en el framework jQuery (para peticiones AJAX y acceso a elementos del DOM) y Twitter bootstrap como framework CSS.
- Una vez decididos todas estas herramientas básicas, cabe destacar que se han elegido los siguientes frameworks o librerías para trabajar la parte de PHP:
    - Composer: en gran medida es la piedra angular de PHP, es un gestor de dependencias que permite definir y descargar y mantener actualizadas librerías y frameworks y establecer la estructura del proyecto.
    - Silex: es un microframework que permite de forma sencilla definir urls y ejecutar código asociado. Además, Silex provee un contenedor de dependencias que permite definir configuración o servicios disponibles.
    - Silex-Simpleuser: es un plugin para Silex que permite gestionar usuarios de manera sencilla, permitiendo: registro, login, recuperar contraseña, etc.
    - Para la gestión de plantillas html, se usa Twig, un gestor de plantillas muy sencillo de usar y flexible a su vez.
    - Debido a la necesidad de realizar peticiones HTTP con PHP desde el gestor web hacia el recolector de información, se elige usar una librería llamada Guzzlehttp, diseñada para lo propio.
    - Finalmente, para la realización de testing unitario del recolector de información, se decide usar PHPSpec. A diferencia de PHPUnit (de la rama XUnit), PHPSpec está más orientado a definir especificaciones de forma clara, en un lenguaje mucho más parecido al natural y sin el exceso de código normalmente requerido por PHPUnit.

8. Análisis y diseño del sistema

8.1. Diagramas de actividad de las acciones principales

8.1.1. Registro de usuario

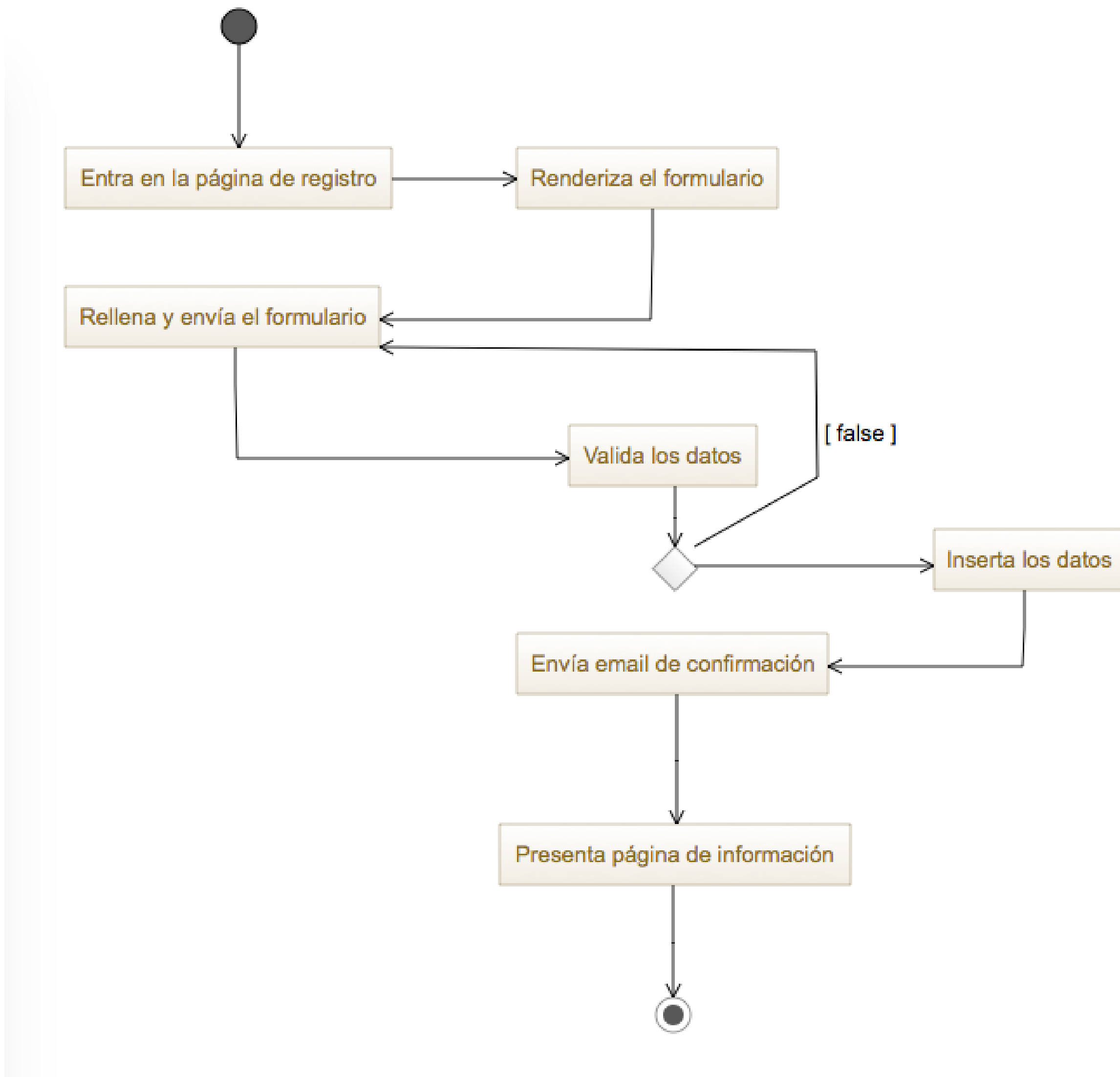


Figura 4.

8.1.2. Login

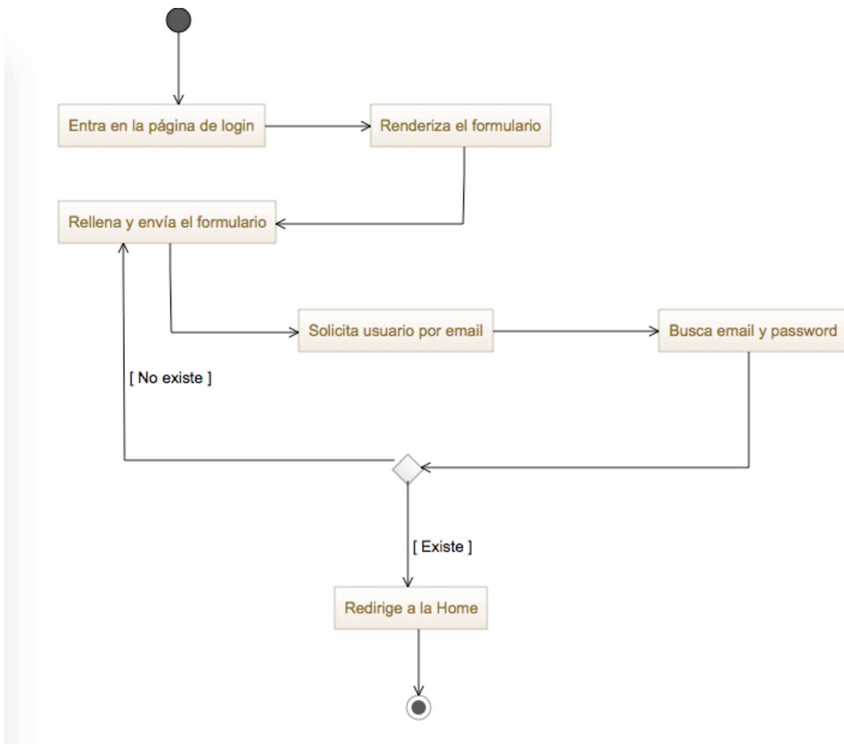


Figura 5.

