UNIVERSITAT POLITÈCNICA DE VALÈNCIA

CAMPUS D'ALCOI

DSIC

DEPARTAMENTO DE SISTEMAS INFORMÁTICOS Y COMPUTACIÓN

Diseño de una plataforma de despliegue e instalación automática de imágenes

---

Niobium – The Network Boot Manager

Cristian Moncho Ivorra
Tutor: Javier Esparza Peidro
Grado en Ingeniería Informática

Agradecimientos

Antes de continuar, me gustaría agradecer a todas aquellas personas –amigos, familiares y profesores– que me han ayudado y apoyado tanto a lo largo de la carrera como durante el transcurso del proyecto, y que me han enseñado todo lo necesario para poder estar hoy aquí escribiendo estas líneas.

En especial a Irene, que ha estado animándome y ayudando en todo lo que estaba a su alcance y que ha sido crucial para poder seguir hacia delante en todo momento. Y a Javier Esparza, mi tutor del TFG, por motivarme a escoger este proyecto y ayudarme a asentar las bases de lo que a continuación presentaré.

Por último, y no menos importante, agradecer a la comunidad *open source* por la cantidad de herramientas –de gran calidad, además– que brinda y que han hecho posible la realización de este proyecto. Espero, con él, poder devolverle el favor.

¡Muchas gracias a todos!

Resumen

La gran mayoría de instituciones y entornos de trabajo requieren el uso múltiples máquinas que, además, suelen compartir unas prestaciones y software similares y en las que realizar tareas de mantenimiento puede suponer una ardua tarea.

Con este proyecto se pretende ofrecer una forma sencilla, rápida y, sobre todo, centralizada de gestionar los sistemas operativos de esas máquinas encargándose de toda la configuración necesaria para la puesta en marcha. Esta configuración se lleva a cabo desde un panel de administración web, adaptable a todo tipo de dispositivos, que ofrece un entorno amigable e intuitivo desde el que poder crear, distribuir y configurar las imágenes de los sistemas operativos para su posterior carga en las máquinas cliente a través de la red.

Palabras clave: panel, administración, web, carga, arranque, red, configuración, sistema operativo, SO

Índice de contenido

1. Introducción

1.1. Motivación

A la hora de configurar múltiples máquinas que, además, comparten *hardware* similar, es recomendable el uso de imágenes o instaladores personalizados que puedan ser desplegados en pocos pasos, pues agiliza en gran medida las tareas de administración.

Actualmente existen numerosas herramientas para ayudar a la hora de generar estas imágenes para sistemas **GNU/Linux**, ya sean para el arranque de sistemas operativos funcionales o para la creación de instaladores. La complejidad de su manejo se ha reducido a lo largo de los años, pero siguen requiriendo la configuración de diversos servicios para su correcto despliegue.

La creación de una herramienta que realice por si misma todo este proceso y facilite las labores de administración requeridas, tendría gran acogida en la comunidad además de ser gran utilidad, siendo este segundo punto el principal incentivo del proyecto.

1.2. Estudio del mercado

Como se ha mencionado previamente, en el mercado actual se encuentran numerosas herramientas para llevar a cabo esta labor. **FAI**[1], **live-build**[2] o **LTSP**[3], son algunas de las más conocidas. Todas ellas ofrecen su propio entorno en el que poder configurar y crear imágenes **GNU/Linux**, así como extensos manuales de ayuda *(ver anexos)* sobre cómo utilizarlas y qué configuraciones adicionales requieren.

El mayor problema que nos encontramos se presenta en estas configuraciones que, en ocasiones, requieren de conocimientos técnicos. Si se pretende utilizar las imágenes resultantes en un entorno distribuido en que toda la carga se realice en red, se precisa la configuración de múltiples servicios como, por ejemplo, la de los servidores **TFTP**, **DHCP** y/o **NFS** y la instalación o ejecución de cargadores de arranque en las máquinas cliente (como **iPXE**[4]).

Desafortunadamente, no existe en el mercado ninguna aplicación o herramienta que facilite o automatice este conjunto de tareas a realizar de una forma sencilla y sin requerir demasiados conocimientos en la materia. Es justo esta carencia la que se pretende suplir con la realización de este proyecto, de forma que la puesta a punto de una infraestructura de ordenadores de una empresa, institución, centro educativo o centro de datos –como por ejemplo los orientados a *cloud computing* o almacenamiento–, entre otros, pueda ser realizada cómodamente y sin mayores complicaciones.

1.3. Objetivos del proyecto

Con este proyecto se busca facilitar tanto el proceso de configuración de todos los servicios implicados como la creación y distribución de imágenes utilizando las herramientas mencionadas, todo desde una única aplicación. Es decir, automatizar todas las operaciones de generación de imágenes así como toda la configuración implicada en su distribución, reduciendo considerablemente la complejidad de esta tarea y la cantidad de pasos a seguir.

Se examinaran en profundidad los diversos métodos de configuración de cada una de las herramientas citadas y se extraerán todos los elementos comunes para poder automatizar el proceso de una forma genérica.

Los objetivos propuestos inicialmente para el siguiente proyecto consisten en diseñar una plataforma capaz de:

- Mantener un inventario acerca de la configuración física de un conjunto de máquinas en un centro de datos
- Gestionar un conjunto de imágenes de sistemas operativos, o repositorios de sistemas operativos, que permitan un despliegue o instalación remota posterior
- Gestionar políticas de despliegue e instalación de imágenes en las máquinas del centro de datos
- Monitorizar y tomar acciones sobre las máquinas del centro de datos

Aunque, adicionalmente, se pretenderá alcanzar también los siguientes:

- Ofrecer una forma fácil de gestionar la creación y distribución de las imágenes sin requerir elevados conocimientos técnicos del proceso.
- La posibilidad de agrupar estas imágenes y de permitir especificar qué usuarios tendrán acceso a cada una de ellas.
- Facilidad en la divulgación de configuraciones.
- Crear distintos niveles de usuario y dar la posibilidad de restringir el acceso a parámetros de configuración según el tipo.
- Permitir el uso de la aplicación de forma remota e independientemente del dispositivo utilizado para acceder a ella.

1.4. Estructura del resto de la memoria

A continuación se listarán y comentarán cada uno de los apartados en que está dividida la memoria para una presentación correcta y ordenada:

2. **Planificación**. Listará con detalle la planificación seguida para la correcta realización del proyecto y se comentarán los resultados de la misma.
3. **Análisis o definición del problema**. Mostrará todos los requisitos que se deberán tener en cuenta en la resolución del proyecto.
4. **Diseño o solución del problema**. Especificará todos los detalles de la implementación y hará una división de esta según las distintas capas.
5. **Resultado o tour por la aplicación**. Se adjuntarán capturas del resultado final y se explicará su manejo y puesta a punto.
6. **Conclusiones y trabajos futuros**. Presentará una lista con todas las impresiones y conclusiones extraídas durante la realización del proyecto, así como todas las posibles mejoras o futuras implementaciones.
7. **Bibliografía**. Todas las referencias a páginas oficiales de productos aparecerán aquí listadas.
8. **Glosario**. Para la correcta comprensión de los términos de la memoria, se adjuntará una tabla con las aclaraciones y significados más representativos.
9. **Anexos**. Manuales, documentación, referencias y todo tipo de documentos se añadirán a la sección de anexos para no entorpecer la lectura de la memoria.

Para facilitar aun más la comprensión por parte del lector se adjuntarán diagramas **UML** tanto en el análisis como en el diseño de la aplicación y se recomendará examinar el glosario antes de proceder a la lectura de la memoria, para una correcta interpretación de los términos de los textos.

2. Planificación

Tal y como cita la normativa, se dispone de un mínimo de 300 horas para la realización del proyecto. Se simulará un horario laboral de lunes a viernes con un horario de 6 horas diarias en el mes de julio (23 días) y 8 horas en agosto (21 días), sumando así un total de 306 horas y dejando cierto margen (18 días) para poder reaccionar a tiempo en caso de tener que reajustar los límites de tiempo.

2.1. Planificación inicial

La planificación se dividirá en 4 partes –o hitos–, y la distribución de tareas y tiempo se realizará siguiendo la siguiente propuesta (descontando los días extra):

1. Plan de trabajo: 5 días.
2. Análisis y diseño: 10 días.
3. Desarrollo y pruebas: 22 días.
4. Memoria y entrega: 7 días.

2.1.1. Plan de trabajo

Tarea	Duración (días)	F. Inicio	F. Fin
Propuesta de proyecto	1	01/07/14	01/07/14
Análisis de viabilidad	3	02/07/14	04/07/14
Selección y aprobación	1	04/07/14	04/07/14
Preparación	1	04/07/14	04/07/14

Tabla 1. Plan de trabajo

2.1.2. Análisis y diseño

Tarea	Duración (días)	F. Inicio	F. Fin
Estudio de los requisitos	2	07/07/14	08/07/14
Generar documentación	5	07/07/14	11/07/14
Diseñar diagramas	3	11/07/14	15/07/14

Tabla 2. Análisis y diseño

2.1.3. Desarrollo y pruebas

Tarea	Duración (días)	F. Inicio	F. Fin
Preparación del entorno	1	15/07/14	15/07/14
Diseño gráfico	3	15/07/14	17/07/14
Creación de plantillas	2	17/07/14	18/07/14
Panel de administración común	5	21/07/14	25/07/14
Sistema de acceso de usuarios	1	25/07/14	25/07/14
Panel de administración de la aplicación	1	28/07/14	28/07/14
Generador de scripts de inicio	2	28/07/14	29/07/14
Generador de imágenes	5	29/07/14	04/08/14
Generador de configuraciones	4	04/08/14	07/08/14
Panel de eventos y visor de actividad	1	08/08/14	08/08/14
Crear instalador	1	11/08/14	11/08/14
Fase de pruebas	3	12/08/14	14/08/14
Revisión y correcciones	5	15/08/14	20/08/14
Cierre	1	25/08/14	25/08/14

Tabla 3. Desarrollo y pruebas

2.1.4. Memoria y entrega

Tarea	Duración (días)	F. Inicio	F. Fin
Realización de la memoria	7	21/08/14	29/08/14
Entrega	1	01/09/14	01/09/14

Tabla 4. Memoria y entrega

2.2. Diagrama de Gantt

A continuación se mostrará la planificación anteriormente mencionada en forma de diagrama Gantt para que sea más visual y permita hacerse una idea de cuál será la progresión del proyecto a lo largo del tiempo.

