Autor: Sergio Díaz Álvarez
Tutor: Alejandro Calderón Mateos

Ingeniería en Informática

Universidad Carlos III de Madrid

Proyecto Fin de Carrera

CONAD
CONNECTIONS ADMINISTRATOR

Aplicación web para la gestión de conexiones SSH a servidores.

Índice de contenidos

Índice de figuras

Índice de tablas

Agradecimientos

En primer lugar agradecer a mis padres, Carmen y Javier, todo su apoyo y toda su ayuda, por haber creído en mí y en que podía terminar incluso cuando yo mismo dudaba de ello. También he de agradecerles el haber puesto a mi disposición todos los medios a su alcance para que pudiese completar mis metas y llegar hasta aquí ahora.

A Carlos, mi hermano mediano, sin cuya compañía a lo largo de mi vida yo no podría haber sido quién soy hoy, y también el agradecerle que aunque a veces tengamos nuestras cosas, también tenga un gran interés por la informática y tener también eso en común.

A Jorge, mi hermano pequeño, porque pese a la diferencia de edad estoy muy unido a él, y porque me siento orgulloso de que quiera dedicarse a la informática y me considere un referente para él en la vida, me ha ayudado a conseguir la conclusión de este proyecto para que vea que se puede terminar los estudios y se puede vivir de lo que te apasiona.

A Ángela, porque con nuestros más y nuestros menos sé que en el fondo siempre ha querido y ha creído que era capaz de terminar mis estudios y convertirme en Ingeniero, y que siempre ha puesto lo que ha estado al alcance de su mano para que lo consiguiese.

A Sergio, Luis, Rodrigo, Pablo, Carmen y Clara, sin los cuales mi paso por la universidad no hubiese sido siquiera parecido a lo que fue, por todos los buenos momentos (y los no tan buenos) que han ayudado a conformar mi estancia en la Carlos III. Gracias en una parte muy grande estoy también hoy aquí con el PFC terminado.

A todos mis amigos, que durante el desarrollo de este proyecto han estado largas temporadas de tiempo sin verme y sin saber de mí y siempre que he vuelto a quedar con todos ellos ha sido como si no hubiese faltado ni un minuto.

A mis profesores en la Universidad, sin todos y cada uno de vosotros no podría haber aprendido todo lo que se ahora mismo, sois responsables no sólo de lo que me enseñasteis en su día sino también de todo lo que he ido aprendiendo hasta ahora, porque sin las bases que me proporcionasteis ahora mismo sería imposible poder dedicarme y vivir de lo que realmente me apasiona.

A Alejando Calderón, mi tutor, no sólo por su ayuda con este PFC, sino por todo, él consiguió inculcarme su pasión por la informática y gracias a ello estoy también aquí, también agradecerle todas las guías y consejos que me ha ido dando para lograr completar este proyecto.

Por último agradecer a todas las personas a las que he olvidado dar las gracias en el resto de los agradecimientos pero que también ha ayudado a que yo pueda estar donde estoy y haber conseguido mis metas.

Muchas gracias a todos.

Resumen

Este proyecto surgió de la idea de darle visibilidad a las posibilidades del comando *last* de Linux, que guarda un registro de la actividad de los usuarios en un equipo. Dado que las conexiones *ssh* no dejan de ser en el fondo conexiones al servidor del mismo modo que podrían realizarse en el propio equipo pero de forma remota se tratan para el servidor de la misma manera que se trataría un acceso a terminal en local, puesto que ahora lo normal es tener los servidores en 'la nube' y cada vez es menos frecuente la imagen del servidor en la oficina o el data center de una empresa, pero a la vez sigue siendo necesario un control de los accesos de los usuarios a los servidores, para cubrir esta necesidad se planteó la posibilidad de obtener las conexiones *ssh* que se realizan a un servidor y mostrarlas de una forma sencilla e intuitiva a un administrador que necesite monitorizar dicha actividad.

De esta idea nació el proyecto CONAD, que a través del comando *last* obtiene las conexiones *ssh* de los usuarios a un servidor, pero solo con esta idea no es suficiente para desarrollar un proyecto software completo y viable, para ello se llevó a cabo un estudio mucho más detallado de la idea para poder establecer unas bases en las que se cimentaría el proyecto a desarrollar. Lo primero que se hizo necesario en ese momento fue comprobar que la idea realmente tenía aplicaciones prácticas para su uso, dado que sino no tenía sentido el comenzar un desarrollo que luego nadie fuese a usar, de ahí surgió la idea de poder monitorizar el uso de los equipos de las salas informáticas, ver cuales tenían un uso mayor, por ejemplo, o poder sacar estadísticas de tiempo de uso de los usuarios de los diferentes equipos. Teniendo ya la idea y al mismo tiempo en que íbamos a emplearla llegó el momento de establecer los requisitos mínimos que iba a tener que cumplir para poder considerarse un éxito.

Este razonamiento es el que llevó a un cambio de estrategia con respecto a la idea original que consistía en centrarse en los usuarios en lugar de estar centrados en servidores y número de conexiones, ya no tenía sentido almacenar usuarios que se conectasen independientemente del equipo y ver en qué equipos se conectaban como podría haber sido una primera idea, de hecho la idea original era el seguimiento de un solo equipo y poder ver los usuarios que se conectaban, a qué hora lo hacían y la duración de sus conexiones, hasta que realmente estudiando el potencial de la idea nos dimos cuenta que era mucho más útil lo expuesto hasta ahora, la utilización del comando *last* para obtener conexiones a diferentes servidores, lo de hacerlo multiservidor surgió del hecho de que dado que hay que almacenar toda la información necesaria para poder conectar con un servidor y obtener sus conexiones, hacer lo mismo varias veces para varios servidores era trivial y aportaba mucha información que el usuario final luego iba a saber valorar.

Una vez establecida la base del proyecto, surgió la primera decisión de diseño, la selección de la herramienta en la que se iba crear y desarrollar la aplicación web, dado que ahora mismo hay muchos lenguajes y *frameworks* para la realización de aplicaciones web se antojaba complicado seleccionar uno frente a otro, como se puede extraer de las comparativas que hay más adelante en este mismo documento la elección del lenguaje no fue trivial, de los principales lenguajes y *frameworks* de desarrollo web los que mejores características tenían eran *Ruby* y *Phyton*. Como se expone con posterioridad una de las grandes complicaciones que tiene el *framework Ruby on Rails* y de Ruby es el conocerlo, es un lenguaje con una gran curva de aprendizaje pero que una vez que se sabe manejar da una versatilidad que no dan otros

lenguajes, teniendo todo esto en cuenta y contando con un conocimiento en el desarrollo con dicha herramienta finalmente se decide hacer el desarrollo del proyecto utilizando el *framework* de *Ruby on Rails*.

Tras tener decididas las líneas generales de lo que va a ser el proyecto y la plataforma en la que se va a realizar el desarrollo se procede a establecer los objetivos que va a tener que cumplir el proyecto para poder considerar que se ha completado con éxito y a su vez para poder marcar unos mínimos que tienen que cumplirse en cualquier caso pero que a su vez pueden ser mejorados durante el proceso de desarrollo de la aplicación. El primer objetivo y principal es la obtención de los datos de conexiones de los distintos servidores que se quieren monitorizar con ella, como ya se ha comentado con anterioridad se utiliza el comando *last* de los sistemas con base UNIX, por lo tanto por ahora el sistema CONAD sólo podrá usarse con servidores UNIX que son los únicos que tienen implementado el comando *last* y por tanto son los únicos de los que se va poder extraer datos con esta versión de la aplicación, una posible mejora para el futuro es obtener una solución similar para otros sistemas operativos que no tengan la base UNIX de modo que la herramienta web tenga una repercusión mayor y pueda ser utilizada por un número mayor de usuarios.