8.1.3. Listar recursos

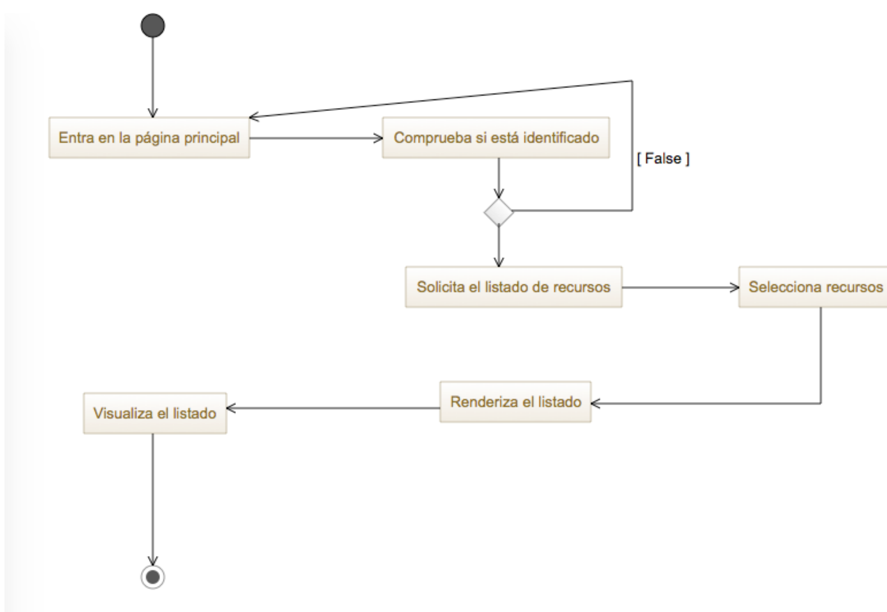


Figura 6.

8.1.4. Borrar recurso

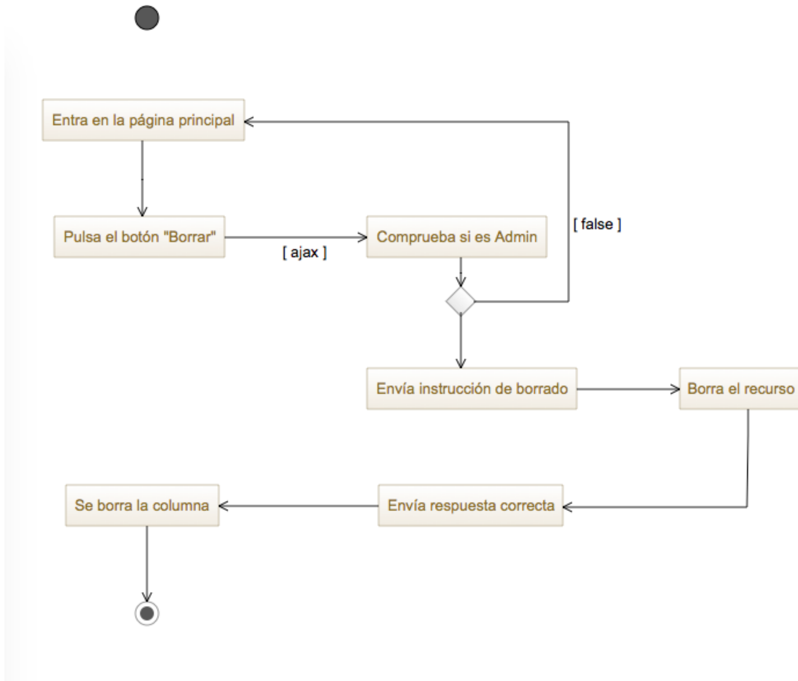


Figura 7.

8.2. Prototipo de las principales interfaces de usuario

8.2.1. Registro

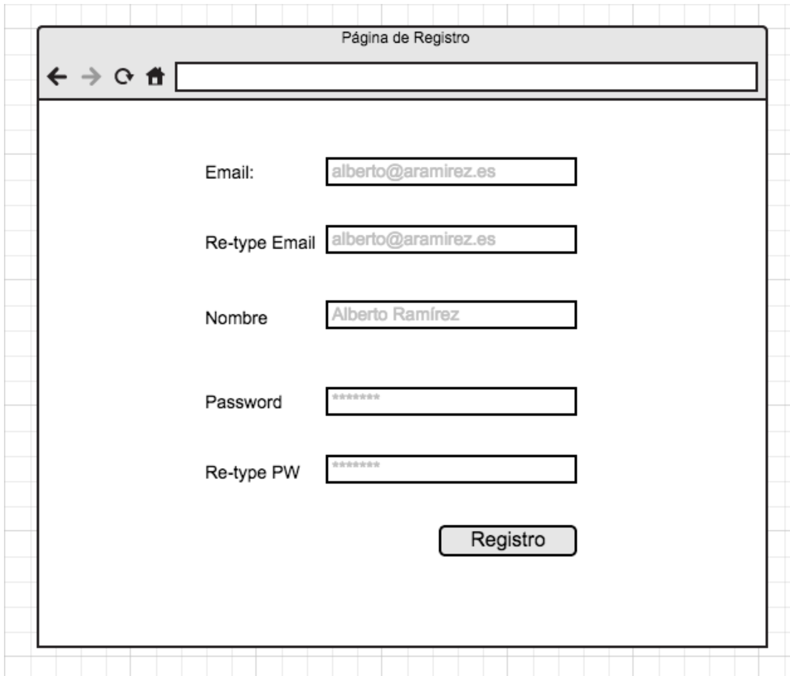


Figura 8.

8.2.2. Login

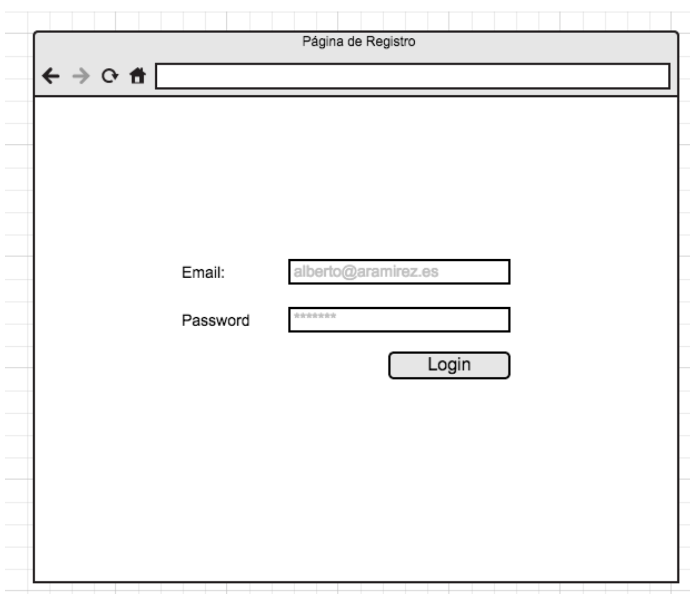


Figura 9.

8.2.3. Listar recursos

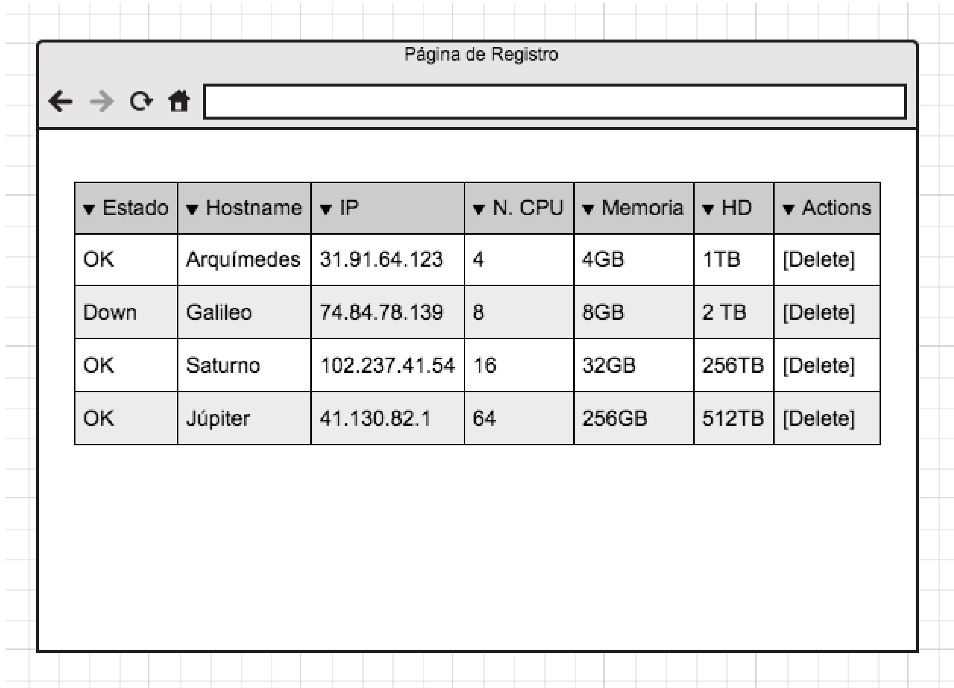


Figura 10.