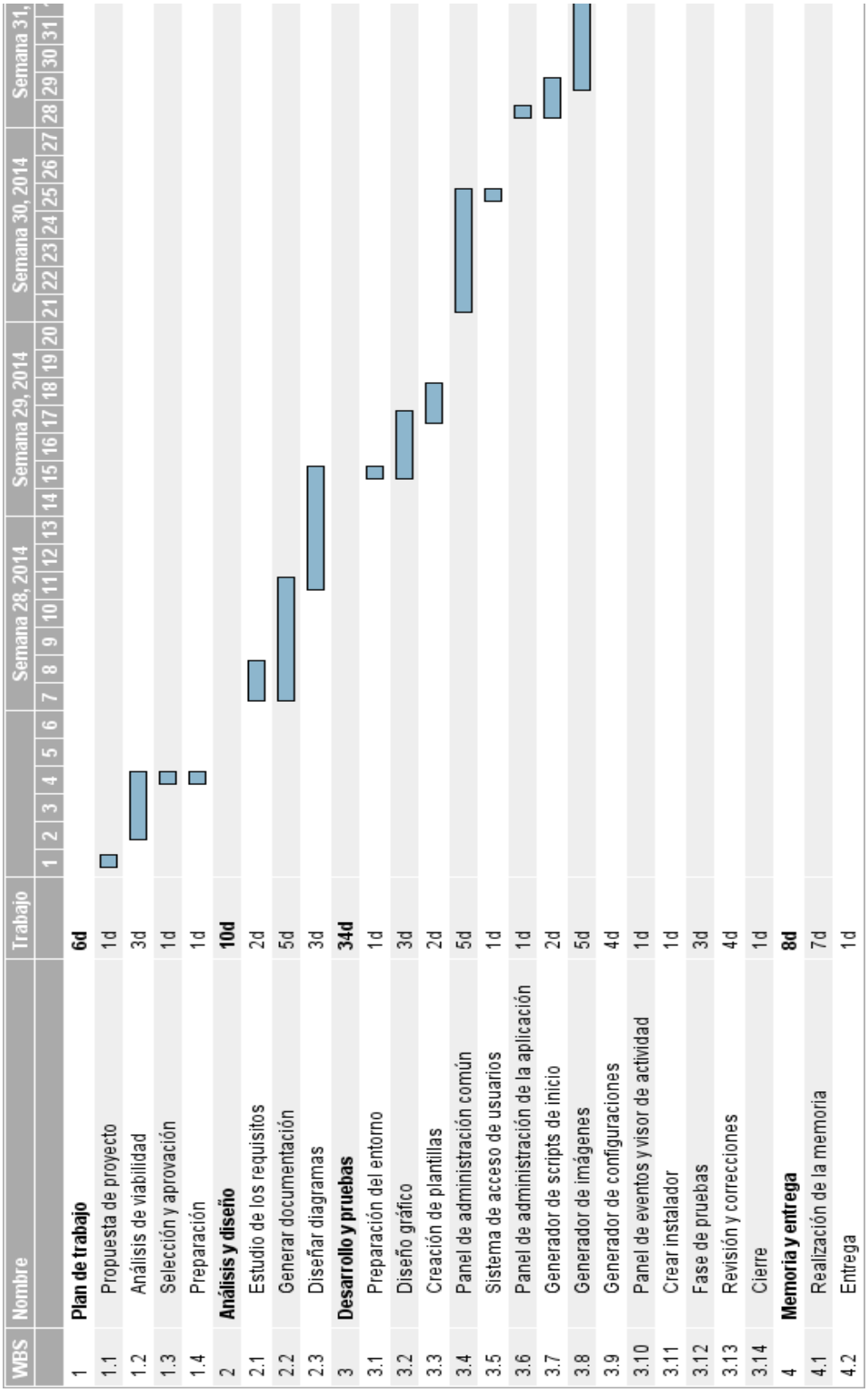

Figura 1. Diagrama de Gantt (parte1).

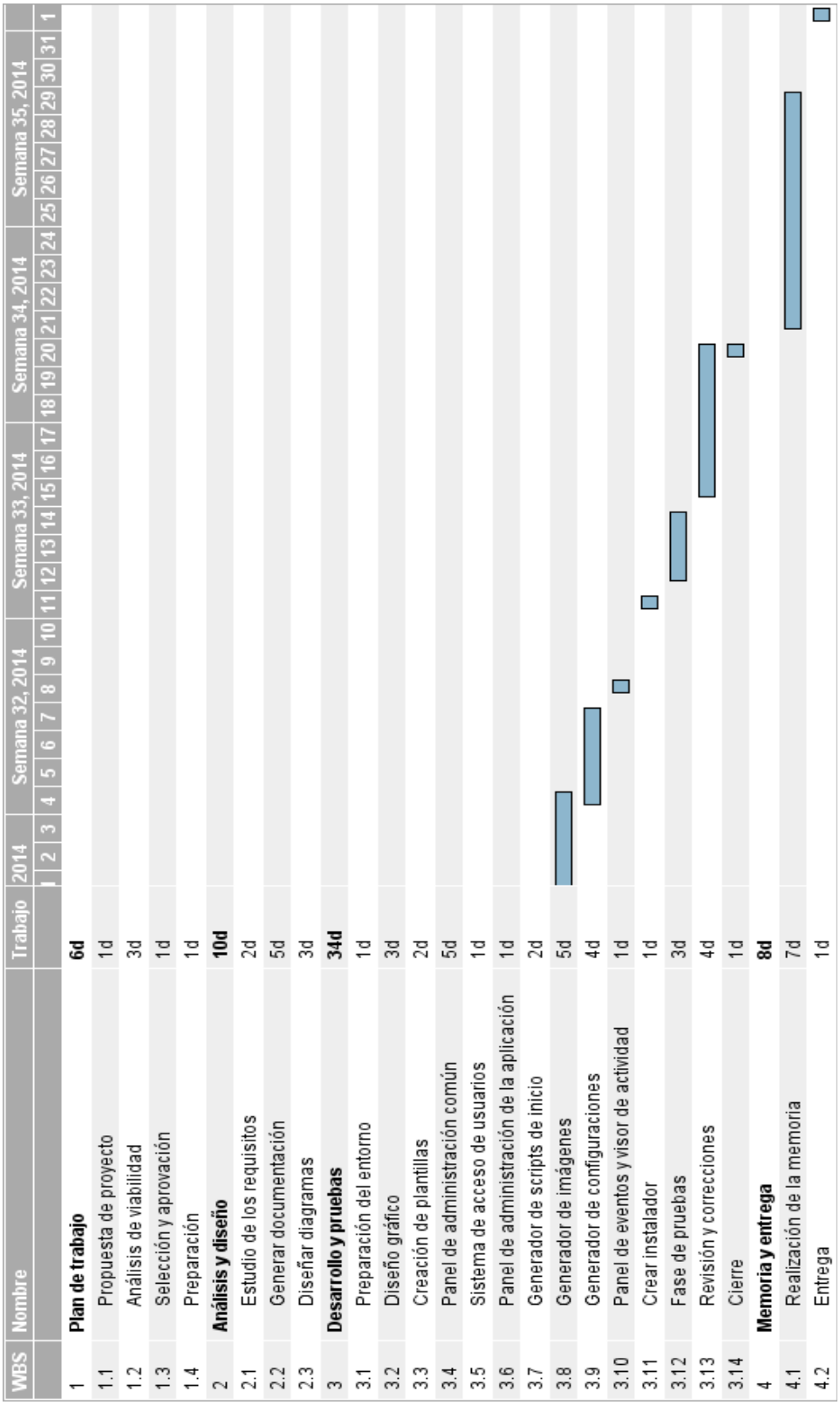

Figura 2. Diagrama de Gantt (parte 2).

2.3. Planificación final

Hay que tener en cuenta que el proyecto no ha sido únicamente implementar funcionalidades, también ha requerido el aprendizaje y dominio de todas las plataformas, lenguajes y programas que se han utilizado en su elaboración. Aun teniendo esto en cuenta, en casi la totalidad de fases del proyecto se ha seguido la planificación propuesta, salvo en las fases 1 –los principales puntos ya estaban previamente concretados– y 3 –se necesitó ampliar algunos plazos–.

En la fase 1 se realizaron las siguientes tareas previamente al plazo de la planificación, por temas de organización:

- Propuesta del proyecto: 18/12/13.
- Análisis de viabilidad: un total de 3 días comprendidos entre el 31/01/14 y el 05/03/14.
- Selección y aprobación: 04/06/14.

En el caso de la fase 3 el principal motivo fue la ampliación del tiempo requerido por las siguientes tareas:

- Preparación del entorno: un total de 3 días (no contiguos), debido a fallos ajenos en el sistema informático.
- Generador de imágenes: un total de 8 días, fue necesario volver a analizar ese apartado ya que la primera propuesta no fue correcta.

Dado que la fase 2 se inició antes de lo previsto –a fecha de 01/07/14–, se dispuso de más días para la fase 3, que resultaron ser de gran utilidad.

3. Análisis o definición del problema

Procedemos a analizar y listar con detalle los requisitos –funcionales y no funcionales– de la aplicación.

3.1. Requisitos funcionales

1. Todo servicio involucrado ha de poder ser configurado: servidor NFS, servidor DHCP, script de arranque iPXE, constructores de imágenes, parámetros de acceso, configuración del servidor del proyecto, etc.

1.1. Se ofrecerá una configuración por defecto para agilizar la puesta en marcha.

1.2. Debe ser fácilmente accesible y centralizado (figura 3).

1.3. Debe ofrecer tolerancia a errores.

1.3.1. Abstraerá el formato del archivo de configuración a una serie de parámetros configurables, para su posterior generación.

1.3.2. En caso de error se intentará corregir, o se volverá a la anterior configuración, o se notificará al usuario, según convenga.

1.4. El usuario debe ser capaz de modificar la mayor cantidad de parámetros de configuración –tanto internos como de servicios externos– desde la interfaz de la aplicación.

1.4.1. Si ocurre cualquier fallo durante la operación, el proceso debe ser reversible.

1.4.2. Se debe notificar al usuario del estado de la operación, mostrando toda la información relevante del resultado.

1.4.3. Se mostrará, en las configuraciones que lo requieran, una ventana de confirmación con todas las modificaciones a realizar.

1.4.4. Cualquier modificación ha de aplicarse al instante y «en caliente». Evitando, en la medida de lo posible, la necesidad de reiniciar procesos.

2. El usuario podrá comprobar el estado de las conexiones al servidor por parte de los clientes.

2.1. La comprobación se ha de hacer en tiempo real.

2.2. Para facilitar la comprensión de los datos recibidos, se mostrarán en tablas y/o gráficas.

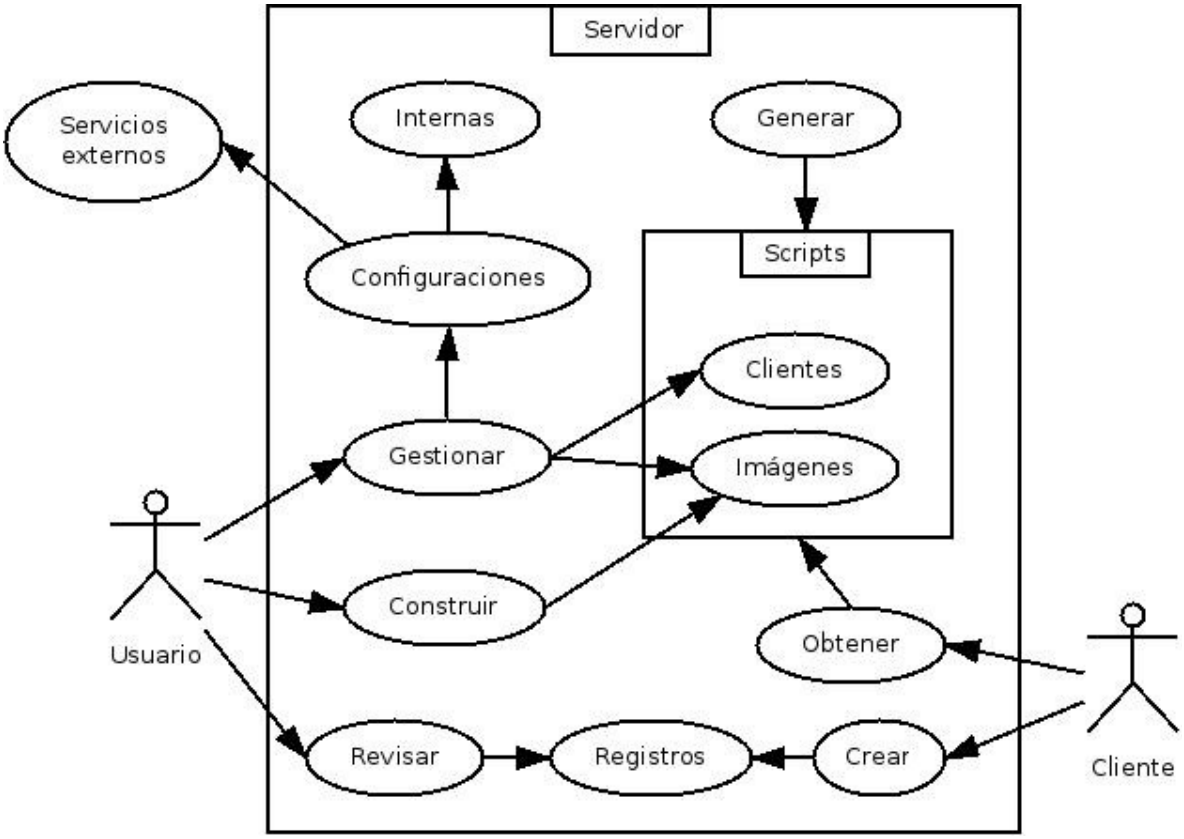

Figura 3. Diagrama de casos de uso.

2.2.1. Deben ser descriptivos y, en los casos en que se requiera, se permitirá interaccionar con ellos.

3. El usuario podrá generar nuevas imágenes en base al conector escogido.

3.1. El sistema devolverá, en tiempo real, cualquier registro o mensaje del proceso y lo presentará al usuario.

3.2. Se facilitará la opción de finalizar la operación en cualquier momento.

3.3. El usuario tendrá la posibilidad de conocer, en todo momento, el estado del proceso de construcción.

3.4. En caso de éxito en el proceso de construcción, el sistema deberá crear la configuración necesaria para su ejecución.