Para la obtención de los datos es necesario disponer de un usuario y una contraseña *ssh* para cada uno de los servidores de los que se quiera hacer seguimiento, con esta información *ssh* la aplicación se conecta al servidor y ejecuta el comando *last*, después la aplicación se encargará de procesar la respuesta obtenida en el servidor y convertirla en las conexiones con las que luego se va a trabajar y cumplir así con el resto de objetivos.

El procesado de los logs obtenidos del comando *last* para cada servidor se realiza en una tarea en *background* que de cada línea recogida extrae la información que necesita para poder mostrarse con posterioridad, en este caso, se extrae el usuario que ha realizado la conexión y la hora a la que se produjo la misma. Para la ejecución de las tareas que recogen la información de los *logs* se usa una gema (librería) de *Rails* llamada *sidekiq* que se encarga de lanzar de forma asíncrona en segundo plano tareas que requieran una gran carga de procesado por parte del servidor, un ejemplo típico de este tipo de tareas sería el envío de correos electrónicos por parte de la aplicación. Teniendo la obtención de datos en tareas asíncronas y paralelas se consigue que el procesado se realice en un tiempo menor dado que no tiene que esperar a que se haya procesado el servidor anterior para comenzar a procesar el siguiente, también se consigue que si por lo que sea falla la obtención de datos en un servidor el resto de servidores no se vea afectado y sí que se pueda obtener información de ellos, además se prevé también que pueda producirse que el tiempo de respuesta del servidor tras la petición de actualización de las conexiones sea superior al máximo establecido y que se detuviese la ejecución, puesto que los servidores web tienen establecido un tiempo máximo de respuesta después del cual la petición se desecha.

El siguiente de los objetivos a cumplir es el de gestionar los diferentes servidores que van a estar almacenados y de los que se va a guardar registros de conexiones en la aplicación. Puesto que para conseguir dicha información hace falta almacenar información sobre ellos, esta información tiene que ser accesible para el usuario final de la aplicación dado que puede necesitarla, para la creación de un servidor nuevo dentro de la aplicación serán necesarias, la

URL del servidor, así como un usuario SSH con permisos para ejecutar el comando *last* con su correspondiente contraseña, también es necesario introducirle un nombre para poder diferenciarlo del resto de servidores ya introducidos en la aplicación.

Estos datos una vez introducidos serán accesibles para el usuario y estarán a su disposición para su modificación en caso de ser necesario, también se permite eliminar el servidor, hacer esto implica eliminar el servidor así como todas las conexiones que se tengan almacenadas de este. La posibilidad de crear nuevos servidores o de modificar los datos de los ya almacenados se hace de una forma sencilla e intuitiva para facilitar el uso de la aplicación a los usuarios.

Lo cierto es que para la presentación de la información almacenada en la aplicación se hace necesaria una forma intuitiva de mostrar las conexiones frente al tiempo o de limitar un periodo de tiempo sobre el que se quieren ver los datos. Con todo esto se hacía evidente que otro de los objetivos de la aplicación debía ser que esta dispusiese de un calendario que mostrará el mes en el que se encuentre navegando el usuario en ese momento. Además si el usuario desea afinar más la información que quiere que se le muestre podrá, para cada día dentro del calendario, tener acceso a los datos de conexión de distintos servidores de los que se componga la aplicación, viendo la información en lugar de por día como se tiene previsto que sea la primera vista de la aplicación, sea por horas.

Para poder incluir un calendario en nuestra aplicación se buscó alguna gema que permitiese de forma sencilla incluir el calendario de un mes, lo que en principio debería haber sido una tarea bastante sencilla comparada con otra como podría haber sido la paralelización de procesos que se ha comentado con anterioridad, pero resulta que no había ninguna gema que se adaptase perfectamente a las necesidades de la aplicación. Se empezó a utilizar una que utilizaba tablas de html para crear el calendario, pero no permitía añadir atributos personalizados, sólo permitía modificar las clases de las celdas de modo que se hizo necesario cambiar la librería para que se adaptase a las necesidades de CONAD. El poder añadirle atributos personalizadas a las celdas era necesario porque se quería que cuando se hiciese click en un día este llevase a la vista de la información para ese día. Esto es posible gracias a la librería *jQuery* que permite a través de atributos personalizados en los elementos html que en estos se pueda hacer click e ir a una URL establecida en dicho atributo del elemento.

Con el calendario lo que se consigue es poder elegir qué datos son aquellos que se va a querer visualizar en el momento, se va a poder elegir un periodo de tiempo de la forma que se ha considerado más sencilla e intuitiva para el usuario final, es decir, aquello a lo que está acostumbrado, y no hay nada más común para representar el tiempo que un calendario. Además esto está muy en relación con el siguiente objetivo que la aplicación quiere cumplir que es la inclusión de gráficas para la representación de los datos, de modo que con el calendario se eligen las fechas a visualizar y luego para esas fechas se tendrá una serie de gráficas en las que ya se podrá ver el número de conexiones para ese periodo de tiempo seleccionado.

Por tanto el siguiente objetivo que se planteó es que la aplicación debería disponer de algún tipo de representación gráfica de los datos que se recojan de los diferentes servidores que ya se han registrado en la aplicación y de los que ya se estaría recogiendo sus conexiones. Se ha pensado que a través de una representación gráfica a la hora de utilizar la aplicación va a

ser más sencillo e intuitivo para el usuario final la visualización, de esta manera se consigue que de una manera rápida el usuario se pueda hacer una idea general del estado de sus servidores con un simple vistazo.

Además estás gráficas serán de diferentes tipos, de este modo, los usuarios de la aplicación podrán elegir el tipo de gráfica que mejor se adapte a las necesidades de visualización que tenga en el momento y de este modo facilitarle la toma de decisión de las posibles acciones que pueda necesitar tomar sobre los servidores que tiene bajo su supervisión, por ejemplo, se creará una representación gráfica que permita al usuario ver qué porcentaje de las conexiones se realizan en que servidor.

Para la representación gráfica se ha utilizado una librería de *javascript* denominada *d3js*, que tiene una repercusión y utilización muy alta en el desarrollo de aplicaciones web dado que proporciona muchos ejemplos y con ciertas variaciones sobre ellos puedes tener representaciones gráficas muy sencillas, pero realmente lo interesante de la librería es que es muy versátil, permite un gran nivel de personalización de las representaciones gráficas para adaptarlas a las necesidades visuales que se tenga en cada momento, además combinada con otras librerías de *javascript* permiten el procesado de la información de una forma bastante eficiente en el lado del cliente de la aplicación, aunque para el caso concreto de esta aplicación el procesado mayor de la información y la preparación de los datos se ha realizado en la parte del servidor que suelen ser máquinas más potentes y estar mejor preparadas para trabajar con una cantidad de datos mayor.

Todo esto es lo que llevó a la elección de d3js para CONAD, dado que se necesitaba poder representar los mismos datos de diferentes formas no se dudó en que la mejor elección era esta. Luego otra decisión que hubo que tomarse fue la de que tipos de gráficas eran las que se iban a utilizar para este proyecto, se buscaban representaciones gráficas que realmente aporten al usuario final información relevante sobre el estado y las conexiones que se realizan en los diferentes servidores monitorizados.