8.2.4. Borrar recurso

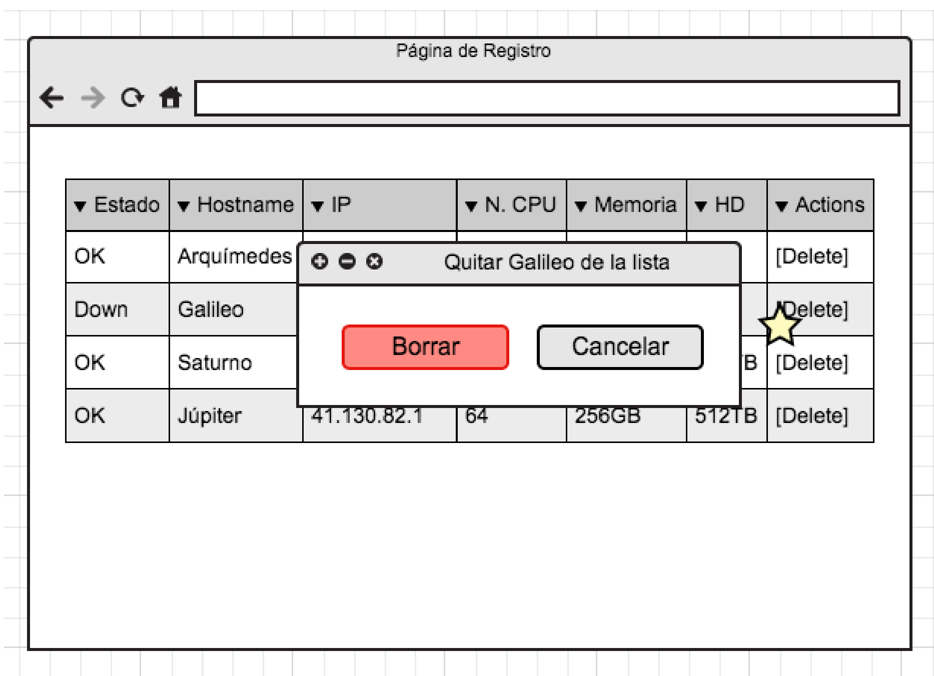


Figura 11.

8.3. Aspectos no funcionales de la aplicación

8.3.1. Seguridad en las comunicaciones

La api de recolección de información es un punto crítico debido a su naturaleza de almacén de datos de múltiples sistemas. Entre estos datos almacenados, existen direcciones IP, información del estado de la máquina, su sistema operativo, etc.

Además, a pesar de que en el diseño actual no está contemplado, cabría la posibilidad que en trabajos futuros sobre este mismo proyecto, se hiciera algún vínculo de máquinas con información más sensible de su propietario, por lo cual, debido a todo esto, es imprescindible pensar en la seguridad.

Dado que este componente es accesible vía HTTP mediante servidor web apache, se decide restringir la comunicación solo a protocolo **HTTPS** (HTTP + TLS). Esto permite que las comunicaciones entre este componente y los otros dos viajen de forma cifrada, no obstante, para evitar potenciales ataques de seguridad como *man in the middle* o suplantación o robo de identidad, se decide usar un certificado SSL autofirmado obligando así a que tenga que existir una autenticación servidor-cliente.

Para forzar esta comunicación https, el servidor web Apache solo responderá en puerto 443, y no en el puerto 80 (el puerto usado por protocolo http).

8.3.2. Seguridad en el acceso a la web

La web requiere de identificación para poder acceder al listado de recursos. Para este caso se provee de un formulario de registro, donde los usuarios que lo deseen podrán darse de alta para su posterior identificación mediante su email y la contraseña que hayan elegido en el momento de registro.

Por tanto, para el acceso a la web, será requisito indispensable pasar por una página de "login" y realizar la identificación previa.

Como nota adicional, comentar que la web se libera con un único usuario administrador, que será quién tenga permisos para borrar recursos cuando estos no se encuentren disponibles.

No toda la web requerirá que el usuario esté identificado para interactuar, en el caso de la página de descarga del agente (una página donde se mostrará la información pertinente para que los visitantes puedan descargar e instalar el agente en su máquina según su sistema operativo), no será necesario esta autenticación, de cara a que sea realmente sencillo que cualquier usuario pueda instalarse el agente.

8.3.3. Contrato Http

En el caso del componente de recepción de información se decide exponer una API RESTful como contrato, de forma que tanto el agente enviando datos como la web a la hora de consultar los mismos, tendrán que comunicarse con este componente mediante la API RESTful. La api rest permite modelar en forma de "recurso" o entidad la información que es accesible. A continuación se detallan los recursos y sus representaciones.

Systems

Operations

- `GET /systems` Retrive the collection of systems.
- `POST /systems` Create new system.
- `GET /systems/{id}` Get information about system resource with ID {id}.
- `PUT /systems/{id}` Update a system's full representation or created new one.
- `DELETE /systems/{id}` Remove the system with ID {id} from the collection.

Model representation

```
{
    id: "0AE82C63-3158-4D11-9613-DE4B55142CCD",
    hostname: "MacBook-Pro.local",
    ip: "37.135.124.191",
    cpus: 4,
    ram: "16.00GiB",
    hdd: "232.62GiB",
    os: "darwin",
    _link: "https://api.pfc.aramirez.es/systems/0AE82C63-3158-4D11-9613-DE4B55142CCD"
}
```

Figura 12.

Healthchecks

Operations

- `GET /systems/{id}/healths` Retrive the collection of system's healthchecks.
- `POST /systems/{id}/healths` Insert a new healthcheck in the system with ID {id}.

Model representation

```
{
    id: "0AE82C63-3158-4D11-9613-DE4B55142CCD",
    created_at: "2015-06-13T09:06:11+00:00",
    cpu_load: "2.22",
    used_ram: "14.14GiB",
    used_hdd: "124.88GiB"
}
```

Figura 13.

8.3.4. Arquitectura por capas

Una de las partes más críticas de todo el proyecto es el modelado de datos. Este modelado de datos definirá la forma que tienen las dos entidades principales de toda la aplicación: “systems” y “healthchecks”.

La primera entidad modela un recurso, una máquina o un sistema en definitiva (se puede apreciar la representación de su modelo en la figura 12) y contendrá información tal como el sistema operativo, un identificador único, la IP pública de la máquina, etc.

En cuanto a la entidad healthchecks, ésta representa el estado concreto de una máquina en un momento temporal concreto, es decir, contiene información física de la máquina (load average o ram usada por ejemplo) en una marca de tiempo (se puede ver este modelo en la figura número 13).

Debido a la criticidad de este modelo de datos y a que estos serán gestionados por el componente de recepción de información (data ownership), se elige un diseño por capas para la correcta representación de lo realmente importante y el **modelo de datos**. Para ello, se elige una arquitectura llamada Ports and Adapters, que enfatiza la esto por encima de detalles de implementación (a menudo con tiempo finito) como son el mecanismo de entrega (normalmente el framework web usado) o la base de datos concreta.