4. El sistema deberá evitar, en la medida de lo posible, la ejecución de funciones o programas que requieran de una elevación de permisos.

4.1. Se hallarán formas de realizar las configuración desde el usuario por defecto.

4.2. Cuando sea imposible ejecutar una función de este ámbito, se avisará al usuario y sera éste el que deberá de ejecutar los comandos requeridos.

4.2.1. Debido a que el uso puede variar entre sistemas, se ofrecerá un forma común o se listarán todas las posibles opciones.

5. La máquina cliente no tendrá ningún privilegio ni ninguna forma de acceder al panel de administración o modificar datos.

5.1. Sus accesos serán almacenados en el registro del servidor.

5.2. Únicamente podrá solicitar un *script* de carga (figura 3) y sólo podrá cargar aquellas imágenes que le hallan sido asignadas (figura 4).

5.3. En caso de no estar registrada, se creará un nuevo registro que identifique a la máquina almacenando todos los datos que sea posible (figura 4) y no se asignará a ningún grupo, no pudiendo acceder a ninguna imagen sin que el administrador la asigne manualmente..

5.4. En caso de no tener ninguna imagen asignada, se ofrecerá un *script* por defecto (figura 4).

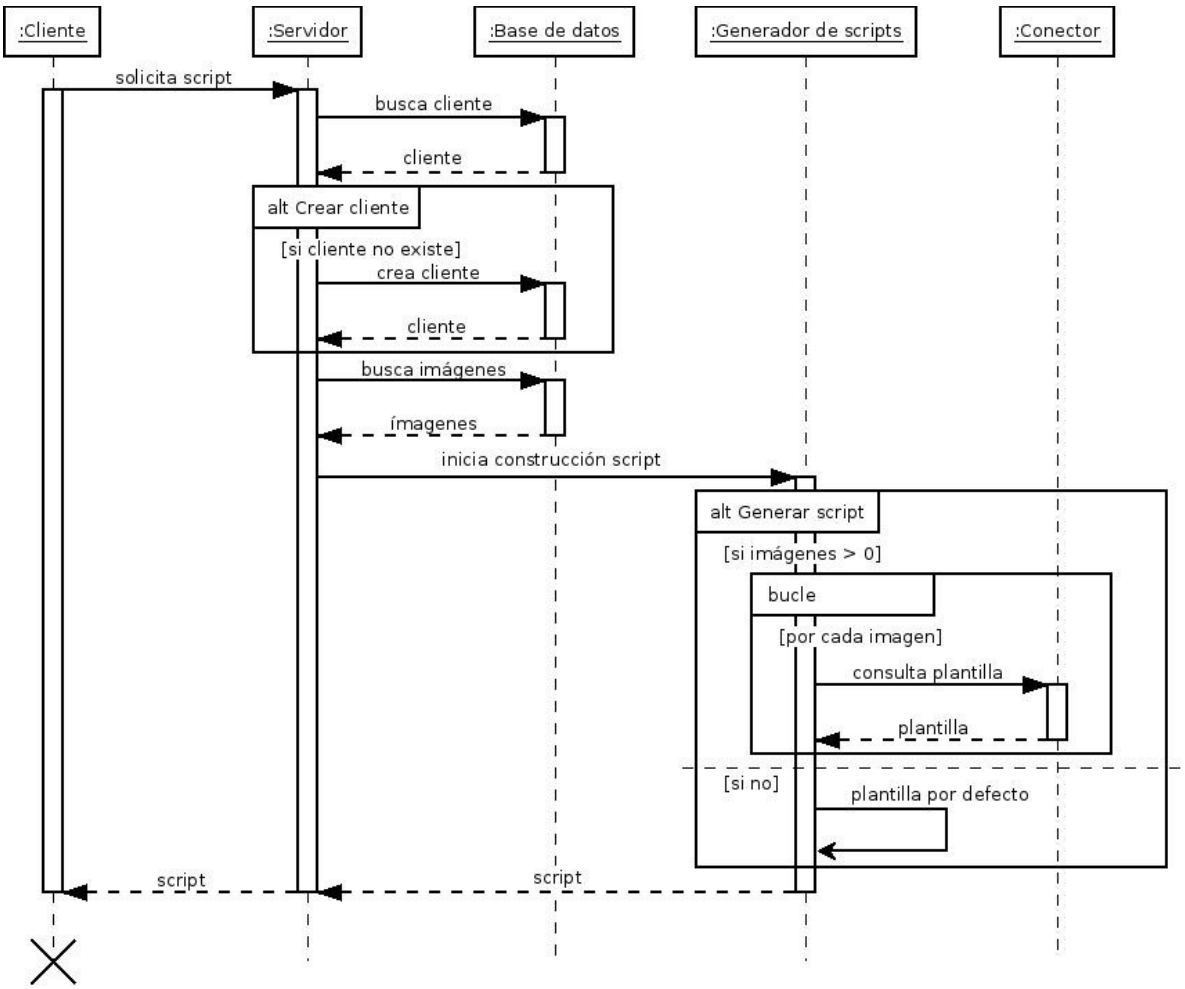

Figura 4. Solicitud de un script de arranque.

3.2. Requisitos no funcionales

1. El servidor deberá permanecer activo la totalidad del tiempo desde el momento en que se inicie, a menos que se detenga expresamente.

2. Deberá ser de fácil manejo e intuitivo. No requerirá conocimientos técnicos avanzados en la materia y deberá ofrecer una adaptación rápida al entorno y modo de trabajo.

3. El sistema deberá poder ejecutarse sobre el mayor número de plataformas posibles y deberá ser utilizable desde cualquier tipo de dispositivo que posea un navegador web.

4. La aplicación deberá ser creada con herramientas bien documentadas para facilitar el mantenimiento y posibilitar la futura ampliación de características.

5. Se minimizarán al máximo los costes del desarrollo.

4. Diseño o solución del problema

En el siguiente apartado comentaremos detalladamente cómo ha sido construida la aplicación. Se describirán los componentes involucrados y como están conectados entre si.

Al utilizar un *framework*, tanto la estructura del proyecto como la forma de funcionar están explicadas y comentadas en su documentación oficial *(ver anexos)*. No se profundizará en cómo funciona internamente más allá de lo estrictamente necesario, pero sí que se mencionará toda la parte lógica relacionada con el proyecto.

4.1. Arquitectura de la aplicación

Con tal de conseguir un software bien estructurado y que, además, ofrezca cierta flexibilidad a la hora de distribuir los componentes, se ha optado por un modelo bastante utilizado: la estructura a 3 capas. De este modo el usuario final puede variar la forma del despliegue, si así lo prefiere, aunque este es el recomendado:

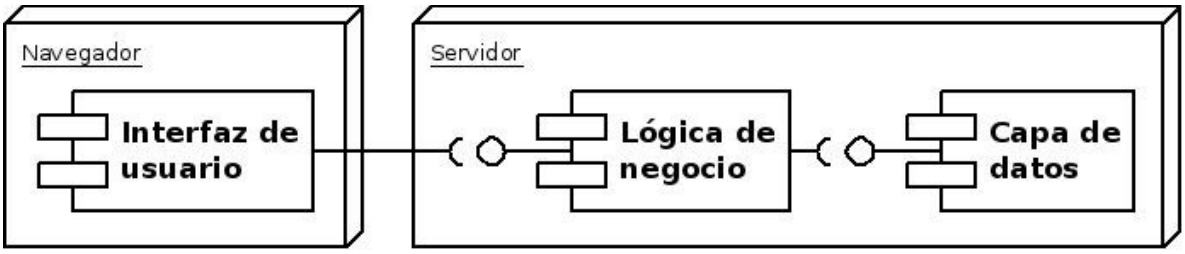

Figura 5. Diagrama de despliegue.

4.2. Tecnologías utilizadas

En los últimos años se ha experimentado un notable aumento en el interés de todo lo relacionado con las tecnologías web. Así pues, se ha optado por construir una solución en base a estas tecnologías utilizando las últimas versiones para asegurar un producto que no quede desfasado en poco tiempo. También se ha tenido en cuenta que las herramientas en cuestión estuvieran bien documentadas y extendidas en la comunidad de desarrolladores. Estas han sido las escogidas para el proyecto:

Capa	Tecnología	Explicación
Interfaz de usuario	HTML5[5] WebSockets JS (+jQuery[6])	La mayoría de dispositivos poseen navegadores web adaptados a estas tecnologías relativamente nuevas y en constante evolución. Esta elección nos asegura que el proyecto permanecerá actualizado por más tiempo.
	CSS3 (+Bootstrap[7])	La versión 3 de CSS ofrece numerosas ventajas respecto a las anteriores y, en conjunto con Bootstrap[7], agiliza mucho el proceso de estilizar la web a la par que la hace muy atractiva visualmente.
Lógica de negocio	Node.js[8] (+Sails.js[9])	Para poder construir aplicaciones en tiempo real hace falta una tecnología capaz de ello y Node.js[8] es una muy interesante plataforma de desarrollo orientado a aplicaciones en red. La combinación de este entorno y Sails.js[9], un *framework* orientado a la construcción de webs basadas en WebSockets, nos ofrece un entorno de trabajo eficiente y con una rápida velocidad de respuesta, además de ser independiente de la plataforma.
Capa de datos	Waterline	Es el ORM por defecto de Sails.js[9]. Nos permitirá trabajar con el modelo de datos sin tener que depender de ningún tipo de SGBD y que la elección de éste se realice por parte del usuario.

Tabla 5. Tecnologías utilizadas.

4.3. Capa 1: Interfaz de usuario

La aplicación gestiona las vistas en plantillas con formato **EJS**[10], que es precompilado por **Sails.js**[9] cada vez que el servidor es puesto en marcha –o cada vez que una plantilla es editada, en modo desarrollo– y servido inmediatamente al cliente. Además, el *framework* está completamente capacitado para producir aplicaciones multilenguaje, por lo que la divulgación de la aplicación en distintos idiomas únicamente requiere la traducción de un fichero que contiene todas las cadenas de texto.

En la parte del cliente, la combinación de estilos de **Bootstrap**[7] y el dinamismo que otorga **jQuery**[6] consiguen crear un entorno agradable e intuitivo para el usuario con tiempos de respuesta muy bajos. Se ha conseguido, además, crear una interfaz web que se ajusta completamente a la pantalla del dispositivo, consiguiendo así uno de los puntos principales.

El uso WebSockets también ha dotado a la web de una sensación de fluidez típica de aplicaciones de escritorio, pues una gran cantidad de cargas se llevan a cabo de forma

asíncrona sin requerir la carga completa de la página, disminuyendo, aun más, los posibles tiempos de carga y las latencias de tener que cargar la web en el usuario en cada petición.

En la sección 4 de la memoria se muestran capturas del resultado final de la web (figuras 9-18) y un ejemplo de cómo ésta se adapta a la pantalla de dispositivos más pequeños, redistribuyendo el mismo contenido pero de forma más óptima a lo largo del ancho total.

4.4. Capa 2: Lógica de negocio

La logina del servidor está divida en 2 ramas bien diferenciadas: **Controladores** y **Servicios** (figura 6). Cabe mencionar que existen más archivo y configuraciones en la parte del servidor, pero éstos tienen que ver con la infraestructura web y no tanto con el proyecto en si. De todas formas, en las últimas páginas se adjuntarán manuales para la configuración y manejo del framework **Sails.js**[9] *(ver anexos)*.