La primera gráfica que se decidió que iba a formar parte de la aplicación fue la gráfica de líneas, dado que es la representación más común de datos se decidió que sería la que se vería por defecto al entrar en la aplicación, en ella se ve la evolución de las conexiones en valor absoluto con respecto al tiempo que se tenga seleccionado, ya sea un mes o un día. La segunda que se ha considerado relevante para este proyecto es la visualización de los datos a través de una gráfica de barras, esta manera se puede ver de un simple vistazo cuales son los servidores con mayor número de conexiones para el periodo de tiempo seleccionado para la visualización de los datos, la desventaja de esta representación es que muestra los totales pero no las fechas en las que se produjeron por eso surge la idea de generar el siguiente gráfico, el gráfico de barras apiladas. En esta versión se puede ver parte de las ideas principales de incluir los otros dos gráficos de una forma simultánea, aunque a la vez distinta, si realmente con este gráfico se recogiese la misma información que en los otros dos no tendría sentido la inclusión de dos gráficas que se han vuelto superfluas a la hora de visualizar los datos. En el gráfico de barras apiladas se ve una barra con las conexiones de un día (u hora si se ha seleccionado un día para verse) en valor absoluto además con un color diferente para cada servidor de modo que se pueden ver de una sola vez las conexiones de un servidor y proporcionalmente lo que

representa en número de conexiones con respecto al resto para el mismo periodo de tiempo, de modo que se puede saber cuáles son los que han recibido más conexiones con respecto al tiempo y no los valores absolutos como nos representa un gráfico de barras normal. El último gráfico incluido en este proyecto es el gráfico de donut, es equivalente al 'gráfico de tarta' y en él se ve sobre el total de conexiones que se han producido en el periodo de tiempo seleccionado la parte proporcional que representa cada uno de los servidores (o usuarios si se tiene seleccionado un único servidor para la visualización), la principal diferencia entre esta representación y la gráfica de barras es que mientras que en la de barras los datos de conexión son absolutos, en este los datos son proporcionales para cada uno de los servidores, de modo que la combinación de ambos es más útil para el usuario final de la aplicación que el disponer únicamente de una de las dos posibilidades de ver los datos.

El último de los objetivos que se quiere alcanzar con este proyecto es el de que la aplicación esté en varios lenguajes para que los usuarios de diferentes países e idiomas puedan utilizar la aplicación en su lengua materna y de este modo se sientan cómodos con la herramienta. *Ruby on Rails* está preparado para la gestión de diferentes idiomas y viene con la librería I18n instalada que es la que realiza el cambio de un lenguaje a otro según el deseo del usuario. Hay varias formas de afrontar la selección de idiomas por parte de los usuarios, una de ellas es la de guardar perfiles de usuarios en los que se almacene el idioma en el que quiere visualizar la web, esta idea se desestimó debido a que no se van a almacenar por ahora perfiles de usuario y por lo tanto no es podía aplicar a las características de nuestro proyecto, otra posible opción de gestión de idiomas es para dominios, según el dominio desde el que se accede a la aplicación web el sistema se adapta al idioma oficial de dicho dominio, es decir, si por ejemplo dispusiésemos de la aplicación en el dominio conad.es, es lenguaje sería castellano y si por ejemplo también tuviésemos la aplicación bajo conad.com, en este segundo dominio se visualizarían los texto en inglés, pero en este caso, tampoco sería aplicable a la herramienta que desarrollada, dado que en este caso no está pensada como una aplicación que en principio vaya a contar con un dominio propio, está pensada como herramienta para los gestores de los sistemas de varias aplicaciones "principales" que probablemente sí que dispongan de su propio dominio o incluso de varios, en los que sí que se podría aplicar esta técnica, de hecho en nuestro caso es probable que se acceda a él directamente con una IP, es menos probable que sea con un subdominio y poco probable que disponga de dominio propio. Esto nos llevó a la siguiente de las posibilidades de selección de idioma, que es la de a través de la obtención del idioma utilizado por el usuario final en el navegador, la cual es una solución muy interesante y que era aplicable a nuestro caso, pero que al final fue desechada. La razón por la que se ha desechado dicha solución pese a ser viable y de hecho bastante buena es la aparición de la solución final adoptada, esta consiste en la determinación "manual" del idioma en el que se quiere visualizar la información de la aplicación.

En un primer momento se intentó adoptar una solución que aunase las dos opciones comentadas con anterioridad, que tomase el lenguaje del navegador como lenguaje por defecto salvo en los casos en los que se proporcionase el idioma deseado en la URL como se ha comentado. El problema surgía cuando al definir el inglés como idioma por defecto, tanto las URL con */en/* como las que no tenían idioma seleccionado redirigían a la URL sin idioma, y una vez ahí cogía el idioma por defecto del navegador, de modo que si navegabas en un lenguaje que no fuese inglés, por ejemplo, castellano, y querías visualizar la aplicación en inglés, esto

era imposible, dado que seleccionabas la URL con */en/* pero te redirigía a la URL sin parámetros y una vez ahí cogía el idioma del navegador.

Por todo esto no se podían aplicar ambas políticas de forma simultánea y se debía optar por una de todas las opciones que se han descrito, y se optó por las URL con el lenguaje incluido en ellas. La razón para la toma de esta decisión es que puede darse el caso de querer ver la aplicación en un lenguaje diferente al que se tenga seleccionado por defecto, por ejemplo, puede darse el caso de que el usuario tenga seleccionado en el navegador un idioma no soportado por la aplicación, en este caso se derivaría al lenguaje por defecto, inglés, pero puede darse el caso de que prefiera otro de los lenguajes definidos en la aplicación distinto al que usa en el navegador, y diferente al idioma por defecto de la aplicación, en este escenario al usuario le sería imposible elegir un tercer idioma distinto al de su navegador o al por defecto. Para evitar este caso es la razón por la cual se optó por URL con el idioma, además otra razón es porque puedes enviarle a otra persona la aplicación en el idioma que te interese o le interese a la otra persona definiéndolo en la URL que se envía.

Por lo tanto, y gracias a todas las decisiones que se han ido tomando y que ya se han ido describiendo al final puede decirse que se han cubierto todos los objetivos propuestos para este proyecto software. Primero y principal la obtención de los datos que posteriormente van a mostrarse durante el uso de la aplicación web, así como permitir que dicha información sea obtenida no sólo de un único servidor, si no que por el contrario puedan ser múltiples servidores los que puedan ser monitoreados por la herramienta. Además también se consiguió la inclusión de un calendario que facilitase la visualización final de los datos una vez procesados por la aplicación junto a las diferentes gráficas comparativas que completan lo que sería la parte principal de la visualización de los datos. Y por último también se ha cumplido el objetivo de conseguir una aplicación multilenguaje para su posible exportación y explotación en otros países diferentes de España, y en esta versión que pueda ser utilizado por cualquier persona que sepa manejarse en inglés o en castellano.

Para la correcta toma de las decisiones comentadas y a su vez para tener los conocimientos necesarios para poder tomarlas con el máximo criterio posible ha sido necesarios aprovecharse de los conocimientos adquiridos durante el estudio de Ingeniería informática, una de las primeras asignaturas que ha sido necesaria para la realización de este proyecto han sido las de Ingeniería del software, dado que es una de las asignaturas que enseñan cómo se ha de gestionar un proyecto software de principio a fin, tanto los diferentes documentos de los que se ha de componer, así como de la correcta redacción de los mismos, de las secciones que tiene que contener y como hay que realizarlos de forma apropiada. Gracias a esta asignatura es a la que ha sido posible la realización de la mayor parte de las secciones de las que se compone este documento.

También ha resultado es muy relevante la asignatura de Sistemas Informáticos, dado que en el fondo es la que sería una asignatura práctica de lo que luego va a ser la realización de un proyecto software de verdad similar a lo que ha sido la realización de este Proyecto Fin de Carrera, ya que no sólo se encargaba de cubrir la parte de documentación del proyecto sino que además contaba con una parte práctica que consistía en la realización de una aplicación completa que al igual que en este caso contaba con una parte de *backend* que era la

encargada de toda la parte de gestión y también de una parte de *frontend* equivalente a la de una aplicación web en la que se veía cómo funcionaba una fábrica automatizada, por tanto, y siempre guardando las distancias ha sido una experiencia similar a lo que luego ha resultado ser la realización de este proyecto, aunque también es cierto que en ese caso el trabajo se realizaba dentro de un equipo de varias personas mientras que esto es un proyecto personal que se desarrolla por un único miembro.