Pensando a largo término, los frameworks evolucionan, a menudos hay migraciones imposibles entre una versión antigua y una nueva (que conlleva normalmente tener que rehacer la aplicación por completo), el sistema de almacenamiento puede verse forzado e incluso llegar a convertirse en un sistema erróneo si la aplicación crece (y con más razón de ser en una aplicación diseñada para un sistema distribuido). Pero, lo que siempre perdura en el tiempo es la lógica del negocio, el core de la aplicación, el dominio (en términos matemáticos) y éste no debería estar acoplado a detalles de implementación, de forma que cambios futuros por migraciones o reemplazo de frameworks, obligue a rehacer por completo la aplicación.

Es ahí donde la arquitectura Ports and Adapter se hace más fuerte, permitiendo esa separación del core y de la parte de infraestructura. Además, esta separación hace que el **testing unitario** sea realmente mucho más sencillo, rápido y con menos posibilidades de verse afectado por agentes externos, ya que normalmente el core de la aplicación desarrollada siguiendo arquitectura Ports and Adapters no depende de ningún agente externo y se modelan mediante clases puras del lenguaje (en este caso PHP).

8.3.5. Envío periódico de datos

Otra consideración muy importante es cómo correrá el agente en cada máquina de forma que envíe datos de forma periódica y sin necesidad de interacción del usuario una vez que éste lo instala.

Hay varias formas de resolver este problema, como por ejemplo estas opciones:

- **Event loop** infinito: es una modalidad muy extendida y usada por herramientas como nodejs o algunos servidores webs. Se trata de tener un bucle infinito que vaya realizando operaciones en un intervalo de tiempo determinado, por ejemplo "durmierdo" el proceso durante ese intervalo (sleep). Además, esta modalidad ha de ser combinada con un lanzamiento en segundo plano del agente, ya que de lo contrario el usuario perdería el control de su terminal al estar ejecutando un script infinito.
- **Cron job**: es una herramienta que permite en sistemas linux realizar acciones en un momento determinado, permitiendo definir una gran combinatoria de opciones: día, día de la semana, hora, minuto, etc. Sin embargo, no permite

granularidad por segundo y, en este caso, se quiere enviar información del estado de la máquina cada 30 segundos.

- **Servicio de sistema operativo**: combinado con un event loop, es sin duda la mejor opción, ya que se delega la responsabilidad de monitorización del script al sistema operativo y es éste quien lo relanza en casos de error o salida inesperada. También se encarga de lanzarlo al inicio de sesión, tras un reinicio o apagado y encendido de la máquina. De esta forma, el usuario, una vez instalado el agente, se puede olvidar de él y es su sistema operativo el responsable de su monitorización y gestión.

Sin duda, esta última es la mejor opción, así que se decide crear un servicio de sistema operativo además del propio agente, el cual se desarrolla con un event loop.

9. Implementación y pruebas

9.1 Consideraciones generales

9.1.1. Estructura de repositorios

Al tratarse de tres componentes separados y claramente diferenciados, se decide crear tres repositorios de código diferentes en vez de uno solo. Esto permite mayor agilidad y evita tener un repositorio monolítico.

Los repositorios son:

- information-collector/
- web-admin/
- agent/

Antes de empezar la implementación, se decide usar un sistema de versionado de código, **Git** en concreto, de forma que se vaya manteniendo un histórico de cambios y permite recuperar versiones anteriores en caso de problemas. Por lo que en cada uno de los repositorios hay que inicializar Git, mediante el comando *git init*.

9.1.2. Gestión de dependencias

Los tres repositorios de código usan un gestor de dependencias, Composer en el caso de los proyectos de PHP (information-collector y web-admin) y go get en caso del agente en Golang.

9.1.3. Virtualización, aprovisionamiento y automatismo

En cuanto a los proyectos de PHP, debido a que ambos necesitan un web server y una base de datos, se comienza por la creación de una máquina virtual y su aprovisionamiento mediante Ansible.

9.2. Information Collector

9.2.1. Estructura de directorios

```
$ tree -L 2 -d -I vendor
.
├── bin
├── provisioning
│   ├── ansible
│   ├── ssh
│   └── ssl
├── resources
│   └── config
├── spec
│   └── Collector
├── src
│   └── Collector
└── web
```

Figura 14.

- *./bin*: contiene archivos ejecutables, como por ejemplo la herramienta para lanzar los tests unitarios: bin/phpspec.
- *./provisioning*: todo lo relacionado con aprovisionamiento y automatismo va en esta carpeta.
- *./resources*: contiene configuraciones.
- *./spec*: es la carpeta de los tests (basados en especificaciones).
- *./src*: contiene todo el código de la aplicación.

9.2.2. Definición de la máquina virtual

```
VAGRANTFILE_API_VERSION = "2"

Vagrant.configure(VAGRANTFILE_API_VERSION) do |config|
    config.vm.box = "ubuntu/trusty64"
    config.vm.network "forwarded_port", guest: 80, host: 8080
    config.vm.network "forwarded_port", guest: 443, host: 443
    config.vm.network "private_network", ip: "192.168.50.101"
    config.vm.synced_folder "./", "/var/www/api.pfc.aramirez.es/current", owner: "www-data", 
group: "www-data"
    config.vm.synced_folder ".", "/vagrant", disabled: true

    config.vm.provider "virtualbox" do |vb|
        vb.customize ["modifyvm", :id, "--memory", "512"]
    end
```

```
    config.vm.define "local" do |local|
      config.vm.provision "ansible" do |ansible|
          ansible.playbook = "provisioning/ansible/site.yml"
          ansible.inventory_path = "provisioning/hosts"
      end

      config.vm.provision "shell", inline: "service apache2 start", run: "always"
    end
end
```

Figura 15.

En el fichero Vagrantfile, como se puede observar en la figura 15, se define el uso de una máquina virtual, con sistema operativo Ubuntu, con IP 192.168.50.101, con puertos redireccionados (para el correcto uso de Apache), también se sincroniza monta toda la carpeta del repositorio en la ruta donde Apache va a buscar el virtual host "/var/www/api.pfc.aramirez.es/current". Por último se define Ansible como sistema de aprovisionamiento y se inicia Apache.

9.2.3. Controlador frontal, enrutador y controlador

Al usar Silex como micro framework web, el flujo de de ejecución desde que llega una petición hasta que se ejecuta el código correspondiente es el siguiente:

- Lo primero en ejecutarse es el controlador frontal (único punto de entrada), situado en *./web/index.php*. Este, ejecuta la siguiente acción:

```
<?php

require_once __DIR__.'/../vendor/autoload.php';

$app = require_once __DIR__ . '/../src/api.php';
require_once __DIR__ . '/../resources/config/prod.php';

$app->run();
```

Figura 16.

Que, cómo se puede observar, incluye tres ficheros diferentes y finalmente lanza la aplicación. El primer fichero, es la definición de carga automática de clases de

Composer (el gestor de dependencias). El segundo, es la definición de rutas y por último, la configuración específica para entorno de prod.

- Si miramos el fichero ./src/api.php, esto es lo que se puede ver:

```
$app = new Silex\Application();
$app->register(new Silex\Provider\RoutingServiceProvider());

$app->get('/', 'Collector\Infrastructure\Controller::indexAction')->bind('home');
$app->get('/systems',
'Collector\Infrastructure\Controller::listSystemsAction')->bind('systems');
$app->get('/systems/{id}',
'Collector\Infrastructure\Controller::showSystemAction')->bind('systems_by_id');
$app->post('/systems', 'Collector\Infrastructure\Controller::createSystemAction');
$app->delete('/systems/{id}',
'Collector\Infrastructure\Controller::deleteSystemAction');
$app->put('/systems/{id}',
'Collector\Infrastructure\Controller::replaceSystemAction');
$app->get('/systems/{id}/healths',
'Collector\Infrastructure\Controller::listHealthsAction')->bind('healths');
$app->post('/systems/{id}/healths',
'Collector\Infrastructure\Controller::createHealthAction');

return $app;
```

Figura 16.