La rama de Controladores comprende todo lo relacionado con los accesos directos a la web. Cada petición –**POST**, **GET**, **PUT** y **DELETE**– ejecuta una función del controlador al que va dirigida, asignándole como parámetros tanto los indicados en la **URL** como aquellos que formen parte de la misma petición. A continuación se detallará el funcionamiento de cada uno de ellos:

- **ControllerBase** y todos aquellos controladores que heredan de éste. Ofrece todas las operaciones básicas de creación, edición, búsqueda y borrado de registros de la capa de datos. Las llamadas a estos controladores devuelven al usuario formularios de edición o listas de registros que, además, contienen cualquier mensaje enviado por el servidor (tanto de errores en el proceso como de éxito de la operación).
- **IndexController**. Es la primera página que se muestra al cargar la web y la única accesible también para usuarios sin acceso a la web o con acceso restringido. En ella se muestra cualquier error de acceso a áreas restringidas y los errores que puedan ocurrir en el formulario de acceso como, por ejemplo, una validación errónea por cuenta no existente o por credenciales erróneas.
- **DashboardController**. Muestra, desde una única página, toda la información relevante para el administrador del servidor: estado de los servicios secundarios, una gráfica y una tabla con los accesos de los clientes y acceso rápido a todas las áreas de configuración.
- **BuildController**. Se encarga de la ejecución y detención de procesos de construcción de imágenes. Muestra al usuario el estado del proceso, un registro de todos los mensajes que éste emita y ofrece la posibilidad de detenerlo en cualquier momento. Por la forma en que ejecuta el proceso, éste no se detendrá en caso de

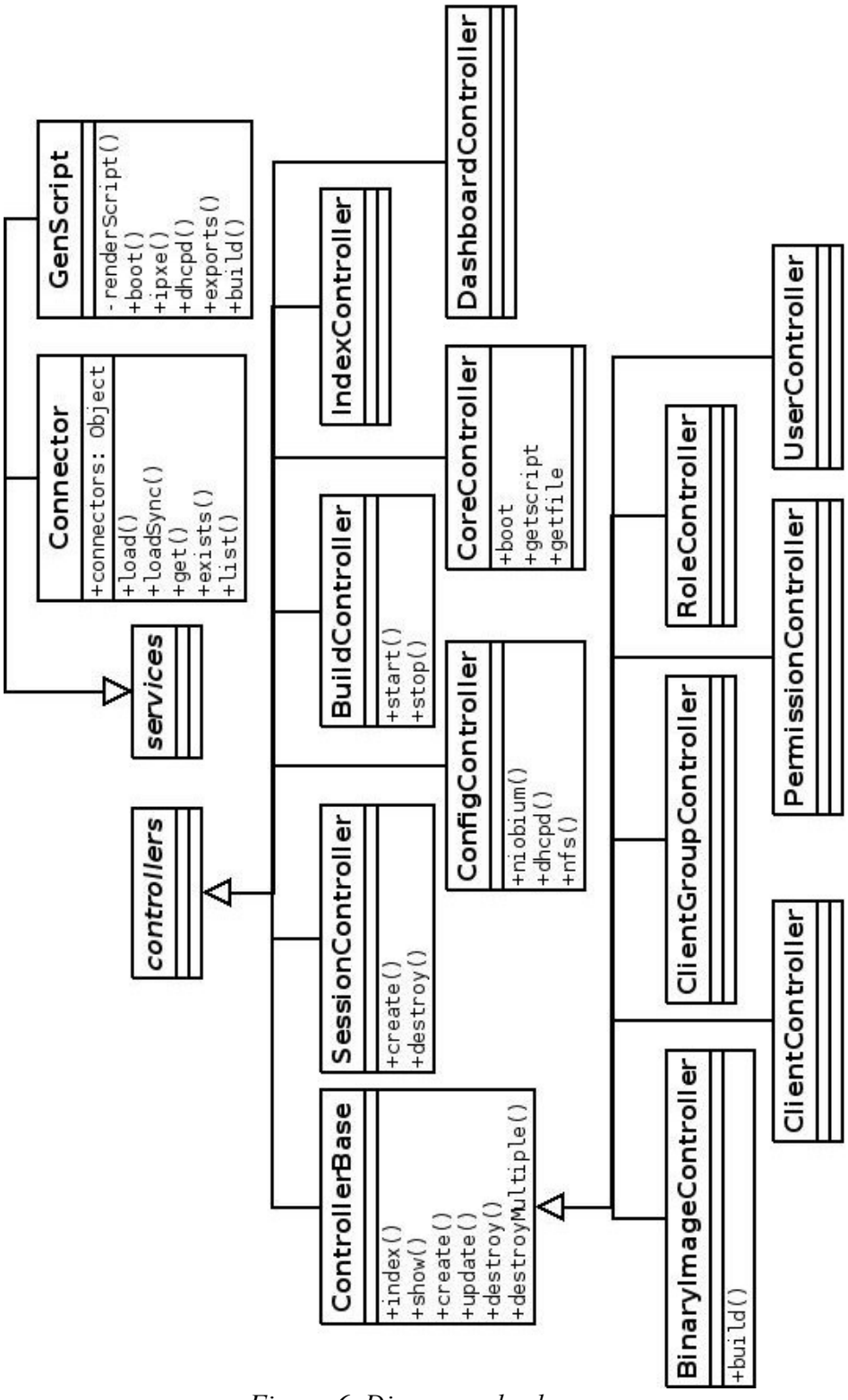

Figura 6. Diagrama de clases.

caída del servidor.

- **SessionController**. Gestiona y comprueba las credenciales tras completar el formulario de acceso. Crea una sesión persistente que caduca al cerrar el navegador o al cerrar la sesión manualmente.
- **ConfigController**. Permite cambiar parámetros de la configuración interna del servidor y gestionar las configuraciones de los servidores **DHCP** y **NFS**.
- **CoreController**. Es el único controlador al que tiene acceso la máquina cliente. Se encarga de ofrecer el *script* de arranque basándose en la dirección **MAC** de la máquina que realiza la petición.

La rama de Servicios contiene 2 elementos esenciales para el funcionamiento del proyecto:

- **Connector**: Se encarga de cargar todos los conectores creados que luego serán utilizados por **BinaryImageController** a la hora de iniciar la construcción de una nueva imagen (figura 7) y por GenScript en el momento de generar el *script* de inicio (figura 4). Un conector consiste en las siguientes partes:
  - Un archivo de configuración con los parámetros a utilizar en la generación del *script* **iPXE**[4] y los parámetros a utilizar en la construcción de una imagen.
  - Un archivo **Bash**[11] con todos los comandos necesarios para construir una imagen nueva. Utilizará la configuración del anterior archivo.
- **GenScript**: Transforma cualquier configuración en un formato inteligible por la aplicación destino mediante el uso de plantillas en formato **EJS**. Está capacitado para producir las configuraciones de los servidores **DHCP** y **NFS** y los *scripts* para **iPXE**.

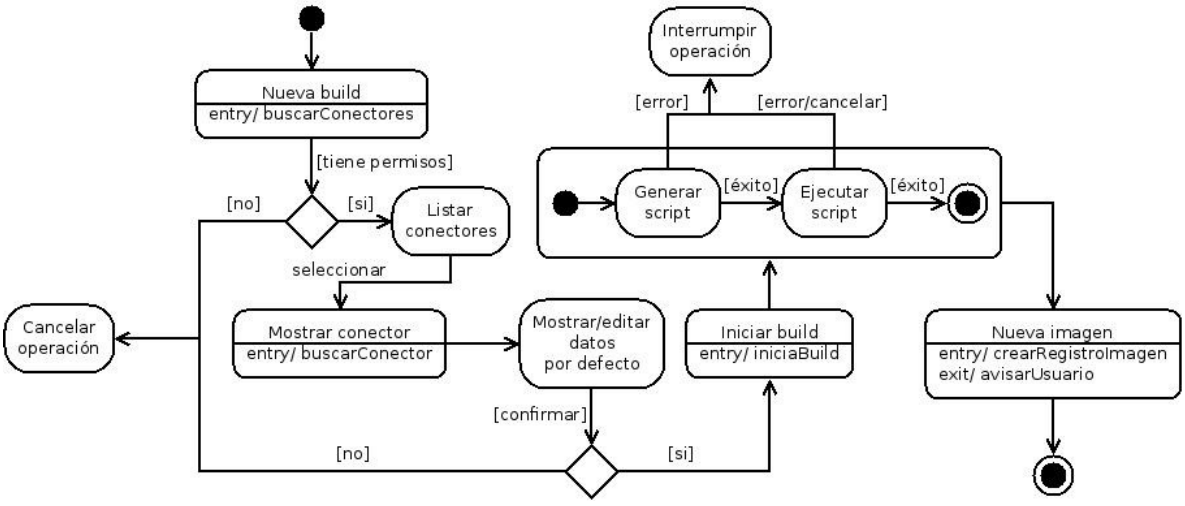

Figura 7. Generación de una nueva imagen

4.5. Capa 3: Capa de datos

El manejo de todo lo relacionado con la capa de datos, como se ha mencionado previamente, se realiza mediante el ORM **Waterline** *(ver anexos)*. Profundizaremos en el conjunto de modelos utilizado y las relaciones entre ellos (figura 8), en el que podemos diferenciar 3 grupos, el de usuarios –y su relación con roles y permisos–, el de la configuración interna del proyecto y el de la relación de imágenes con clientes y conectores.

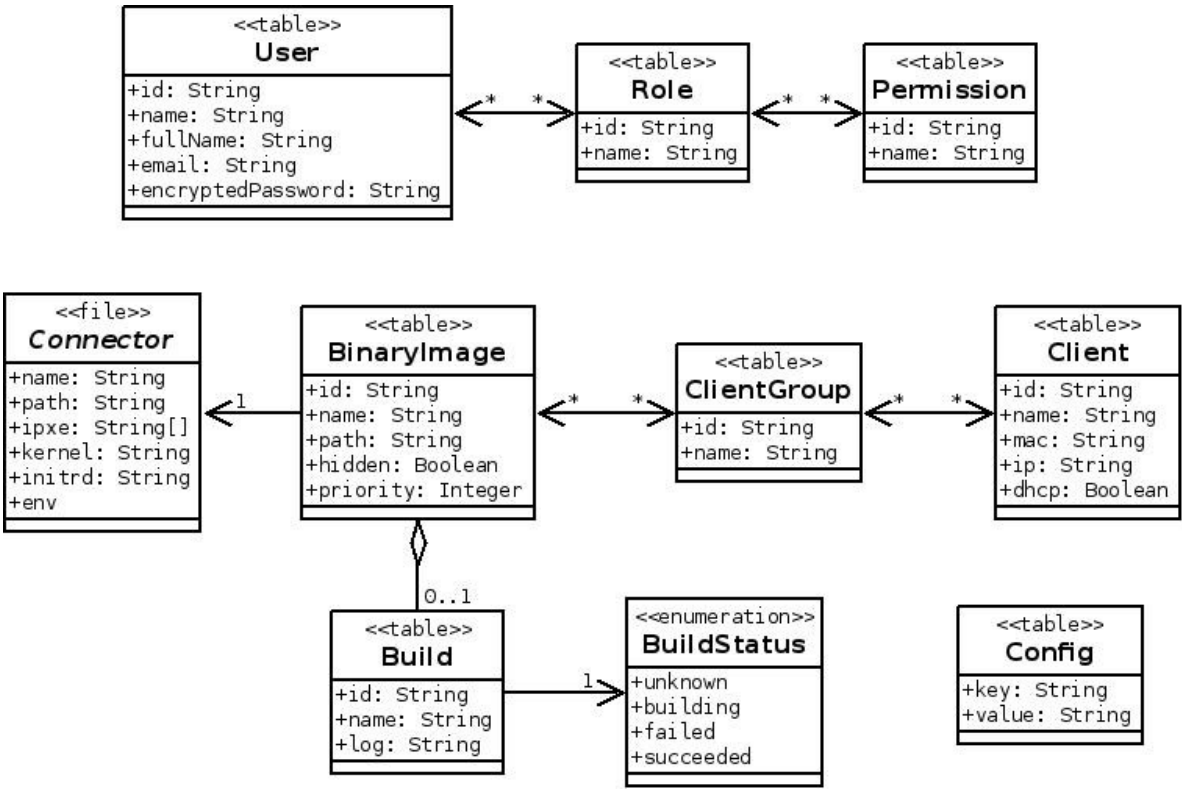

Figura 8. Diagrama de modelos.

- Usuarios
  - **User**. Contiene los datos de cada usuario, sus credenciales para poder acceder correctamente y cualquier cantidad de roles asociados.
  - **Role** y **Permission**. Mientras que los permisos son los que otorgan acceso a ciertas secciones de la web –incluso otorgan permiso para acceder a la web–, los roles otorgan un nombre a un conjunto de éstos y permiten que puedan ser asociados a los usuarios.
- Configuración. En **Config** se almacenan en forma de diccionario, o associative *array* –conjunto de pares clave-valor–, todos los parámetros de configuración

interna tales como la IP del servidor y el puerto a utilizar y otros relacionados con los *scripts* **iPXE**[4] y el servidor **DHCP**.