En este caso al tratarse de una aplicación web han sido imprescindible contar al menos con los conocimientos sobre conexiones y protocolos adquiridos en las asignaturas de telemática y redes de ordenadores. En estas asignaturas se estudió como se realizan las conexiones y la comunicación entre las distintas máquinas, sin saber cómo funcionan las conexiones y se comportan los dispositivos al interactuar entre ellos es imposible realizar y utilizar aplicaciones web. Gracias a estos estos conocimientos se ha podido realizar la configuración de los ficheros *Nginx* para permitir la conexión de la aplicación a internet y poder manejar dominios y subdominios en una misma máquina, también permite la gestión de diferentes aplicaciones en una misma máquina redirigiendo el tráfico a diferentes puertos por de acuerdo al subdominio o dominio del que vienen o se pueden redirigir varios dominios a una misma aplicación alojada en la máquina en cuestión. También han ayudado a conocer otros protocolos como el *SSH* que es imprescindible para poder realizar la aplicación, dado que es la manera en la cual puedes conectarte a la máquina o máquinas en la que está alojada la aplicación y poder configurarla, comprobar *logs* o cualquier tipo de operación que se necesite realizar sobre alguno de los servidores que se gestionen, otro protocolo muy importante es el *SSL* que es necesario para las conexiones https, y conocer cómo hay que configurar los certificados necesarios para poder hacerlo funcionar, también ha sido necesaria para poder configurar las entradas de *DNS* para poder manejar dominios, ha sido necesario conocer los diferentes tipos de entradas *DNS* para saber cuáles eran las necesarias para poner a funcionar una aplicación web, y la combinación de *DNS* y *Nginx* es lo que permite que la aplicación sea accesible desde la *URL* http://conad.staging.kotraders.com

También han sido necesarias las asignaturas de bases de datos, aunque *Rails* cuenta con un sistema para abstraer el acceso a la base de datos por medio de lo que llaman *ActiveRecord,* este sistema de abstracción está basado en el patrón Modelo-Vista-Controlador y por tanto permite tratar el modelo como un objeto con persistencia en la base de datos sin necesidad de conocer que base de datos hay detrás en cada caso. Esta es la teoría, y en la práctica se cumple para la mayor parte de los casos, pero en el momento que se quiere hacer algo un poco más complicado es necesario hacer uso al menos de algunas sentencias básicas de SQL, esto es debido a que *ActiveRecord* está preparado y pensado para bases de datos relacionales, de hecho normalmente se utiliza con *MySQL* o con *PostgreSQL*, pero en otros proyecto se ha utilizado con otras de bases de datos no relaciones, lo único que es necesario para utilizar una base de datos u otra es instalar el adaptador correspondiente para utilizar *ActiveRecord* con casi cualquier base de datos. Dado que para el caso concreto de esta aplicación se utiliza *postgreSQL* como base de datos ha sido muy necesario y útil como ya se ha dicho las asignaturas de Bases de Datos y Ficheros, dado que en ellas se nos enseñaron los principios en los que se fundamentan las bases de datos así como las nociones básicas de bases de datos relacionales, imprescindibles para la realización de este proyecto software.

Antes del paso por la Universidad no había realizado ningún tipo de desarrollo con ningún lenguaje de programación por lo tanto ahora mismo no sería posible la realización de este proyecto sin contar con los conocimientos de asignaturas como por ejemplo ADA (Análisis y diseño de algoritmos), Programación o Paradigmas de la programación, ya que son las que me enseñaron a programar con paradigmas como la orientación a objetos y las que me hicieron adquirir lo que podría denominarse la *mentalidad del programador* que lo que hace que ante los diferentes problemas o retos que se nos presentan durante un proyecto de desarrollo web sepamos cómo enfrentarlos y solucionarlos de una forma estructurada, permitiendo reducir siempre el problema que hay que solventar en otros más sencillos. Además el hecho de que en estas asignaturas se aprendiese con lenguajes que siguen el paradigma de la programación orientada a objetos me han ayudado a que pese a no desconocer un lenguaje como *Ruby* no haya sido demasiado complicado el aprendizaje ya que guarda similitudes con otros lenguajes estudiados con anterioridad en la carrera como *Java*.

Pero sin lugar a dudas las asignaturas relacionadas con Sistemas Distribuidos han sido de las que más útiles me han resultado para el desarrollo de esta aplicación dado que se trata de una aplicación web basada en una arquitectura *Cliente – Servidor* y sin los conocimientos necesarios adquiridos en asignaturas como la propia Sistemas Distribuidos que trataban este tipo de arquitectura me hubiese sido mucho más complicado comprender exactamente en qué consiste el *framework Ruby on Rails* y no se le hubiese podido sacar todo el partido necesario para el correcto desarrollo de la aplicación cumpliendo con las necesidad y requisito solicitados a la conclusión del proyecto. A la hora de procesar las conexiones de los distintos servidores la aplicación se sirve de un sistema de procesos que se ejecutan en paralelo en el *background*, para poder conocer y entender los conceptos que hay detrás de este sistema fueron necesarias asignaturas como Sistemas Operativos para saber de qué manera se llevaban a cabo dichas tareas, además también resultaron imprescindibles otros conceptos adquiridos en dicha asignatura como las partes críticas del código o los bloqueos que podían producirse si se utilizan mal los recursos compartidos y no se reservan de forma adecuada, la asignatura de Sistemas Distribuidos también ayudó a coger conceptos como los *RPC (Remote Procedure Call)* y *RMI (Remote Method Invocation)* en el caso de programación orientada a objetos. Aunque pueda parecer que las invocaciones remotas no se utilizan en las aplicaciones web lo cierto es que para la ejecución de tareas pesadas en *background* en los denominados *workers* lo que se hace es precisamente esto, se llama a procesos que se programados en el lenguaje en el que se trabaja (o a veces en otros) y que se ejecuta de forma, en este caso, asíncrona. Un ejemplo típico de este tipo de procesos sería el del envío masivo de emails, que se llamaría desde un método o procedimiento del código normal pero que se ejecutaría usualmente en otra máquina y de manera asíncrona descargando de esta manera al servidor web de tareas lentas o que consumen mucho recursos y a su vez se le permite a la tarea que hace la llamada de la tarea pesada terminar en poco tiempo y evitar así tiempos de respuesta en la web altos o incluso los *timeouts*.

Muchas de las tareas que se suelen realizar en *background* en las aplicaciones web suelen requerir que se lleven a cabo en paralelo o incluso de forma concurrente de modo que para poder realizar esto hay que conocer cómo. También hay que conocer principios como que el poner más *workers* o más máquinas, o más procesadores no es sinónimo de escalado proporcional, si un trabajo que haces en X lo haces con 2 procesos, eso no quiere decir que el

trabajo se vaya a realizar en X/2 de tiempo, paralelizar de un modo adecuado es importante porque de realizarse de forma incorrecta puede incluso empeorar el rendimiento de la aplicación. Aparte de eso es importante también tener en cuenta conceptos como los recursos críticos y como evitar que varios procesos intenten acceder a ellos al mismo tiempo, para ello hay que utilizar variables de bloqueo *mutex* y semáforos para avisar a los demás procesos que un recurso está siendo utilizado y para avisar al resto de procesos de cuando el recurso está disponible para usarse, esto se relaciona mucho con parte de lo aprendido en Sistemas Operativos y que se ha comentado con anterioridad, pero que son cosas muy importante y a tener en cuenta tanto en la concurrencia como en la paralelización ya que pueden generarse incongruencias en los datos y errores. Por lo tanto hay que saber cuándo es conveniente paralelizar y tener procesos concurrentes que partes del código son susceptibles de realizarse de forma no secuencial o incluso asíncrona, a todo ello y a conocer el grado de paralelización óptimo para cada problema es entre otras cosas a lo que me enseñó la asignatura de Arquitectura de los computadores II y dichos conocimientos me han sido muy útiles a lo largo de mi vida profesional.