Esto no es más que, instanciar el framework Silex y luego definir todos los pares de URL -> Controlador a usar. Como se puede observar en la figura 16, el único controlador que existe es Collector\Infraestructure\Controller.

9.2.4. Ports and Adapters y testing unitario

Para ver esta separación de capas entre core de aplicación y capa de infraestructura, hay que ir hasta la carpeta *./src/Collector/*, donde se podrán observar lo siguiente:

```
$ tree -L 3 src/Collector/
src/Collector/
├── Core
│   ├── HealthCheck
│   │   ├── Collection.php
│   │   ├── Repository.php
│   │   └── Service.php
│   ├── HealthCheck.php
│   ├── Measurements
│   │   ├── CpuLoad.php
│   │   ├── UsedHdd.php
│   │   └── UsedRam.php
│   ├── Measure.php
│   ├── System
```

```
│       │       ├──── Collection.php
│       │       ├──── Id.php
│       │       ├──── Repository.php
│       │       └──── Service.php
│       └──── System.php
└──── Infrastructure
        ├──── Controller.php
        ├──── HealthCheckInMemory.php
        ├──── HealthCheckMongodb.php
        ├──── SystemInMemory.php
        └──── SystemMongodb.php
```

Figura 17.

Como se puede observar en la figura 17, hay dos claramente diferenciadas carpetas "core" e "infraestructura", la primera contendrá entidades (el core de la aplicación), repositorios (interfaces que definirán cómo interactuar con la colección de entidades - los **Ports**) y servicios (el punto de entrada que el controlador tendrá que usar para interactuar con el core).

Por otro lado, en la carpeta "infrastructure", se encuentran todos los **Adapters**, que son implementaciones concretas de las interfaces definidas en core.

Un ejemplo concreto:

- **Collector\Core\System** sería una entidad, y tendría las propiedades id, hostname, cpus, ram, hdd y operating_system.
- **Collector\Core\System\Repository** es una interfaz, que define operaciones sobre una colección de entidades System, como por ejemplo add, findById o delete.
- **Collector\Core\System\Service** es el punto de entrada, define todos los casos de uso posibles contra System. Este servicio recibe una implementación concreta del Repository (InMemory o MongoDb por ejemplo), pero **depende de la interfaz Repository**, no de esa implementación concreta.

- **Collector\Infrastructure\SystemMongodb** es una implementación concreta de la interfaz **Collector\Core\System\Repository**, y es el encargado de interactuar con **MongoDb**.

Por otro lado, como el core de la aplicación consta de clases PHP básicas, sin dependencias de MongoDb, Silex o cualquier otro framework, servicio externo o dependencia, los tests son muy sencillos de hacer, se ejecutan de forma muy rápida y describen perfectamente qué hace la aplicación.

Se pueden lanzar todos los tests, con formato especificación, para visualizar claramente qué hace la aplicación:

```
$ bin/phpspec run --format=pretty

      Collector\Core\HealthCheck\Collection

  10  ✔ should return zero when collection is empty (155ms)
  16  ✔ should return the number of elements (103ms)

      Collector\Core\HealthCheck\Service

  17  ✔ adds a new helthcheck
  25  ✔ should get healthchecks collection
  30  ✔ should contains all systems in collection
  39  ✔ should retrieve healthchecks collection
  44  ✔ should retrieve a system health

      Collector\Core\HealthCheck

  15  ✔ should expose its id
  20  ✔ should expose created at
  25  ✔ should expose cpu load
  30  ✔ should expose cpu load as float always
  36  ✔ should expose used ram
  41  ✔ should expose used ram as float always
  47  ✔ should expose used hdd
  52  ✔ should expose used hdd as float always

      Collector\Core\System\Collection

  10  ✔ should return zero when collection is empty
  16  ✔ should return the number of elements

      Collector\Core\System\Id

  11  ✔ should be comparable
```

```
  18  ✔ should not be null

      Collector\Core\System\Service

  17  ✔ should register new system
  23  ✔ should retrieve system by id
  30  ✔ should return null if does not exist system with given id
  35  ✔ should replace a whole system by new one
  45  ✔ should create new system if replace did not match any system
  52  ✔ should throw an error if old system and new one have different ids
  61  ✔ should delete system by id
  69  ✔ should throw an error if tries to delete unexistant system
  74  ✔ should get system collection
  79  ✔ should contains all systems in collection

      Collector\Core\System

  23  ✔ should expose its id
  28  ✔ should expose hostname
  33  ✔ should expose ip
  38  ✔ should expose empty ip if not valid
  52  ✔ should expose number of cpus
  57  ✔ should expose ram capacity
  62  ✔ should expose hdd capacity
  67  ✔ should expose operating system
  72  ✔ should be comparable
  87  ✔ should populate created date


7 specs
39 examples (39 passed)
414ms
```

Figura 18.

9.2.5. HTTPS

Tal y como se había definido, para garantizar seguridad en este punto, se usa https en vez de http, para ello hay varios pasos a realizar:

- Creación de un certificado autofirmado.
- Configurar Apache para que use ese certificado

9.2.5.1. Creación de un certificado autofirmado

Consta de varios pasos, primero hay que crear una autoridad certificadora propia, que permita validar certificados, luego hay que crear el certificado de servidor y posteriormente firmarlo con la entidad certificadora creada anteriormente.

Para el desarrollo de esta parte, se eligió usar openssl como herramienta, ya que permite llevar a cabo la totalidad de tareas necesitadas, es una herramienta que viene instalada en distribuciones linux y hay mucha documentación en internet.

Estos pasos descritos anteriormente desembocan en dos ficheros para el servidor: un certificado y una clave, y en un fichero para la entidad certificadora, que además de permitir firmar el certificado del servidor, permitirá a los clientes que hagan peticiones http, validar el certificado del servidor (al tratarse de un certificado autofirmado y no firmado por una entidad autorizada, este paso es necesario).

9.2.5.2. Configuración de Apache

Para configurar Apache de forma que acepte solo peticiones bajo https, se usa mod_ssl y hay que asegurar que la siguiente configuración está presente en el fichero de configuración del vhost de Apache:

```
<VirtualHost *:443>

    SSLEngine on
    SSLCompression off
    SSLCertificateFile /etc/apache2/ssl/server.crt
    SSLCertificateKeyFile /etc/apache2/ssl/server.key
```

Figura 18.

9.3. Web Admin

Muy similar en cuanto a estructura de directorios y definición de máquina virtual, que el recolector de información, el problema más interesante a comentar es el uso del certificado CA para la validación del certificado SSL del servidor a la hora de hacer peticiones http al mismo.

Como ya se definió previamente, estas peticiones http se realizan mediante la librería Guzzlehttp, como se puede leer en la documentación de esta librería, validar certificados con un certificado propio es tan sencillo como añadir la siguiente configuración cada vez que se hace una request http.

```
'verify' => '/path/to/ca.cert.pem'
```

Figura 19.

Otro aspecto técnico interesante a comentar es el uso del plugin silex-simpleuser para toda la gestión de usuarios. Mediante este plugin, se puede gestionar de forma sencilla registro y login de usuarios, recuperar contraseña y editar perfil. Además, se ha extendido el plugin para añadir una funcionalidad no existente, que es la posibilidad de que el administrador pueda otorgar privilegios de administrador a otros usuarios, de forma sencilla, mediante un solo botón en el listado de usuarios.

9.4. Agente

Como ya se había definido, el agente se desarrolla mediante el lenguaje de programación Golang, que permite compilar a varias plataformas y hace realmente sencillo la comunicación con el sistema para la recopilación de datos.

9.4.1. Estructura de directorios

```
$ tree -L 2
.
├── README.md
├── installers
│   ├── centos
│   ├── debian
│   └── mac
├── main.go
└── system
    ├── ca.cert.pem.go
    ├── healthcheck.go
    └── system.go
```

Figura 20.