- Imágenes y máquinas cliente.
  - **BinaryImage**. Guarda los datos de cada imagen generada por el sistema –o añadida manualmente por el administrador– para su posterior uso. A cada imagen se le ha de asignar, obligatoriamente, un conector.
  - **Client**. Cada registro de la tabla **Client** equivale a una máquina cliente distinta y se diferencia entre ellos por el campo **mac**, que almacena la dirección **MAC** de la tarjeta de red que posean.
  - **ClientGroup**. Se encarga de asociar cualquier cantidad de máquinas cliente con cualquier cantidad de imágenes, de forma que en una misma red puedan coexistir máquinas con acceso a distintas imágenes.
  - **Build**. Permite almacenar de forma persistente todos los mensajes lanzados por el *script* de generación de nueva imagen. Almacena también otros datos como fecha de creación, fecha de última modificación y estado del proceso (**BuildStatus**). Inicia el proceso que se muestra en la figura 7.

La siguiente estructura nos permite mantener un sistema de usuarios con permisos asociados mediante roles para crear usuarios con acceso restringido a ciertas partes de la web. También nos permite saber si una imagen ha sido construida mediante el asistente de construcción de imágenes, así como el conector asociado para generar los *scripts* de arranque en **iPXE**[4]. Ofrece, además, la posibilidad de crear grupos de clientes y asociarles imágenes de forma que en una misma red se puedan tener distintas respuestas para según qué cliente.

5. Resultado o tour por la aplicación

A continuación procederemos a mostrar el aspecto final de la web resultante, sus pantallas más características y muestras del resultado final en la máquina cliente. Se mostrará cómo la página se adapta al ancho total del dispositivo consiguiendo un diseño web adaptable *(RWD)* y se comentarán los aspectos de configuración y pruebas más relevantes.

En el apartado **Despliegue y funcionamiento**, se especificará su puesta en marcha y cómo simular un entorno de pruebas si se quiere probar y no se tiene acceso a una red local de ordenadores.

5.1. Presentación

Existen más pantallas distintas a las mostradas, pero no se mostrarán dada su similitud con otras equivalentes (p.ej., todas las relacionadas con la administración de registros de la BBDD).

La primera página que se muestra nada más iniciar el servidor y abrir la web es la página de inicio (figura 9). En ella se muestra el formulario de acceso al panel de administración y cualquier error en el proceso será mostrado en forma de bloque informativo.

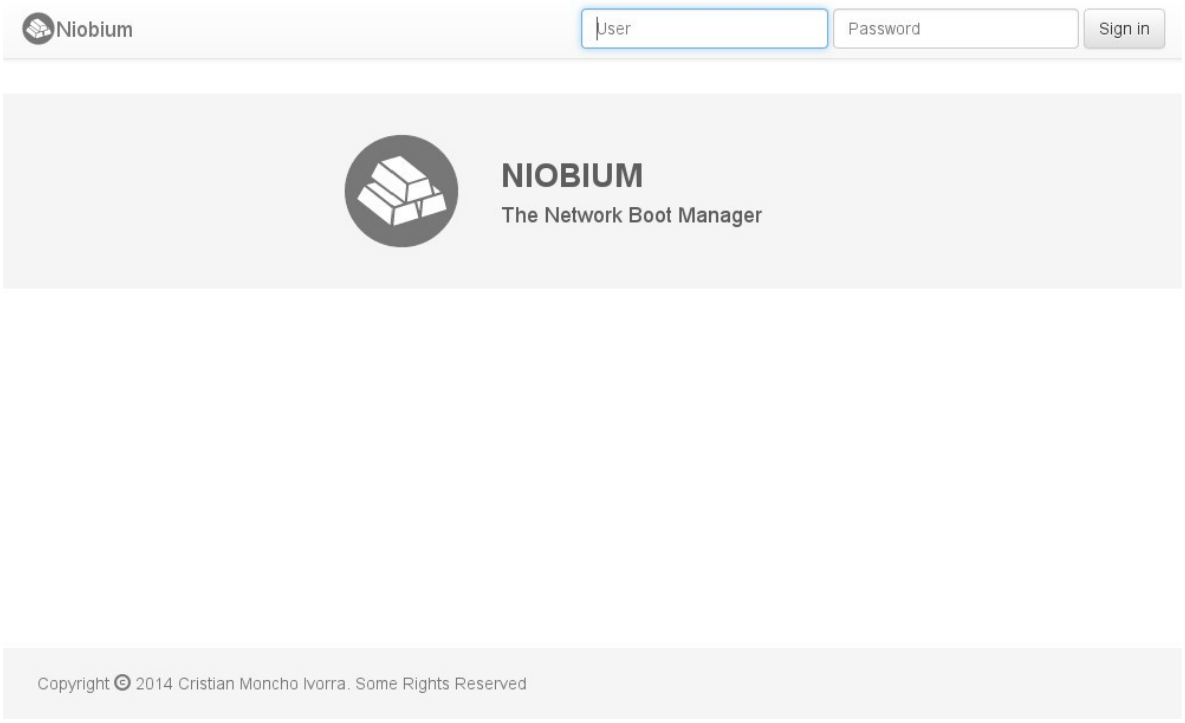

Figura 9. Página de inicio.

En caso de acceso correcto, se mostrará al usuario –independientemente de sus permisos

asociados– la página de escritorio o **Dashboard** (figuras 10 y 11). En ella se podrá observar de forma rápida tanto el estado de los servidores como el acceso en tiempo real de las máquinas cliente y un registro con los 20 últimos accesos para ayudar a localizar de forma más rápida los registros asociados a cada una.

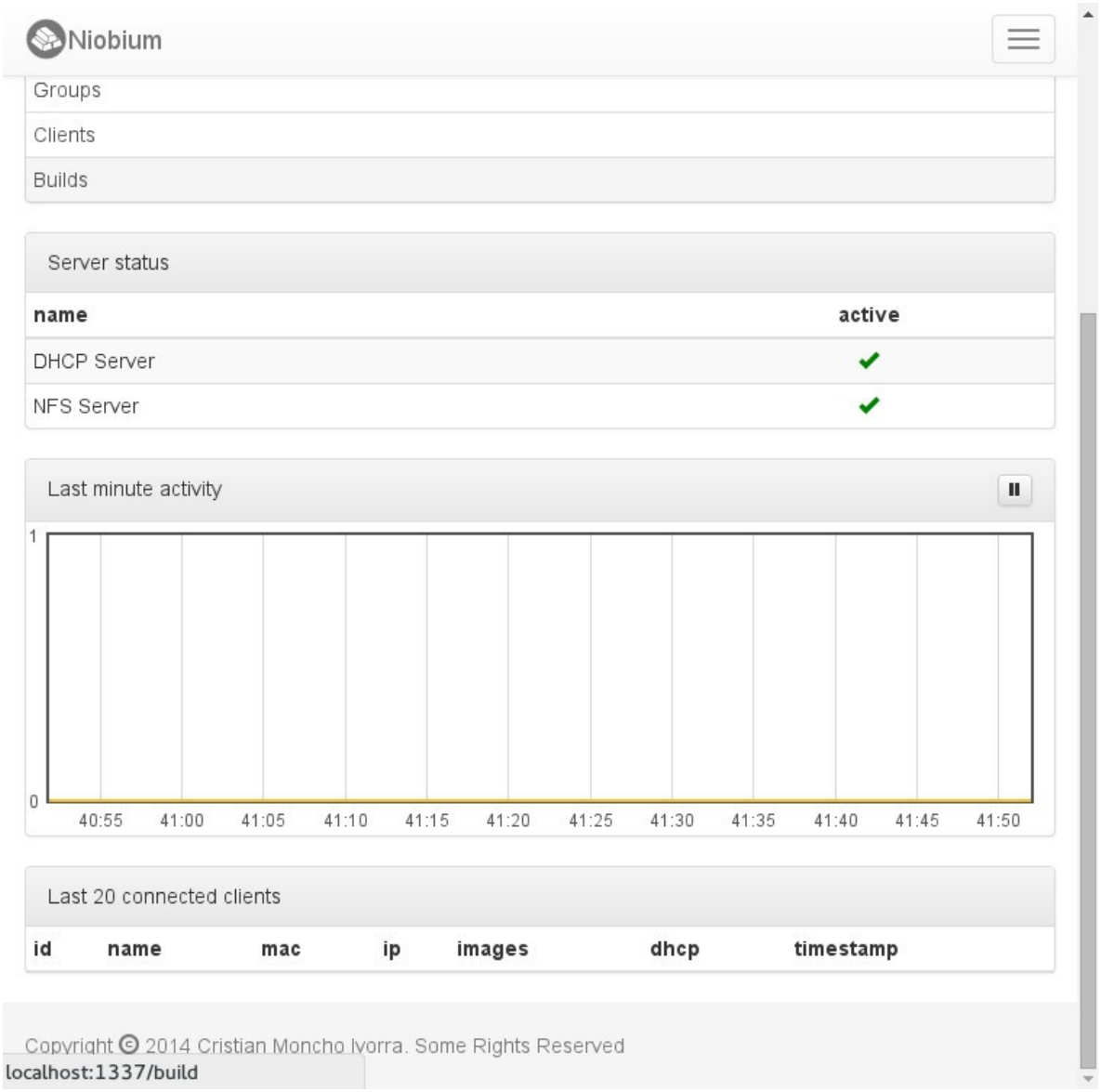

Figura 10. Apariencia de la web en dispositivos móviles.

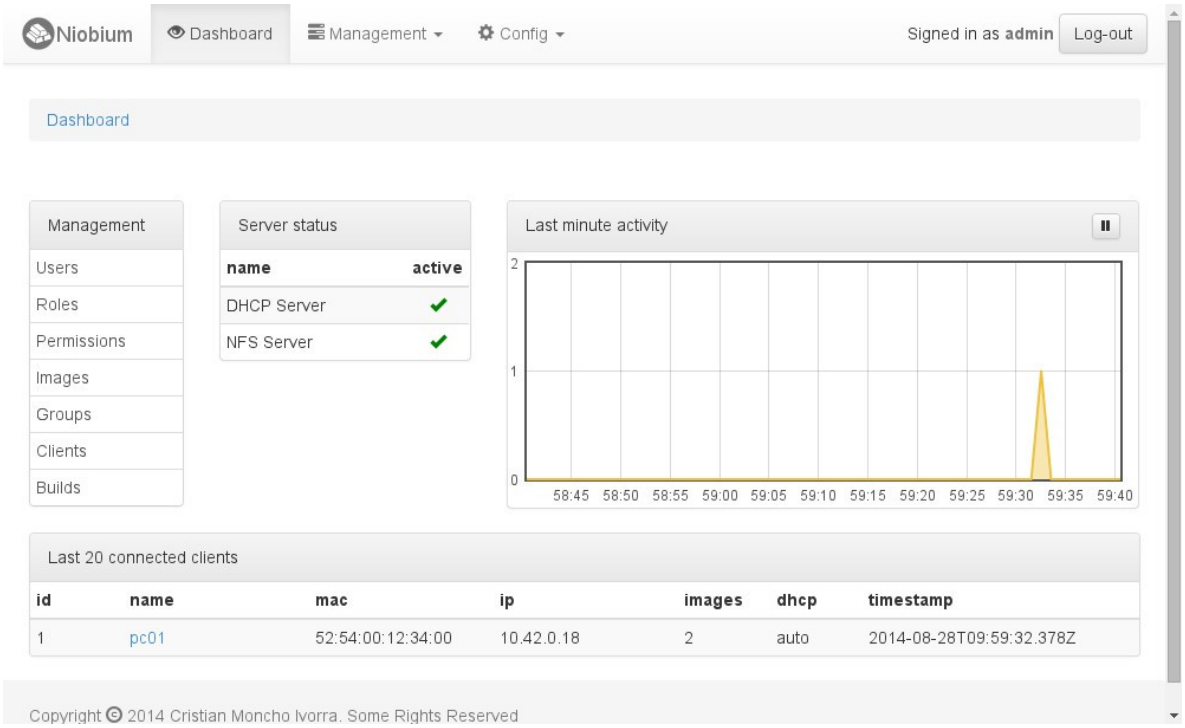

Figura 11. Escritorio.

Como podemos observar en las anteriores capturas, la funcionalidad es la misma pero la distribución de los elementos cambia según la anchura del dispositivo, tal y como se había descrito en anterioridad en los objetivos del proyecto.

En las figuras 12, 13 y 14 podemos ver cómo se ha estructurado la gestión de la base de datos. Hay un total de 2 formas de hacer las consultas, en modo lista y en modo formulario.