Como nuevos conocimientos adquiridos, he tenido que aprender la configuración y despliegue de aplicaciones *Rails* en servidores, así como la configuración de *Nginx* para poder redirigir las peticiones dentro del servidor a los diferentes servicios y aplicaciones que se tengan instalados en él. También ha sido necesario aprender a trabajar con máquinas virtuales dentro de un servidor. A parte de esto, que va enfocado fundamentalmente con la parte de configuración y despliegue de aplicaciones, con respecto al desarrollo de aplicaciones en sí, ya tenía ciertas bases en el desarrollo en *Ruby on Rails*, pero este proyecto me ha ayudado a aprender a crear un proyecto de cero con este *framework*, así como me ha ayudado a mejorar mi habilidad desarrollando en Ruby, además para la realización de la parte de *frontend* de la aplicación he tenido que aprender a utilizar varias librerías de *javascript*, la primera y fundamental es *jQuery*, que es en la que se suelen basar el resto de librerías, además para la visualización general se ha utilizado *bootstrap*, que es un paquete de estilos y *javascript* básico que mejora las los componentes mínimos de HTML5 e incluye algunos nuevos. Pero de todo lo aprendido durante el Proyecto Fin de Carrera lo más complicado ha sido aprender a utilizar la librería *d3js*, la librería de *javascript* utilizada para la generación de las gráficas que se pueden visualizar en la aplicación, fue complicado porque no sólo se tenían que mostrar las diferentes gráficas, si no que debían trabajar todas con los mismo datos y a la vez se tenía que permitir una transición no traumática (con recarga de página) entre los diferentes tipos de gráficas disponibles.

Por último habría que centrarse en diferentes mejoras o cambios que podrían realizarse sobre el proyecto llevado a cabo ya sea como base para nuevos proyectos y funcionalidades como para mejorar la aplicación que ya está terminada pero que pese a que podría contar con más cosas no se han incluido en esta versión por diversas razones.

Primero, se podría hacer que la actualización de los servidores que se tienen monitorizados se realizase de una forma automática teniendo lugar cada vez que se cumpliese un periodo preestablecido, de este modo el usuario dispondría siempre de información más o menos actualizada cuando entrase en la aplicación sin necesidad de llevar a cabo ninguna acción, aunque tampoco se ha investigado mucho en ello para este proyecto dado que

produce que no se tenga información actualizada en el momento que el usuario final realmente quiera obtenerla, en cualquier caso se podría investigar sobre una manera de mantener ambos sistemas de una forma simultanea de forma que el usuario final pueda tener información más o menos actualizada y al mismo tiempo si necesita que la información se procese en el momento pueda solicitarla. Otros sistemas lo que hacen es instalar *disparadores* en la aplicación destino que se reciben en el la aplicación monitor, pero esta es una solución intrusiva y que requiere de un desarrollo en los diferentes lenguajes de las aplicaciones que se quieran monitorizar.

Otra posible mejora que se le podría incluir a este sistema sería la inclusión de mecanismos de autenticación, de este modo un usuario cualquiera no tendría acceso a todos los servidores incluido en la aplicación, se podría implantar un sistema de roles y permisos que permitiesen incluir servidores que sólo pudiesen ser vistos por el usuario que los creo, de esta manera no sería necesaria una instancia de la aplicación para cada usuario que quisiese utilizarla, en cualquier caso se acordó que para esta primera versión del proyecto la autenticación se realizaría a través de http de modo que *Ruby* no tomaría parte. De todos modos existen gemas que se encargan de la autenticación como *Devise* y otras que se encargan de la gestión de roles y permisos como por ejemplo sería el caso de 'can-can-can'.

Otra mejora que vendría muy bien para esta aplicación sería el incluir filtros en la visualización de las gráficas de modo que por ejemplo, si se quieren visualizar sólo los datos de determinado servidor o servidores se vea la información solo de esos en la gráfica y no de todos, y en el caso de estar en la visualización de un servidor concreto, que el filtro se pudiese realizar sobre los diferentes usuarios que han utilizado el servidor en el periodo de tiempo que se esté mostrando en el momento.

Además de las mejoras habría que tener en cuenta otro ámbitos en los que se podría utilizar la aplicación aparte de su función original que era la monitorización de conexiones *ssh* para los ordenadores de la aulas informáticas, de forma que pudiese comprobarse que ordenadores tienen un uso mayor de modo que se pueda tener un seguimiento más de cerca porque será más susceptible de tener una avería, o necesitará ser sustituido con mayor celeridad en caso de tener problemas. Otro posible ámbito de uso distinto al original podría ser la monitorización de servidores pero en lugar de centrarse en el uso de forma comparativa con otros servidores que cumplen características similares se puede enfocar por ejemplo en servidores *git*, para ver las conexiones que se le realizan y estudiar comportamientos como horas a las que la gente suele hacer *push* al repositorio o la frecuencia con la que la gente sube o baja su código del repositorio, el número de conexiones media que realiza un usuario de *git* cuando quiere trabajar con el repositorio, ya sea para obtener información o para actualizarlo.

Otro posible ámbito relacionado con el anterior podría ser más enfocado a seguridad, dado que la aplicación CONAD monitoriza conexiones *ssh* a un servidor, si el servidor tiene información sensible se puede ver las conexiones que se han realizado a él quién las ha realizado y a qué horas de modo que sería sencillo tener controlado el uso de esos registros restringidos de forma que si hay algún uso indebido de ellos se pueda encontrar de manera rápida y eficaz a la personas o personas responsables.

Una mejora que ya conllevaría realmente un desarrollo completo sería el poder incluir otro tipo de conexiones a la monitorización de modo que no solo fuesen *ssh* sino que también pudiesen ser http o de errores, pero esto supondría la generación de librerías que se conectasen con la herramienta cuando se produjesen errores o conexiones, o si no definir un API, para que la gente pueda conectar sus aplicaciones web a la herramienta, pero como se ha dicho esto supondría cambios de base en el planteamiento del proyecto aunque bien es cierto que gran parte de lo existente podría ser reutilizable, es lo que se ha descrito como posibilidad para tener datos en tiempo real en la aplicación.

Otra posible mejora también podría ser la eliminación de datos después de un tiempo determinado para evitar llenar la base de datos con información obsoleta o que no se va a volver a consultar, se puede programar una tarea que limpie los registros de conexiones con una antigüedad mayor a un valor que se haya determinado con anterioridad.

Todas las ideas y conceptos de los que se ha hablado y descrito durante este resumen son expuestos y desarrollados de una forma mucho más intensiva y extensa a continuación a lo largo del resto del documento.

1.- Introducción

1.1 Motivación

Este proyecto surge de la necesidad de conocer el uso de los ordenadores de las salas informáticas, en qué horarios se utilizan más, qué ordenadores tienen mayor afluencia de usuarios o qué usuarios utilizan los ordenadores con una mayor regularidad de una forma que no se precise la instalación de software específico para dicha monitorización. La forma de conseguir esto es a través del comando *last* de Linux, que devuelve el registro de conexiones al equipo en el que se ejecute.

Tras hablar con mi tutor de proyecto y tras comenzar yo el desarrollo de dicha aplicación se llegó a la conclusión de que había que almacenar los datos SSH del servidor ya que con ellos se podría conectar con el servidor a monitorizar y en él ejecutar el comando *last* como se indicó con anterioridad. La primera gran dificultad que se presentó fue cómo almacenar las claves *ssh* de una forma segura, pero una vez solventado el problema se vio que se podía hacer lo mismo pero para varios servidores diferentes, puesto que la única diferencia era tener que almacenar más datos, y además teniendo en cuenta que los administradores de sistemas suelen gestionar más de un servidor nos pareció un paso lógico para la evolución de la aplicación el poder monitorizar varios en la misma aplicación web.