Como se puede observar, la estructura de directorios es muy sencilla, en este caso hay un fichero inicial “main.go”, una carpeta system, donde se ubican el resto de ficheros fuente y una carpeta installers, dividida por tipos de instalador (para OS X y para distribuciones de Linux basadas en RHEL o Debian).

9.4.2. Aspectors técnicos

Lo primero que merece la pena destacar es la forma de realizar el event loop para enviar la información periódica al servidor, para ello se muestra el punto de entrada al agente, el archivo main.go:

```
package main

import (
    "github.com/aramirez-es/agent/system"
    "time"
)

func main() {
    system.SendSystemInformation()

    sendSystemInformationInterval := time.NewTicker(24 * time.Hour)
    healthCheckInterval := time.NewTicker(30 * time.Second)

    for {
        select {
            case <- sendSystemInformationInterval.C:
                system.SendSystemInformation()
            case <- healthCheckInterval.C:
                system.SendHealthCheck()
        }
    }
}
```

Figura 21.

La función **main** será la ejecutada por Go, es el punto de entrada (por definición Go busca la función main dentro del paquete main y la lanza).

En esta función, se puede observar como lo primero que se hace es realizar un envío de información al servidor mediante la llamada *system.SendSystemInformation()*, que es una función que está en la carpeta *system* en el fichero *system.go*. Acto seguido, y aquí lo interesante, se declaran dos variables que contendrán un *Ticker* (una construcción nativa de Go), encargado de enviar una señal cada cierto intervalo de tiempo, en este caso, el primer ticker es cada 24 horas y el segundo cada 30 segundos. Lo siguiente es el loop infinito, mediante la instrucción for, que contiene una estructura de control, tipo Switch, que será llamada cada vez que una de las dos variables (tickers) anteriores reciban una señal, en cuyo caso se realizará la acción pertinente (realizar un envío de información al servidor o hacer un heartbeat al mismo).

Otro aspecto interesante del agente es cómo resolver el problema de identificación de una máquina cada vez que se lanza el agente. Para ello, se decide crear un identificador único (UUID) la primera vez que se ejecuta el agente y almacenarlo en un fichero en el directorio del usuario que lo lanza. Las veces posteriores, el agente tratará de leer este archivo y usar el mismo UUID. El identificador creado es completamente aleatorio, se podría haber usado la dirección MAC del ordenador como semilla, pero, a día de hoy lo normal es que los ordenadores tengan más de una tarjeta de red, además de las ficticias declaradas por las máquinas virtuales, así que descarté esa opción.

En el caso del agente, realiza dos peticiones al servidor (recolector de información) por lo que, al igual que la web, ha de resolver el problema de la validación del certificado autofirmado, para ello, a la hora de realizar una petición http, se necesita hacer lo siguiente:

```
pool := x509.NewCertPool()
pool.AppendCertsFromPEM(caCertificate)

tr := &http.Transport{
   TLSClientConfig:    &tls.Config{RootCAs: pool},
   DisableCompression: true,
}

client := &http.Client{Transport: tr}
```

Figura 22.

Donde la variable caCertificate contiene el certificado de la entidad autorizada para la validación, se puede comprobar en el fichero ./system/ca.cert.pem.go

```
var caCertificate = []byte(`
-----BEGIN CERTIFICATE-----
MIIGoTCCBImgAwIBAgIJAMSnsbYqw4h7MA0GCSqGSIb3DQEBCwUAMIGRMQswCQYD
...
```

Figura 23.

9.4.3. Servicio de sistema operativo

En este aspecto, finalmente se da soporte a sistemas basados en systemd (distribución CentOS por ejemplo), initd (por ejemplo Ubuntu) y finalmente OS X.

9.4.3.1. Systemd

```
[Unit]
Description=Agent that send system information besides periodical heart beat to central information collector
After=network.target

[Service]
Type=simple
PIDFile=/var/run/pfcaramirezagent.pid
ExecStart=/usr/bin/pfcaramirezagent
Restart=on-failure

[Install]
WantedBy=multi-user.target
```

Figura 24.

Los puntos más destacados aquí son:

- After=network.target, que permite que este servicio se inicie cuando hay red.
- ExecStart=/usr/bin/pfcaramirezagent, que es el binario a ejecutar (agente compilado)
- Restart=on-failure, que determina que se ha de reiniciar el agente cada vez que salga de forma inesperada (por ejemplo cuando falla el envío de una petición http, cuyo caso causa que el agente mande un exit(-1).

9.4.3.2. Initd

```
#!/bin/bash

NAME=pfcaramirezagent
DESC="Agent that send system information besides periodical heart beat to central information collector."
PIDFILE="/var/run/${NAME}.pid"
```

```
LOGFILE="/var/log/${NAME}.log"

DAEMON=/usr/bin/pfcaramirezagent
DAEMON_OPTS=

START_OPTS="--start  --background  --make-pidfile  --pidfile  ${PIDFILE}  --exec  ${DAEMON}
${DAEMON_OPTS}"
STOP_OPTS="--stop --pidfile ${PIDFILE}"

test -x $DAEMON || exit 0

set -e

start() {
   echo -n "Starting ${DESC}: "
   start-stop-daemon $START_OPTS $LOGFILE
   echo "$NAME."
}

stop() {
   echo -n "Stopping $DESC: "
   start-stop-daemon $STOP_OPTS
   echo "$NAME."
   rm -f $PIDFILE
}

status() {
   if [ -f $PIDFILE ] ; then
           echo "Running"
   else
           echo "Stopped"
   fi
}

restart() {
   echo -n "Restarting $DESC: "
   start-stop-daemon $STOP_OPTS
   sleep 1
   start-stop-daemon $START_OPTS >> $LOGFILE
   echo "$NAME."
}


case "$1" in
 start)
       start
       ;;
 stop)
       stop
       ;;
 status)
       status
       ;;
 restart|force-reload)
       restart
       ;;
 *)
       N=/etc/init.d/$NAME
       echo "Usage: $N {start|stop|restart|force-reload}" >&2
```

```
        exit 1
esac

exit 0
```

Figura 25.

9.4.3.3. OS X

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
    <key>Label</key>
    <string>org.pfc.aramirez.agent</string>
    <key>ProgramArguments</key>
    <array>
        <string>/usr/local/bin/pfcaramirezagent</string>
    </array>
    <key>KeepAlive</key>
    <dict>
        <key>SuccessfulExit</key>
        <false/>
        <key>NetworkState</key>
        <true/>
    </dict>
    <key>RunAtLoad</key>
    <true/>
    <key>StandardOutPath</key>
    <string>/var/log/org.pfc.aramirez/agent.log</string>
    <key>StandardErrorPath</key>
    <string>/var/log/org.pfc.aramirez/agent.log</string>
</dict>
</plist>
```

Figura 26.

En el caso de OS X, tal y como se puede apreciar en la figura 26, la definición de un servicio de sistema operativo se hace mediante un fichero XML, de donde cabe destacar lo siguiente:

- **ProgramArguments** es un array donde la primera posición define el archivo ejecutable
- **KeepAlive** se usa para indicar al sistema operativo cuándo el modo de supervisión a aplicar, en este caso, el sistema operativo debería mantener el proceso abierto siempre que este sea abortado (no tenga una salida correcta) y exista conexión de red.

10. Implantación, puesta en producción y resultados

10.1. Puesta en producción de los proyectos PHP

La puesta en producción tanto de la api de recolección de información como de la web de gestión del grid se ha hecho hacia un servidor dedicado de uso personal, con sistema operativo Ubuntu Server.

Se ha usado el dominio personal aramirez.es para alojar ambos proyectos, bajo las urls:

- https://api.pfc.aramirez.es
- http://admin.pfc.aramirez.es

Ambas accesibles desde cualquier ordenador conectado a internet.

Para la puesta en producción, debido a que todo el aprovisionamiento y configuración se ha automatizado con Ansible, solo ha sido necesario añadir un paso más (el de deploy) y crear un script de coordinación (para acortar el comando de terminal a lanzar)