- **Lista**. Muestra una lista de todos los elementos de la base de datos para el modelo escogido así como un botón para crear un elemento nuevo. Permite cambiar el parámetro de ordenación y la dirección tras hacer clic sobre el nombre de la columna, y también se puede modificar la cantidad total de registros a mostrar por página. Cada una de las filas ofrece 3 botones: **mostrar**, **editar** y **eliminar**; mientras que los 2 primeros muestran el registro en modo formulario, el último lo elimina –con un mensaje de confirmación–. Además, las filas son seleccionables y se puede ejecutar una acción en masa, o **Bulk Action**, que inicie el proceso de borrado de toda la selección.
- **Formulario**. Se muestra en las acciones **crear**, **mostrar** y **editar**. Permite crear o editar un elemento y almacenar o revertir los cambios. Cada uno de los campos tiene asociado, si es necesario, un *validador* para evitar datos erróneos en el campo (además de validarse de nuevo en el servidor tras recibir los datos) y su generación es automática para cada modelo..

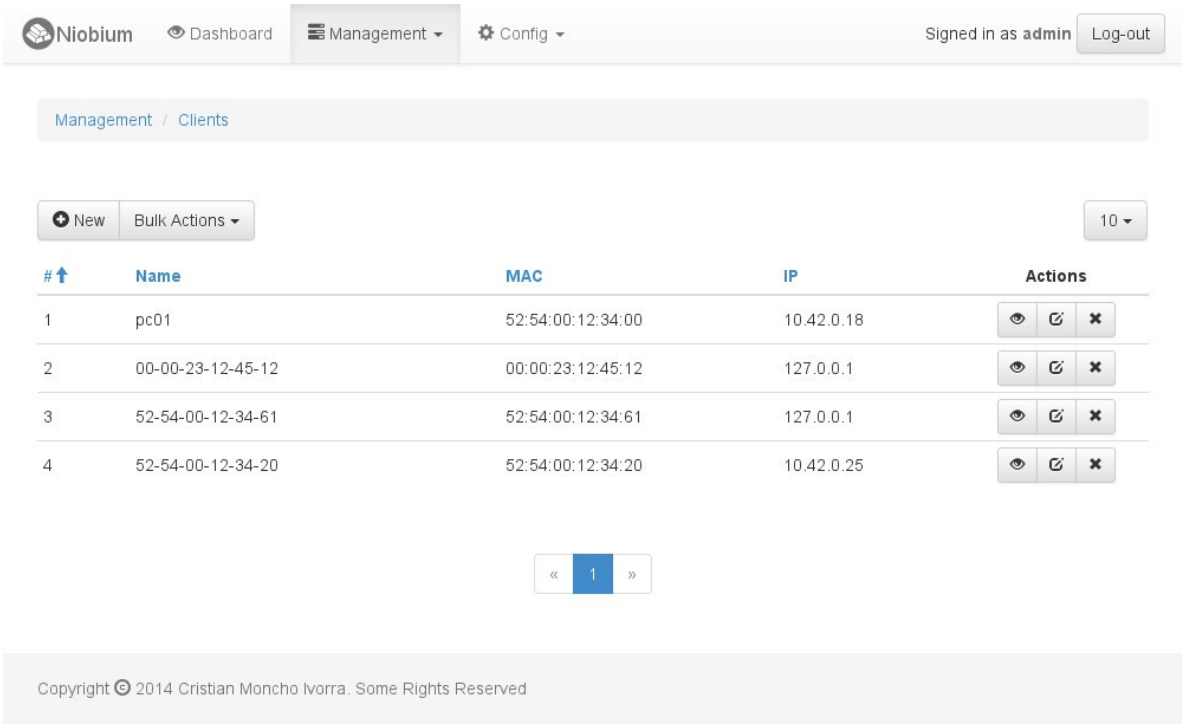

Figura 12. Administración de clientes.

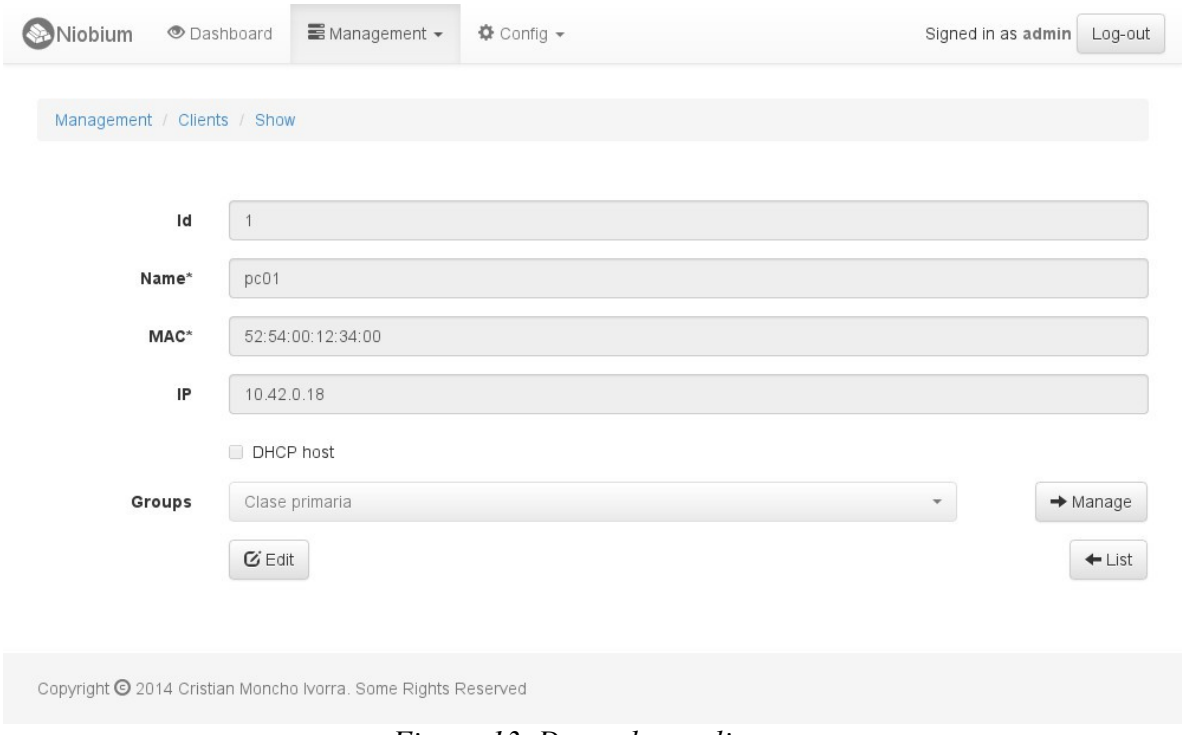

Figura 13. Datos de un cliente.

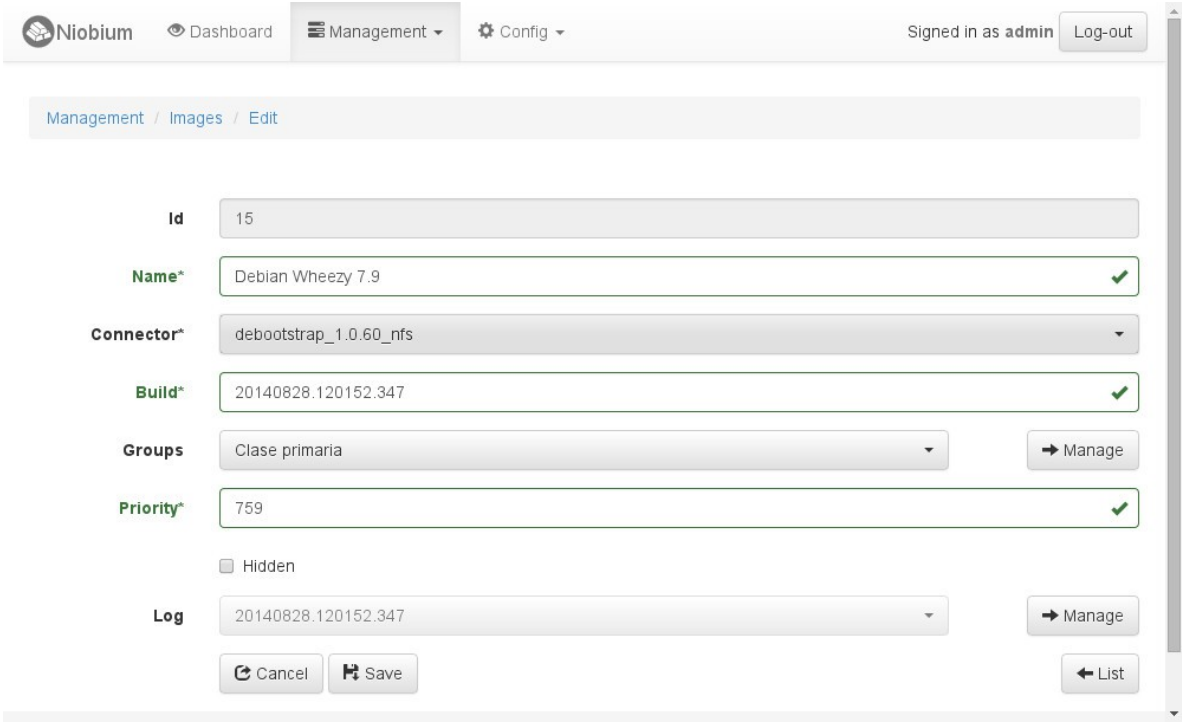

Figura 14. Edición de una imagen.

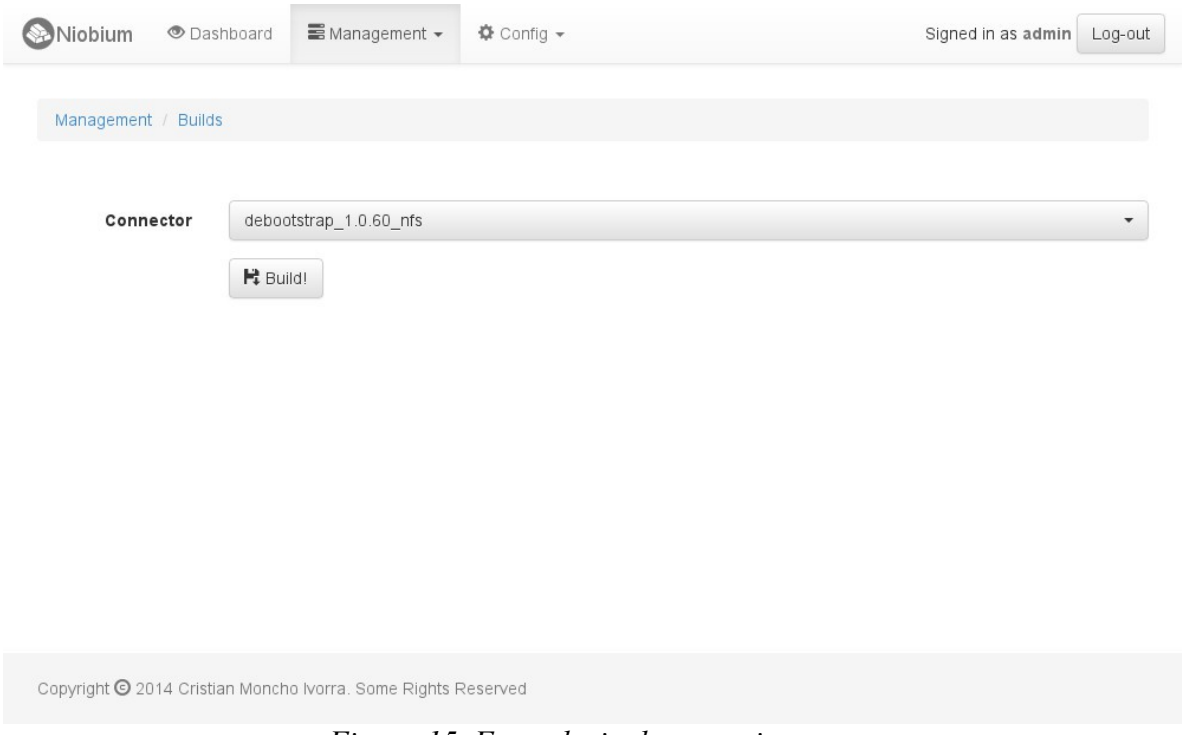

Figura 15. Formulario de nueva imagen.