Al cambiar el planteamiento de uno a varios servidores, la visión de administrador de usuarios para un servidor perdió fuerza, dado que al ser varios servidores el número de usuarios puede llegar a ser tan elevado que no sea manejable por el administrador, además los usuarios de un servidor no tenían por qué coincidir con los de otro, y dado que el manejar los servidores de una forma independiente entre uno y otro no parece muy práctica para el administrador de un sistema con varios servidores, al final se ha hecho de tal manera que de una forma sencilla y visual el administrador del sistema se puede hacer una idea de la carga de trabajo de los servidores a su cargo, para de este modo facilitarle la toma de decisiones con respecto a la distribución de trabajo, de cada servidor así como de la distribución de recursos o su eventual ampliación. Así pues y para evitar el hecho de mostrar muchos usuarios de golpe la solución adoptada ha sido la de que el administrador pueda seleccionar un servidor, y obtener de él la misma información que en la vista general pero para el conjunto de usuarios que han usado ese servidor en lugar de para todos los servidores registrados en la aplicación.

Para llevar a cabo estos objetivos se decidió crear una herramienta web basada en *Ruby on Rails* que fuese sencilla de utilizar para el administrador pero que al mismo tiempo le proporcionase información precisa y útil con la gestionar los servidores, para obtener estos resultados la aplicación muestra distintas gráficas en las que se puede apreciar las conexiones *ssh* a los servidores monitorizados durante un mes natural, tanto en valores absolutos como en comparación con el uso del resto para comprobar a simple vista que servidores tienen mayor uso y cuales tienen un uso menor, y de esta forma tomar decisiones, además cuenta con un calendario para poder comprobar el desglose de uso por horas de los servidores en un día concreto.

Es cierto que actualmente ya existen otras herramientas para el control de conexiones a servidores, pero no son tan sencillas e intuitivas para el usuario final, a la vez que dan una

información muy completa para cubrir las necesidades del administrador del sistema, tanto de carga en un mes como en un día o incluso por horas para poder llevar una estrategia mejor e incluso predecir posibles picos de uso que puedan sufrir los servidores que gestiona a partir de los históricos de los que ya dispone.

1.2 Objetivos

En un principio los objetivos de la aplicación a desarrollar como se ha descrito con anterioridad son los mismos que los propuestos en un principio con el añadido de una visión general del conjunto de servidores que se monitorizan y además la posibilidad de gestionar varios servidores simultáneamente en lugar de únicamente uno.

1.2.1 Obtención de datos

La aplicación web que va a desarrollarse debe obtener los datos de conexión de los distintos servidores que se quiere monitorizar con ella. Para la obtención de dicho datos se va a utilizar el comando *last* de Linux, por lo tanto por ahora el sistema CONAD sólo podrá usarse con servidores Linux o que tengan implementado el comando *last* de una forma similar a este sistema operativo.

Para la obtención de las distintas conexiones realizadas en un servidor es necesario un usuario y una contraseña *ssh*, para poder obtener la salida del comando *last* en el servidor, después la aplicación se encargará de procesar esta salida y convertirla en conexiones para luego poder trabajar con ellas y cumplir así con el resto de objetivos.

1.2.2 Gestión de servidores

Dado que para conseguir la información de los distintos servidores hace falta almacenar información sobre ellos, esta información tiene que ser accesible para el usuario final de la aplicación dado que puede necesitarla, para la creación de un servidor nuevo dentro de la aplicación serán necesarias, la *URL* del servidor, así como un usuario *ssh* con permisos para ejecutar el comando *last* con su correspondiente contraseña, también es necesario introducirle un nombre para poder diferenciarlo del resto de servidores ya introducidos en la aplicación.

Estos datos una vez introducidos serán accesibles para el usuario y estarán a su disposición para su modificación en caso de ser necesario, también se permite eliminar el servidor, hacer esto implica eliminar el servidor así como todas las conexiones que se tengan almacenadas de este.

1.2.3 Calendario

La aplicación dispondrá de un calendario que mostrará el mes en el que se encuentre navegando el usuario en ese momento. Para cada día dentro del calendario el usuario tendrá acceso a los datos por horas de conexión a los distintos servidores de los que se componga la aplicación.

Para representar estos datos de conexión por horas se servirá de las gráficas de las que también dispondrá la aplicación y que vienen descritas como el siguiente objetivo de la aplicación web en desarrollo.

1.2.4 Gráficas comparativas

Dado que la información se le quiere dar al usuario final de una forma sencilla visual e intuitiva, uno de los objetivos será el mostrar gráficas comparativas que le permitan extraer conclusiones de una manera rápida con un primer vistazo.

Además los datos deberán ser representados con diferentes tipos de gráficas, de modo que el usuario final de la aplicación puede elegir entre las posibilidades que tiene la gráfica que le permita en un momento dado obtener las conclusiones que quiere alcanzar para la toma correcta de decisiones basadas en los datos de que dispone.

Las gráficas deberán ser tanto para el mes natural en el que se encuentre en ese momento, así como guardar historiales y gráficas de los meses pasados para que el usuario pueda volver sobre ellos si lo considera necesario. También como se ha descrito con anterioridad se tendrá acceso a las gráficas diarias por horas a través del calendario.

1.2.5 Multilenguaje

La aplicación web en desarrollo debe contar con al menos 2 idiomas en los que poder trabajar con ella, en este caso inglés y castellano, dado que inglés es el lenguaje estándar para el uso de internet y en castellano dado que es la lengua utilizada en la Universidad Carlos III de Madrid.

Además de estos 2 idiomas por defecto se debe especificar y facilitar la posible inclusión de nuevos idiomas en futuras mejoras que se le quieran realizar a la aplicación en desarrollo.

Para hacer más sencillo saber en qué lenguaje se está utilizando la aplicación y dado que en esta versión no se almacenan datos de usuario y sesión, el idioma en el que se use vendrá dictado en la URL en forma de *locale* (por ejemplo para castellano la *URL* raíz sería http://conad.staging.kotraders.com/es) en caso de no aparecer ningún *locale* se toma como idioma por defecto para el uso de la aplicación el inglés, además se incluirá un selector de idiomas en el *navbar* por si el usuario no se siente cómodo con el idioma en el que se encuentra en ese momento.

1.3 Resto del documento

Dado que esto forma parte de la introducción se va a proceder a la exposición del resto del documento, de las partes que lo componen y los conceptos que van a ser introducidos en cada una de ellas.

En el capítulo del estado de la cuestión se describirán el estado actual de las aplicaciones web para la monitorización de conexiones para varios servidores, en esta parte del documento se va a explicar que otras aplicaciones hay que puedan ser parecidas a la que se va a desarrollar, que aportan esas soluciones y que aporta *CONAD* con respecto a las otras y porque es una mejor solución que el resto ya existentes. Esto se logra a través de explicaciones de las distintas herramientas en el mismo nicho de mercado y también con tablas comparativas entre ellas y la solución propuesta.

Después se va a realizar una descripción detallada de todas las fases del desarrollo software para esta aplicación, desde la elección de una metodología de desarrollo adecuada para la realización de una aplicación web de las características ya descritas, así como la obtención de los requisitos tanto funcionales como no funcionales que debe cumplir la aplicación. Tras esto se realizará un diseño apropiado que cumpla tanto los objetivos descritos en la introducción así como los requisitos obtenidos del análisis del problema al que se quiere poner solución, luego se describirá como se ha realizado el desarrollo propiamente dicho, y las principales dificultades que se han tenido que superar para conseguir crear un buen producto software y finalmente en esta parte del documento se realizará una descripción de la implantación de la aplicación software para que de una forma sencilla cualquiera pueda disponer de ella y así como un manual de uso para que los usuarios finales sepan cómo manejarla una vez hecha la instalación.