```
#!/bin/sh

PATH_DEPLOY_FROM=$(pwd)

ansible-playbook \
    -i provisioning/production_hosts \
    --private-key=./provisioning/ssh/id_rsa \
    -u deploy \
    --ask-sudo-pass \
    -e ansistrano_deploy_from="$PATH_DEPLOY_FROM/" \
    provisioning/ansible/site.yml
```

Figura 26.

Como se puede observar en la figura 26, este script lanza el comando ansible-playbook, al que le pasa como argumentos (por orden):

- El inventario de de servidores a los que subir el código.
- La clave privada de acceso al server.
- El usuario con el que conectarse al servidor.
- Confirmación para que pregunte la contraseña de usuario root.
- Path donde se encuentra el código a subir.
- El playbook (fichero YAML) de Ansible a ejecutar.

Para la realización del copiado remoto de ficheros se ha usado el módulo de Ansible "ansistrano/deploy".

10.2. Compilación y empaquetado del Agente Go

En el caso del agente en go, el proceso es sustancialmente diferente a los otros dos repositorios. En este caso, la puesta en producción consiste en el compilado para las diferentes plataformas que se han decidido soportar (OS X y Linux) y un posterior empaquetado del fichero binario generado, junto con el servicio del sistema, de cara a facilitar la instalación a los usuarios.

10.2.1. Cross Compilation

Para realizar la compilación de binarios GO para múltiples plataformas, hace falta instalar Go con soporte para todas ellas, en Mac se puede hacer de la siguiente forma:

```
$ brew install go --cross-compile-all
```

Figura 27.

Luego hay que instalar una librería que permite hacer la compilación multiplataforma:

```
$ go get github.com/laher/goxc
$ goxc -t
```

Figura 28.

Y por último, llevar a cabo la compilación:

```
$ goxc -pv="0.10"
```

Figura 29.

10.2.2. Debian

Para sistemas operativos basados en Debian, se crea un paquete .deb, que contiene la declaración del servicio de sistema operativo (initd), el agente compilado (binario) y una serie de hooks de instalación. La estructura de directorios para crear el paquete .deb se puede ver en ***./installers/debian/deb/pfcaramirezagent-1.0.5***. En este caso, el binario se introduce en la carpeta ***usr/bin/*** con nombre ***pfcaramirezagent*** y el servicio en ***etc/init.d/*** con nombre ***pfcaramirezagent***, de esta forma, cuando se instale el .deb, estos archivos se colocarán en el lugar correspondiente.

Por otro lado, la carpeta DEBIAN contiene tanto el fichero de definición del paquete (control) como dos hooks que se ejecutarán después de realizar la instalación (que permitirá registrar el servicio y lanzarlo por primera vez) y antes de eliminar el agente (que permitirá parar el servicio previamente).

Por último, lo único necesario es lanzar el comando dpkg-deb --build para crear el paquete debian. Se puede ver más información en la bibliografía.

10.2.3. RHEL (Red Hat Enterprise Linux)

Para sistemas operativos basados en RHEL, se crea un paquete ***rpm***, que contiene la declaración del servicio de sistema operativo (para systemd), el agente compilado (binario) y un fichero de especificación del RPM. En el caso de los rpm, hay algo más de configuración a realizar que con respecto a lo que se hacía en debian.

En este caso, se ha de crear un fichero tar.gz que contiene tanto el binario como el servicio, y este archivo es ubicado en la carpeta *SOURCES*. Por otro lado, el fichero de especificación de RPM está alojado en el directorio *SPECS*.

El proceso de generación del RPM también es algo más complejo que el DEB, se puede ver más información en la bibliografía.

10.2.4. OS X

Para sistema operativo OS X, lo que se ha realizado es la creación de un .PKG, que permite instalar de forma visual el agente (y el servicio de sistema operativo asociado). Similar a debian en cuanto a estructuración del paquete, en ese caso el fichero de definición del servicio irá en ***sources/Library/LaunchDaemons/*** con nombre ***org.pfc.aramirez.agent.plist*** y en ***sources/usr/local/bin/*** el binario, llamado ***pfcaramirezagent***.

En el caso de OS X, al tratarse de un instalador visual, hay que definir una serie de recursos HTML que serán cargados en el instalador. Estos, se ubican en ***resources*** y son los siguientes ficheros welcome.html, license.html y conclusion.html.

Finalmente, para el empaquetado del PKG, son necesario dos pasos, primero la creación de un pkg con el comando pkgbuild y acto seguido el producto final con el comando productbuild. En este caso, he creado un fichero llamado build.sh, que lanza ambos, se puede ver en *installers/mac*.

10.3. Instalación del agente

Las instrucciones de instalación del agente se pueden encontrar de forma pública en http://admin.pfc.aramirez.es/downloads no obstante, se resume en los siguientes puntos.

10.3.1. Debian

```
$ curl -s -O
http://admin.pfc.aramirez.es/agents/linux/x86_64/pfcaramirezagent-1.0.5-1.x86_64.deb && \
sudo dpkg -i pfcaramirezagent-1.0.5-1.x86_64.deb && \
rm -f pfcaramirezagent-1.0.5-1.x86_64.deb
```

Figura 30.

10.3.2. RHEL