El último elemento del menú **Management**, la entrada **Build**, muestra el asistente de generación de nueva imagen (figura 15) y la configuración concreta del conector una vez

escogido. Una vez confirmada la operación, inicia el proceso de construcción de una nueva imagen al mismo tiempo que muestra al usuario todos los mensajes enviados por el constructor (figura 16).

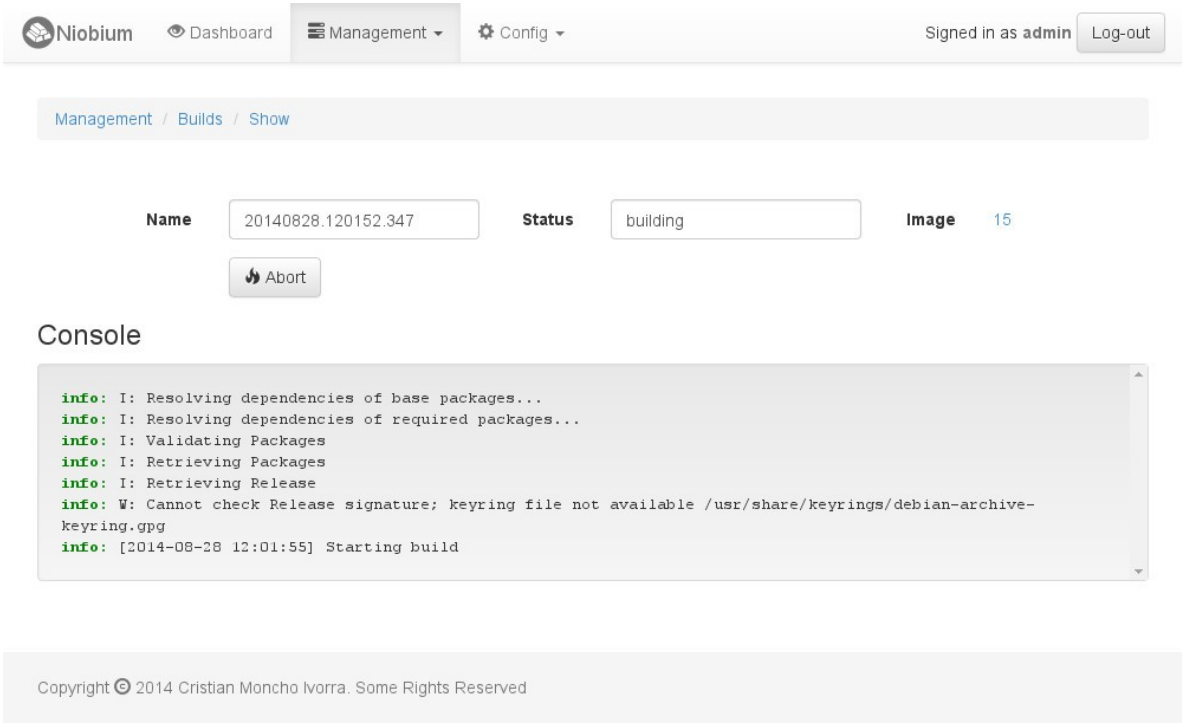

Figura 16. Información y mensajes del proceso de nueva imagen.

En el último menú, **Config**, tenemos acceso a los editores de configuración. El primer elemento, **Niobium**, muestra todos los parámetros internos del servidor. Podemos observar (figura 17) como se avisa al usuario de que son parámetros críticos y que el más mínimo error de configuración puede alterar la estabilidad del servidor o su correcto funcionamiento.

Los otros 2 elementos, **NFS Server** y **DHCP Server**, permiten editar los archivos principales de configuración de cada servicio –`/etc/exports` y `/etc/dhcpd.conf`, respectivamente–. En primer lugar, se generará la configuración de forma automática según los datos internos del servidor. Acto seguido, se mostrarán al usuario un total de 3 bloques (figura 18) con el contenido actual del archivo, el nuevo contenido a guardar y las diferencias entre ambos. Es recomendable dejar todos los valores por defecto, pero se deja opción a que el administrador pueda especificar parámetros de configuración propios.

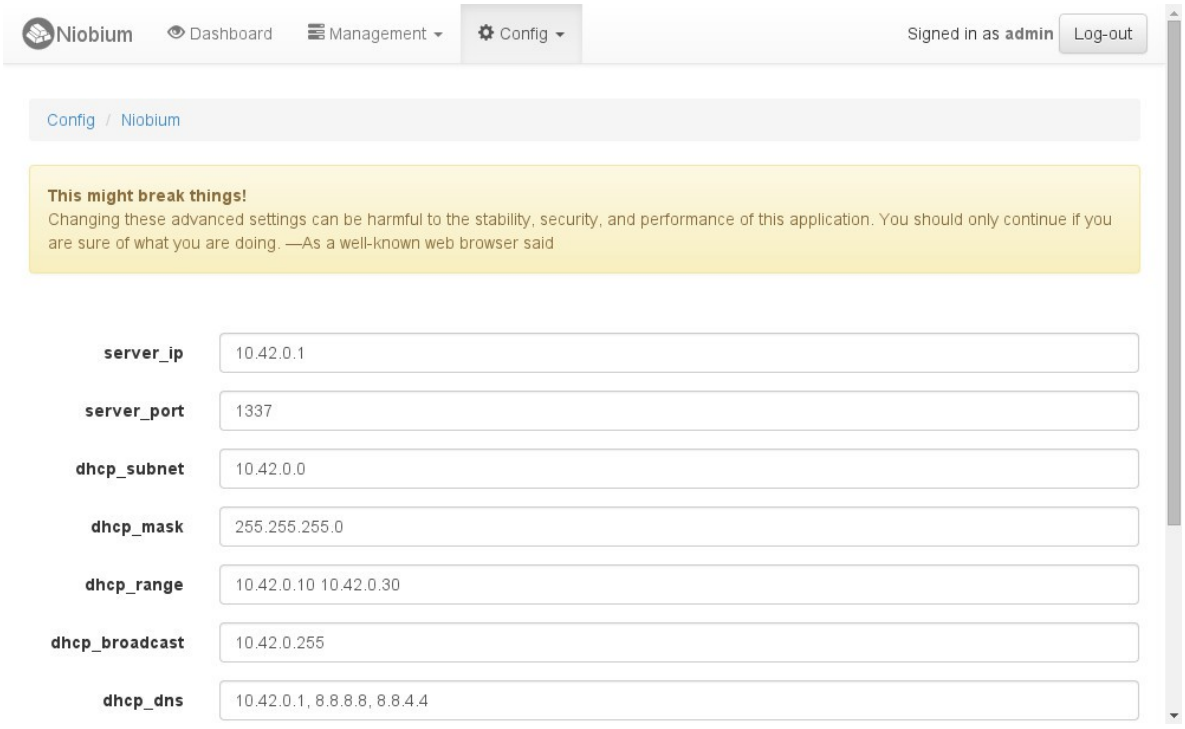

Figura 17. Configuración del servidor.

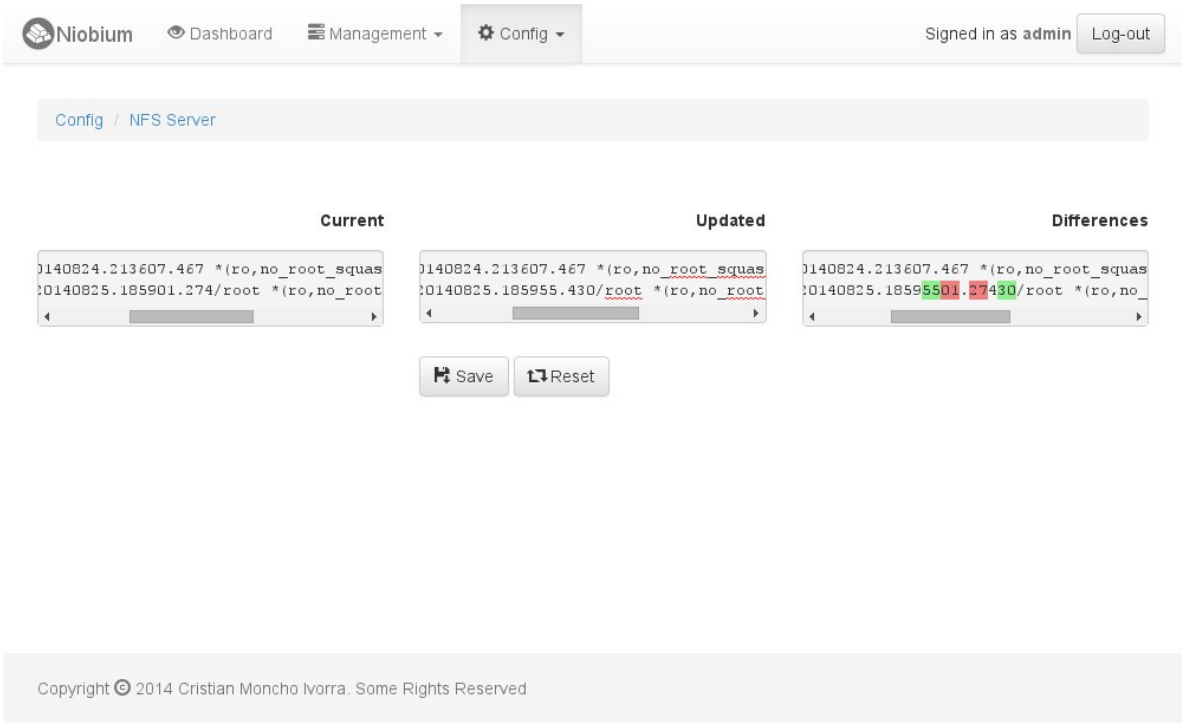

Figura 18. Editor de configuración de servicios externos.

Por último, se adjuntan 2 capturas para mostrar cómo la máquina cliente recibe información de todas las imágenes a las que tiene acceso, en forma de menú (figuras 19), y permite su

selección para su posterior carga (figura 20).

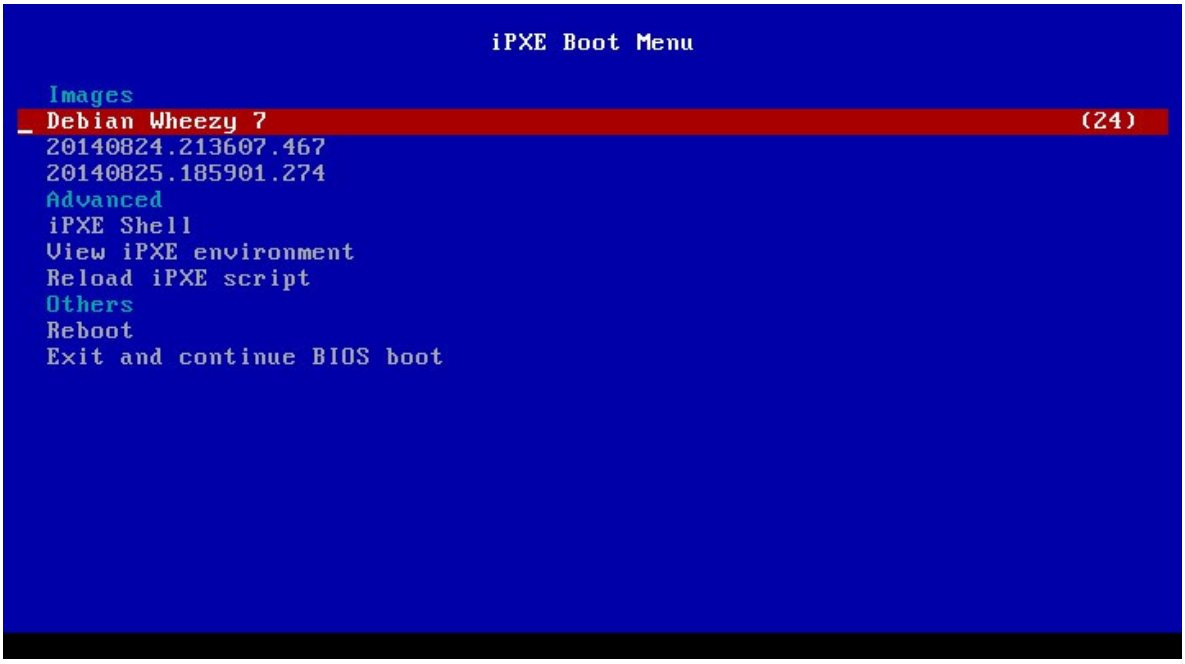

Figura 19. Menú de carga de imágenes en el cliente.