Tras esto se procederá a la evaluación del producto final para comprobar si el sistema tiene un rendimiento apropiado para las funciones que ha desarrollar y si los usuarios finales están satisfechos con la aplicación, comprobar si cumple también con todo lo previsto en las secciones anteriores del documento.

En la parte de planificación se va a realizar una división del trabajo a realizar en pequeñas tareas que luego se representará su ejecución a través del tiempo por medio de un diagrama de Gantt y que se llevarán a cabo sirviéndose de las herramientas propias que los repositorios GIT dan para el manejo de pequeñas tareas a través de *Issues*, así como un presupuesto del coste total que tendrá el desarrollo de este producto software teniendo en cuenta tanto los gastos por personal como los derivados de las herramientas de las que se valdrán los desarrolladores para por producir la aplicación web final.

Por último se procederá a emitir las conclusiones que se han extraído a partir del desarrollo del producto, que incluirán tanto sobre el producto en sí, si se ha conseguido todo lo que se esperaba y pretendía de la aplicación, también conclusiones sobre el proceso de desarrollo para saber si se tomaron las decisiones adecuadas para la producción de la aplicación web requerida y los problemas surgidos de la planificación, también se sacarán conclusiones sobre los conocimientos adquiridos durante el estudio de la carrera universitaria han ayudado al desarrollo y también sobre los nuevos conocimientos que se han adquirido durante la realización de este proyecto con los que no se contaba antes del inicio y que han

sido necesarios para el correcto desarrollo del producto software final y que han servido para aumentar los conocimientos adquiridos durante sus estudios.

Durante las conclusiones también se tratarán los posibles trabajos futuros que se puedan llevar a cabo sobre la aplicación producida, tanto si de una forma sencilla, o no tan sencilla, la aplicación estaría abierta a posibles mejoras o incluso si desde otro enfoque podría ser utilizada para otros ámbitos o por otros usuarios que difieran de los planteados inicialmente para su uso y para los que en un principio podría estar diseñada.

Por último el documento contará con un apéndice en el que se incluirán los manuales tanto de usuario, para que los futuros usuarios de la aplicación sepan utilizarla y sacarle el máximo partido de modo que no pierdan nada de funcionalidad al usarla, aunque debería no ser necesario porque una de los objetivos de esta aplicación es que sea intuitiva para el usuario final. Además de este manual de usuario se adjuntará uno de instalación para que cualquier administrador de sistemas web pueda disfrutar de una copia funcional de la aplicación en su propio servidor.

2.- Estado de la cuestión

2.1 Introducción

Actualmente hay muchos servicios de monitorización de servidores, pero la mayoría de ellos están orientados al seguimiento de los *WebServices* que el servidor proporciona a los diferentes usuarios del sistema. Lo que aquí se propone es una idea completamente distinta en lugar de hacer un seguimiento (tracking) de las distintos servicios que los usuarios finales hacen del servidor lo que se hace es procesar el log de conexiones SSH (como se ha dicho el comando *last* de Linux) y de ahí obtener información sobre las conexiones que se han realizado. Una posible aplicación de la herramienta desarrollada podría ser para cada ordenador de un aula informática ver que usuarios se han conectado y en qué horas, y usando el monitor de servidores se podría ver que máquinas del aula son las que tienen mayor uso.

Como ya se ha dicho al ser una herramienta diseñada para cubrir una necesidad muy concreta es complicado encontrar otras herramientas parecidas con las que comparar, de modo que se ha tomado la determinación de compararlo con otros sistemas de monitorización que aunque puedan tener características u objetivos diferentes tienen lo suficiente en común como para poder ser comparables.

2.2 New Relic, Inc. (01, s.f.)

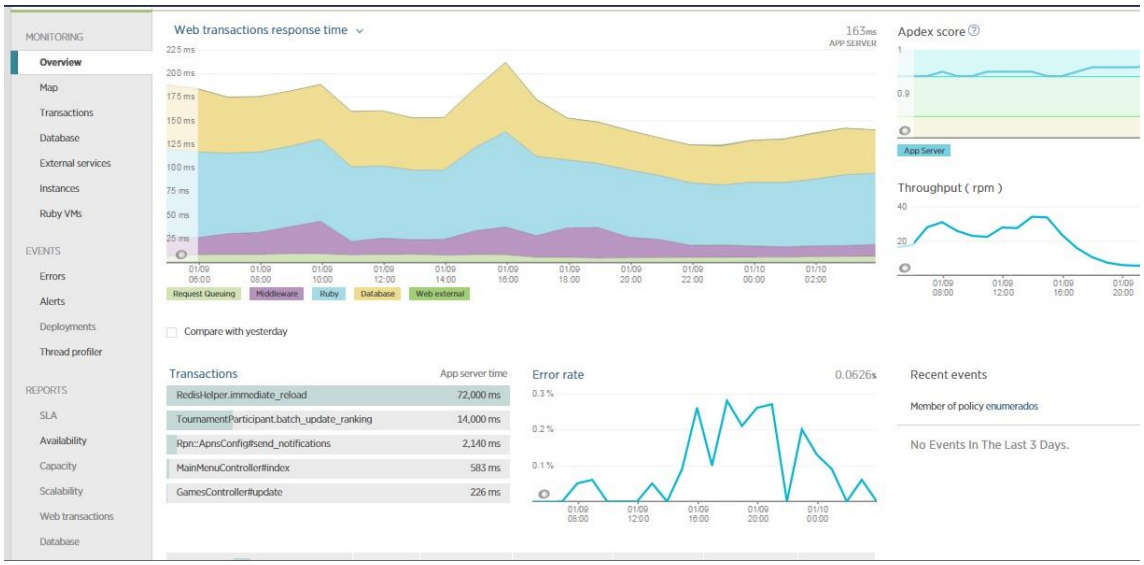

Ilustración 1 – Dashboard de New Relic

La ilustración 1 muestra un ejemplo de monitorización utilizando *New Relic*, en este caso se refiere a una aplicación en *Ruby on Rails*, y sólo con la vista general se tiene una visión general bastante completa de los parámetros que necesita conocer el administrador de un sistema web basado en servicios.

Se ven en la primera gráfica los tiempos de respuesta del servidor sobre el tiempo, además según el color se ve que parte de cada tiempo se lleva cada uno de los procesos que se están corriendo. En amarillo se ve la parte de la respuesta que se invierte en llamadas a la base de datos, mientras que en azul se ve el tiempo que lleva el programa en sí, en este caso al ser

un servidor en *Ruby on Rails*, el azul representa el tiempo de ejecución del código en Ruby. En morado se ve el tiempo de procesado en el *middleware* en este caso son operaciones sobre la caché en *Redis*. Y por último se ve la parte que se utiliza en procesar las tareas pendientes. En este caso no hay nada en verde oscuro porque este servidor concreto no realiza peticiones a servicios externos.

A la derecha de la Ilustración 1 se puede observar la gráfica de peticiones por minuto frente al tiempo, es un gráfico linear que permite de una forma sencilla ver las horas punta de peticiones al servidor para saber si hay que tomar algún tipo de medida especial a esas horas para evitar el colapso del sistema o lentitud en el servicio que pueda repercutir en su calidad.

Además de esto en la imagen anterior se observa que en la parte inferior otra gráfica que muestra la tasa de errores que se ha producido en el servidor según la hora en la que se ha producido. Si se clica en esta gráfica o se va a través del menú de la izquierda a errores se llega a una vista en la que puede observarse la misma gráfica pero en un tamaño mayor así como los distintos errores que se han ido produciendo a lo largo del tiempo, y el número de veces que se ha producido el mismo error, además también se puede ver la traza del error cuando se produjo.