```
$ sudo yum install
http://admin.pfc.aramirez.es/agents/linux/x86_64/pfcaramirezagent-1.0.5-1.x86_64.rpm
```

Figura 31.

10.3.3. OS X

En este caso es simplemente descargar el fichero y ejecutarlo, de este modo aparecerá un instalador visual.

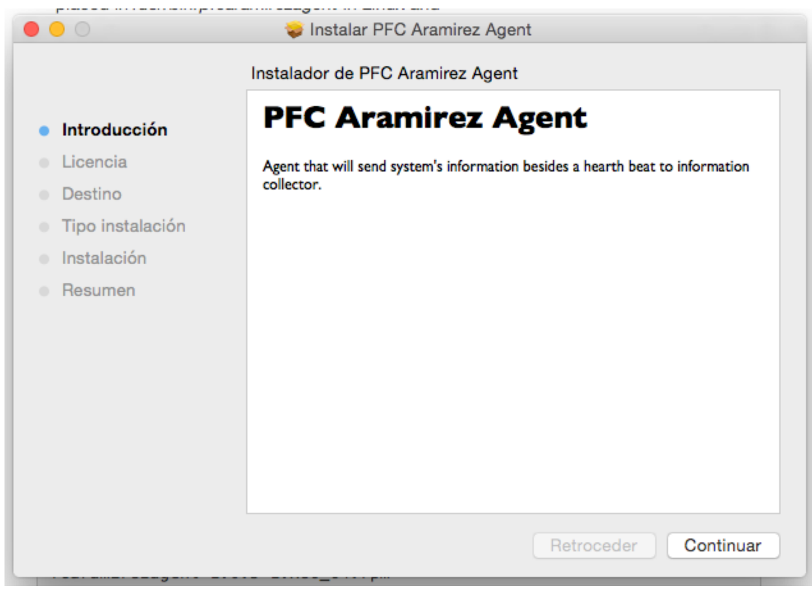


Figura 32.

10.4. Resultados

A nivel funcional, la web de administración se puede ver adjunto como anexo, tanto el manual de usuario como manual de administrador.

Haciendo balance entre requisitos funcionales a los cuales me comprometí en el proyecto y los llevados a cabo, podemos hacer el siguiente resumen:

Requisito funcional/no funcional	Actual
La aplicación web debe permitir listar estos recursos y su disponibilidad.	En la homepage de la web, el dashboard muestra el listado completo de máquinas y su disponibilidad mediante un icono (verde o rojo) y cuánto hace que se supo algo sobre esa máquina.
Por cada máquina agregada al grid se ha de poder visualizar CPU, Memoria y Disco de que disponen.	El dasboard muestra estado, hostname, ip, número de cpus, cantidad de ram y de disco.
La aplicación web debe poder eliminar recursos.	El usuario administrador, puede borrar de la lista aquellas máquinas que están caídas (hacen más de 30 minutos que no envía un hearthbeat)
Los usuarios podrán instalar el agente en sus ordenadores para comunicarse con el grid.	En la página de descargas de la web, los usuarios tienen la información de cómo descargar e instalar el agente, disponible para diferentes sistemas operativos, mediante un paquete rpm, deb o pkg. Además, la instalación incluye un extra no contemplado en los requisitos y es que el agente se instala como servicio del sistema operativo.
El agente enviará datos periódicamente a la aplicación web gestora.	Cada 24 horas envía información básica del sistema y la actualiza en el servidor en caso de que esta haya cambiado desde la última actualización. Concretamente envía la IP, el hostname, el identificador de la máquina, el número de cpus, la capacidad de ram y disco y el sistema operativo. Además, cada 30 segundos se envía un check para notificar de que la máquina está viva. Este check además incluye información del estado de la máquina, como es el load average, ram y disco usados.
El acceso a la	Hay gestión completa de usuarios, éstos se pueden

aplicación web será restringido a usuarios mediante identificador y contraseña.	registrar y loguear para acceder a la aplicación, de lo contrario solo tienen acceso limitado (a la página de descarga). Además se provee con un usuario administrador, con privilegios para listar a todos los usuarios y otorgar privilegios de administrador. Por último, también se incluye la posibilidad de recuperar la contraseña en caso de pérdida.
La comunicación entre los componentes ha de estar cifrada y viajar bajo HTTP + TLS (https)	Efectivamente, tanto el agente como la web se comunican con el recolector de información bajo https de forma única y exclusiva, ya que éste no acepta peticiones http. Se ha creado una autoridad certificadora y un certificado autofirmado para el servidor de producción.

11. Conclusiones

Finalmente, me gustaría comentar que durante el desarrollo de este proyecto, he podido aprender muchísimas cosas que no había podido aprender hasta ahora, destacando el trabajo realizado en el lenguaje de programación Golang, que me ha permitido introducirme en el mismo, aprender la sintaxis básica, resolver un problema real y poder compilar para múltiples plataformas.

También he aprendido cómo empaquetar y distribuir paquetes a diferentes sistemas operativos, como han sido OS X, Debian y RHEL. Además de poder crear un servicio de sistema operativo para todos ellos.

Por otro lado, también he podido refrescar conocimientos aprendidos sobre seguridad, aplicado a la creación de certificados SSL autofirmados mediante la herramienta openssl.

He podido además, entregar todos los requisitos funcionales con los que me había comprometido y he dejado la puerta abierta para poder incrementar éstos, mediante la explotación de más datos que se están enviando actualmente pero no sacando partido.

Por último, creo que ha sido un ejercicio muy interesante sobre sistemas distribuidos, pero pare ser completamente honesto, me hubiera gustado poder invertir más tiempo para probar algoritmos de computación distribuida como gossip.

12. Trabajo futuro

A lo largo de estos meses trabajando con este proyecto, han salido nuevas ideas a medida que iba conociendo más en profundidad el problema que se estaba resolviendo.

Como puntos principales de potencial trabajo futuro, me gustaría destacar los siguiente:

- Mejor explotación de datos: a día de hoy, se están almacenando más datos de los realmente mostrados en la web. Por ejemplo, cada 30 segundos cada máquina envía su load average, su ram y disco usados. Con estos datos se podría hacer un panel de estado de cada máquina, con gráficos para que fuera más visual.
- Evitar tener un único punto de fallo (Single Point of Failure): este es el caso del recolector de información, que recibe inputs de tantas máquinas haya registradas y enviando información (cabe recordar que pueden enviar hearbeats cada 30 segundos). Así pues, si este número de máquinas creciera, sería muy difícil mantener la efectividad del recolector de información. Para solventar este problema, se podría llegar a usar un protocolo de membership y failure detection más eficiente como Gossip, diseñado para sistemas distribuidos.
- Información en tiempo real: el panel web muestra información sobre el estado de cada máquina, indicando cuándo fue el último heartbeat que se recibió. Para refrescar esa información, hay que recargar la página por completo, así que una mejora importante podría ser hacer un pulling ajax para conocer el estado en real-time
- Servidor de integración continua y build automático: el proceso de compilación cross-platform, empaquetado para debian, RHEL y OS X es manual a día de hoy, y hay que lanzarlo por cada nueva versión del agente. Una mejora muy relevante sería tener un servidor de integración continua (como Jenkins) que permitiera automatizar todo este proceso y que se lanzara de forma automática por cada commit a Git.

- Soporte del agente para Windows: ya soportadas máquinas linux (diferentes distribuciones) y OS X, lo interesante sería dar soporte a máquinas Windows también. Go permite compilar el mismo código fuente para Windows, pero habría que realizar un trabajo de adaptación del código del agente para que fuera plenamente funcional en esta plataforma.
- Sharding de la tabla de datos “Healthchecks”: dada la cantidad de información que esta tabla recibe y que crecerá sin control, una estrategia de escalado sería usar sharding horizontal, que quiere decir guardar particiones de datos (mediante una función de distribución aplicada al ID por ejemplo) en diferentes máquinas, en vez de una única máquina con todos los datos. De esta forma, se consigue que cada máquina tenga una porción menor de datos, por lo que las queries e indexaciones serían potencialmente más rápidas.

13. Bibliografía

A continuación se detalla el listado de fuentes de información utilizado para el desarrollo del proyecto:

[1] **Kimsufi**, página web oficial del proveedor de servidores dedicados. http://www.kimsufi.com/es/

[2] **Certificados SSL**, blog que habla a cerca de cómo crear certificados SSL autofirmados. https://jamielinux.com/articles/2013/08/create-and-sign-ssl-certificates-certificate-authority/

[3] **Compilación multiplataforma**, blog donde se detalla cómo compilar un programa escrito en go para que funcione en varias plataformas diferentes. http://spf13.com/post/cross-compiling-go/

[4] **Creación de archivos pkg**, blog donde se explica cómo crear paquetes para OS x. http://vincent.bernat.im/en/blog/2013-autoconf-osx-packaging.html

[5] **Servicios en OS X**, web oficial de Apple, donde explican cómo crear servicios de sistema operativo. https://developer.apple.com/library/mac/documentation/Darwin/Reference/ManPages/man5/launchd.plist.5.html

[6] **Servicios KeepAlive en OS X**, documentación de cómo hacer que servicios de OS X se mantengan vivos. https://github.com/tjluoma/launchd-keepalive

[7] **Creación de servicios initd**, foro de discusión donde se debate sobre la forma de crear servicios para el sistema init. http://stackoverflow.com/questions/22336075/linux-process-into-a-service

[8] **Servicios initd arrancar procesos**, web donde se explica cómo crear servicios initd y se muestran ejemplos de cómo arrancar estos servicios. http://www.nginxtips.com/create-linux-daemon-service/

[9] **Crear paquetes debian**, se explica en detalle cómo crear paquetes debian sencillos. http://ubuntuforums.org/showthread.php?t=910717

[10] **Creación de servicios systemd**, web donde se explica cómo crear servicios para el sistema systemd. https://scottlinux.com/2014/12/08/how-to-create-a-systemd-service-in-linux-centos-7/

[11] **RHEL**, web oficial de Red Hat Enterprise Linux donde se explica en detalle cómo funcionan los servicios systemd. https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/7/html/System_Administrators_Guide/chap-Managing_Services_with_systemd.html

[12] **Crear rpm**, debate en la web Stackoverflow donde se comenta la forma de crear paquetes rpm. http://stackoverflow.com/questions/880227/what-is-the-minimum-i-have-to-do-to-create-an-rpm-file

[13] **Hooks en rpms**, web donde se explica cómo aplicar hooks a los rpm. http://stackoverflow.com/questions/8854882/why-does-service-stop-after-rpm-is-updated