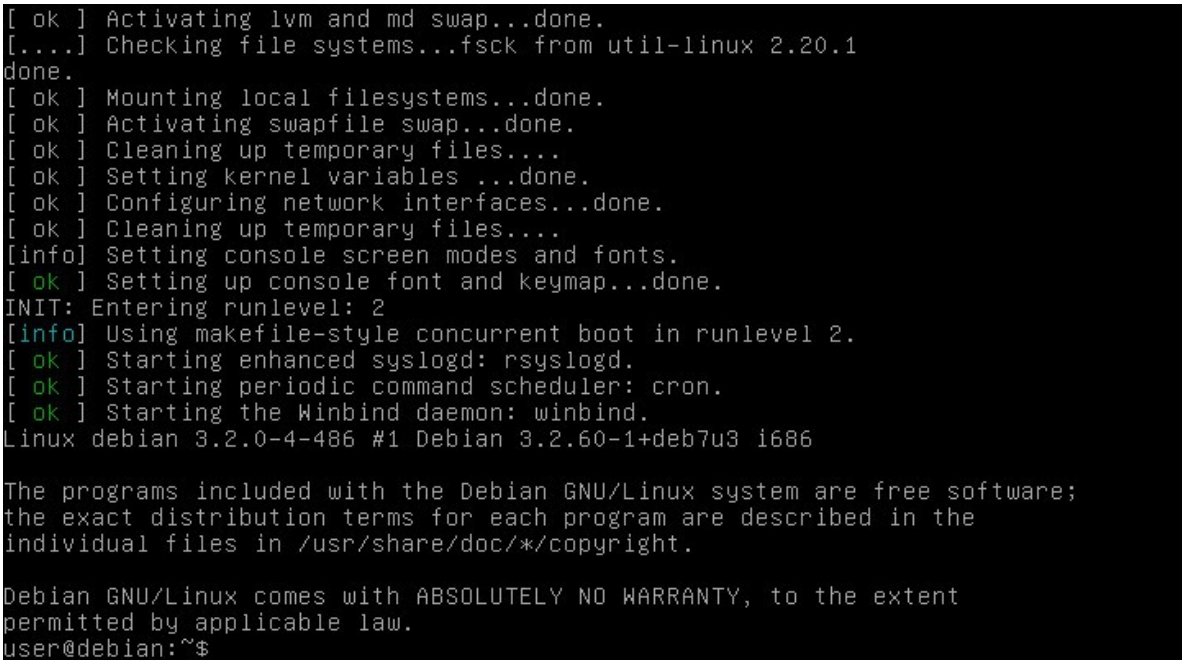

Figura 20. Imagen completamente cargada en el cliente.

5.2. Despliegue y funcionamiento.

En el directorio base del proyecto se encuentra el fichero `utils/install.sh`, cuya

función es la de reproducir todos los comandos requeridos para la correcta instalación del proyecto. Está diseñado para poder identificar qué pasos de la instalación son requeridos según sistema en que se ejecute (actualmente soporta **Debian GNU/Linux** y **Arch GNU/Linux**). Para utilizarlo, es suficiente con otorgar permisos de ejecución a dicho fichero con `chmod +x utils/install.sh`, ejecutarlo con permisos de *superusuario* y completar los pasos del asistente.

Una vez realizada la instalación con éxito, sólo faltará iniciar el servidor ejecutando el fichero `utils/start.sh` y, automáticamente, se pondrá el servicio a la escucha en el puerto 1337. Para acceder al administrador web bastaría con acceder a la siguiente URL desde el navegador, reemplazando el parámetro <*ip_servidor*> por aquella IP –o *localhost,* en caso de tratarse de la misma máquina– que proceda: http://<ip_servidor>:1337. En el primer inicio se crearán, por defecto, 2 usuarios para pruebas (**admin** y **user**, con las contraseñas **admin** y **user** respectivamente), los permisos y roles requeridos por la web y todos los parámetros de configuración interna del servidor. Así como el usuario **user** sólo tiene acceso a la primera página, el usuario **admin** tiene acceso a la totalidad de la web y sería recomendable modificar su contraseña tras una instalación correcta para evitar posibles fallos de seguridad.

La estructura está pensada de modo que todas las máquinas clientes estén conectadas directamente al servidor mediante tarjetas de red y/o conmutadores (*switches)*, aunque se ofrece una posible forma de simular este entorno mediante el uso de `iproute2` y `qemu` a través del *script* `utils/vm.sh`.

No existe ninguna imagen creada por defecto tras una instalación limpia, aunque se ofrecen 4 conectores de ejemplo para poder construirlas desde **Management/Builds**. Para poder utilizar la imagen una vez creada, se puede crear un nuevo grupo, o utilizar uno ya existente, y asignarla a éste. Una misma imagen puede pertenecer a más de un grupo y un mismo grupo puede albergar más de una imagen. Cada vez que un nuevo cliente acceda al sistema, se registrará automáticamente –si no existía ningún registro previo– y se mostrará en **Dashboard**, permitiendo acceder a él para asignarle uno o más grupos. A partir de este momento, cuando el usuario acceda podrá elegir una imagen de entre todas las asociadas a aquellos grupos a los que pertenezca, ordenadas por prioridad descendente y evitando duplicidades.

Para el correcto funcionamiento de las imágenes, es posible que se requiera actualizar la información del servidor NFS. Para ello, se ha de acceder a **Config/NFS Server** y actualizar el fichero de configuración (el sistema lo creará de forma automática, en condiciones normales no se debería modificar ningún valor). Es posible que requiera la ejecución de comandos adicionales, en dicho caso, se especificará al usuario en un bloque

informativo.

Si se desea asignar una IP fija a una máquina cliente, se puede configurar el campo **IP** y habilitar la opción **DHCP host**. Tras esto, se deberá actualizar la configuración del servidor DHCP a través de **Config/DHCP Server** de la misma forma que se ha explicado anteriormente.

6. Conclusiones y trabajos futuros

Con este proyecto se ha conseguido suplir la falta de automatización de las tareas relacionadas con la generación de imágenes, mediante el análisis y abstracción de las partes comunes implicadas en el proceso. Se han cumplido la totalidad de objetivos planteados en un principio, salvo el de facilitar la divulgación de configuraciones –que se comentará posteriormente en el apartado de futuras ampliaciones–.

Las impresiones obtenidas con las herramientas utilizadas han sido más que satisfactorias. Han agilizado en gran medida el proceso de desarrollo y el producto final ha resultado ser tal y como esperaba. La única en la que destacaría algún problema fue **Sails.js**[9] que, aunque novedosa y con un gran potencial, todavía tiene mucho camino por recorrer. A continuación cito dos de los mayores problemas que me he encontrado con este *framework*:

- El ORM no soporta el concepto de transacción –esto se conocía desde el principio y no suponía un problema para el tipo de proyecto– y el mapeo de asociaciones sólo es soportado en parte. El procedimiento para realizar búsquedas relacionales y obtener registros con sus asociaciones es, en mi opinión, mejorable y no posee disparadores para actualizar y/o borrar en cascada.
- Dada la filosofía de programación en que está basado,. no posee el concepto de herencia y ésta se ha de simular o evitar.

A continuación se citarán las posibles mejoras y/o ampliaciones que podrían tener lugar en una futura actualización del proyecto:

- Habilitar el soporte para **Wake-on-LAN**; un protocolo que permite encender, a través de la red, máquinas apagadas. Esto requeriría una configuración mayor en las máquinas cliente e, incluso, en las imágenes a cargar.
- Los datos de las últimas conexiones y de la actividad se procesan en el cliente web, y sería interesante que se procesaran en el servidor para que un usuario que acaba de acceder pueda acceder a esta información de forma actualizada. Esto se podría completar con el uso de **Backbone.js** en la parte de cliente y un mayor uso de **WebSockets**, para mejorar, aun más, los tiempos de carga.
- Añadir más opciones de carga. Actualmente permite cargar desde un sistema de ficheros **NFS** o desde un archivo **SquashFS** y se podría ampliar tanto para funcionar con el sistema de ficheros **NBD** como para compatibilizar con **wimboot**, un cargador de arranque para el formato de imágenes de Windows ***.wim**.
- Sería interesante implementar las funciones de exportar/importar datos para poder

permitir realizar copias de seguridad (o *backups*).

- Junto al proyecto se distribuye un instalador genérico. Debería crearse un paquete específico para cada distribución **GNU/Linux** designando, correctamente, las dependencias y permitiendo su correcta desinstalación.
- Desde **iPXE**[4] es posible averiguar si la máquina soporta la arquitectura de 64 bits, las imágenes podrían almacenar este valor de modo que el generador del *script* de inicio pueda asignarlas en función de las arquitecturas soportadas.

7. Bibliografía

1: FAI Project, Fully Automatic Installation, 2014, http://fai-project.org/
2: Live Systems Project, live-build, 2006-2014, http://live.debian.net/
3: DisklessWorkstations.com, LLC, Linux Terminal Server Project, 2014, http://www.ltsp.org/
4: iPXE Project , iPXE - Open source boot firmware, 2014, http://ipxe.org/
5: W3C, HTML5, 2014, http://www.w3.org/TR/html5/
6: The jQuery Foundation, jQuery, 2014, http://jquery.com/
7: @mdo and @fat, Bootstrap, 2014, http://getbootstrap.com/
8: Joyent, Inc, node.js, 2014, http://nodejs.org/
9: Mike McNeil, Balderdash, Sails.js | Realtime MVC Framework for Node.js, 2014, http://sailsjs.org
10: Jupiter Consulting, Embedded JavaScript, 2014, http://embeddedjs.com/
11: Free Software Foundation, GNU Bash, 2014, http://www.gnu.org/software/bash/

8. Glosario

Término	Definición
Build	También referido como **construcción de una imagen**. Proceso creación de una nueva imagen.
Cliente	Máquina que accede al sistema solicitando una imagen.
Conector	También referido como **Connector**. *Driver*, o capa, que abstrae la configuración de un constructor a un formato reconocible por la aplicación.
Constructor	Programa cuya función es la de iniciar la creación imágenes
Diccionario	Es un conjunto de valores referidos mediante claves alfanuméricas.
Framework	Estructura que sirve como base para la organización y desarrollo de *software*.
Imagen	También referido como **binario** o **BinaryImage**. Fichero, o conjunto de éstos, que contiene los elementos necesarios para poder cargar un sistema operativo en una máquina.
MAC	La dirección MAC, o *dirección física*, es un identificador único que posee cada tarjeta o dispositivo de red.
Máquina virtual	Es una simulación de una máquina –tanto *software* como *hardware*– que ofrece el mismo nivel de funcionalidad que una máquina física.
ORM	Mapeo objeto-relacional o, en inglés, *Object-Relational mapping*. Motor que convierte datos entre una base de datos y un lenguaje de programación orientado a objetos.
Red local	Conjunto de máquinas y/o dispositivos conectados entre sí mediante cables u otras tecnologías inalámbricas.
Registro	Cada uno de los datos creados en una tabla concreta de una base de datos.
RWD	Diseño web adaptable o, en inglés, *Responsive Web Design*. Filosofía de diseño con el objetivo de adaptar la apariencia de la web al dispositivo que la visualiza.
Script	Archivo de texto plano que contiene una serie de órdenes que son entendidas y ejecutadas por un intérprete.
SGBD	Sistema de Gestión de Base de Datos. Sistema que permite el almacenamiento, modificación, extracción y borrado de información en una base de datos.

Usuario	También referido como **Administrador**. Encargado de la configuración y gestión de la aplicación.
Validador	Secuencia de código que verifica que un campo se ha rellenado con valores correctos.
WebSocket	Tecnología que proporciona un canal de comunicación bidireccional entre navegadores y servidores web que evita tener que estar consultando constantemente un mensaje o estado.

Tabla 6. Glosario

9. Anexos

Live-build – Manual: http://live.debian.net/manual/unstable/html/live-manual/toc.en.html

LTSP – Manual (Gentoo): http://wiki.gentoo.org/wiki/LTSP

LTSP – Manual (Ubuntu): https://help.ubuntu.com/community/UbuntuLTSP/FatClients

Sails.js – Documentación: http://sailsjs.org/#/documentation/.

SailsCast – Una extensa colección de videotutoriales del usuario Irl Nathan para iniciación en la programación web con Sails.js: http://irlnathan.github.io/sailscasts/

Waterline – Documentación: https://github.com/balderdashy/waterline-docs.