2.3 Google Analytics (02, s.f.)

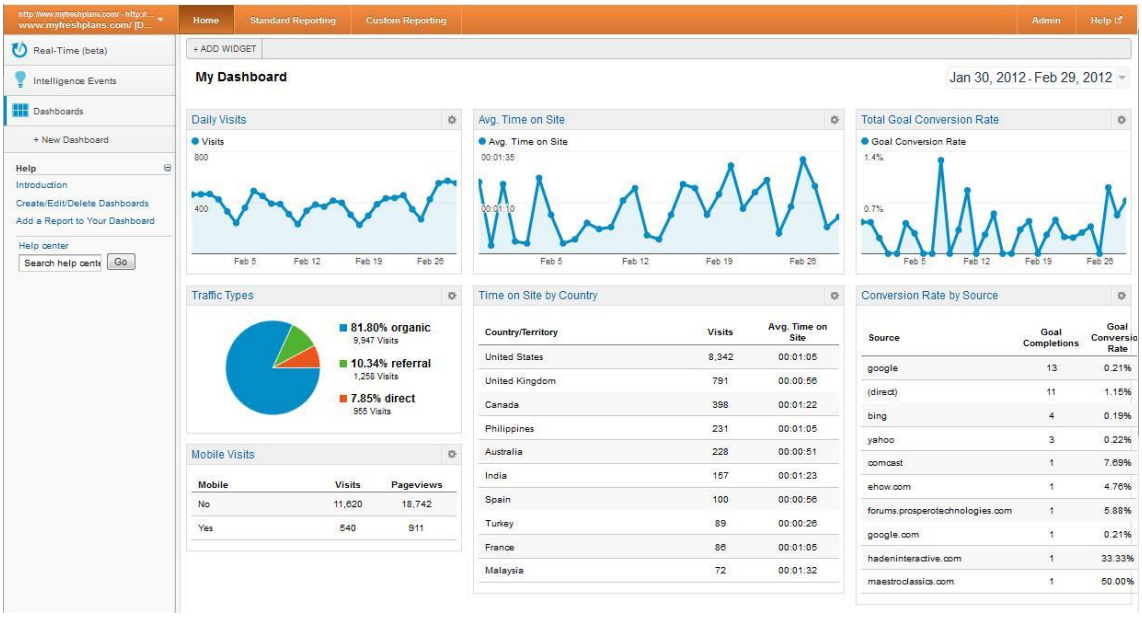

Ilustración 2 – Dashboard de Google Analytics

En esta ilustración puede verse una captura de pantalla de menú de *Google Analytics*, arriba a la izquierda se nos muestra el total de visitas diarias para cada día del mes en valores absolutos, en este caso las vistas diarias están entre 400 y 600 aproximadamente. Estos valores son independientes del navegador o el dispositivo desde el que se accedió.

La gráfica a la derecha de la que se acaba de describir muestra el tiempo medio que cada usuario ha estado en la página, el tiempo en el que se puede ver la evolución de dichos

datos es configurable, en el caso de la ilustración 2, la gráfica muestra el tiempo medio diario de estancia en la web en un mes.

Entre otras cosas también puede verse los diferentes tipos de tráficos que tiene la página, ya sea orgánico, de referencia o directo, mostrándolos de forma muy visual con un gráfico de tarta en el que cada porción es una parte proporcional del total, para diferenciar un tipo de otro se utilizan distintos colores que los representan. Además debajo se muestra una tabla que representa del total de conexiones al sitio web cuantas se han realizado desde dispositivos móviles.

Otra información que se puede obtener a través de *Google Analytics* son los países desde los cuales la gente se conecta al servicio web que se está monitorizando con dicha herramienta, se muestra el número absoluto de conexiones desde cada uno de los países desde los que se haya accedido. Además junto a los países de procedencia de las conexiones se puede ver también desde donde se ha accedido, a través de que buscador concreto o si han sido conexiones directas.

2.4 Nagios (03, s.f.)

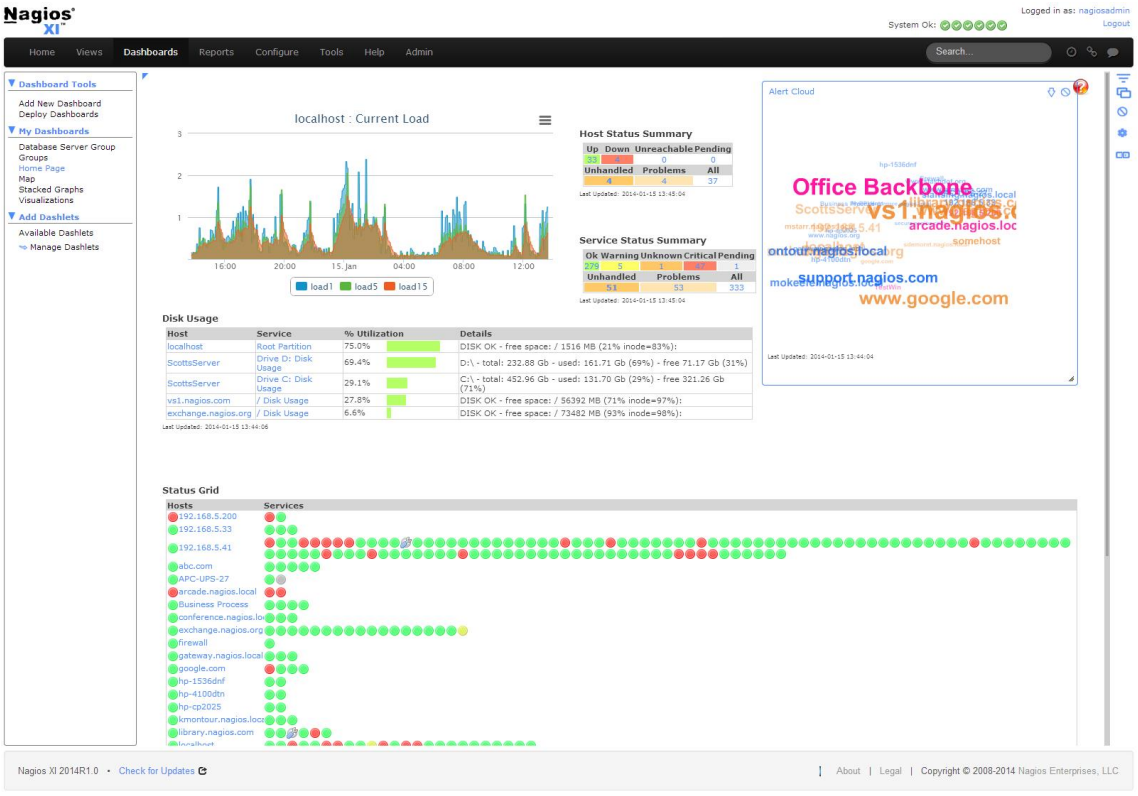

Ilustración 3 – Dashboard de Nagios

Nagios es una herramienta de monitorización de servidores, pero a diferencia de las otras 2 vistas hasta ahora esta se centra más en el tipo de información que necesita un administrador de sistemas para conocer el estado de sus servidores, mientras que las otras están focalizadas en obtener toda lo que se pueda conocer de las conexiones que se han realizado a la páginas web registradas, que páginas se han visitado más desde que países o dispositivos.

Con *Nagios* la información que se procesa es la relacionada con el uso de los recursos físicos y el estado de los diferentes servicios web (*Web Services*, W.S.), indicando si las diferentes máquinas están proporcionando todos los servicios que se le suponen, o si por el contrario tiene servicios caídos o la propia máquina se encuentra inhabilitada.

En la gráfica que se ve en la ilustración 3 se puede observar que representa la carga del servidor (en este caso *localhost*) según las distintas horas del día. Con esto se puede ver de una forma muy intuitiva las horas punta de tráfico del servidor y saber si está trabajando bien, o si por el contrario se encuentra sobrecargado en algún momento.

A la derecha de la gráfica hay 2 tablas, una indica el estado de los distintos nodos que se están monitorizando, el número de que están *OK*, caídos o que tienen problemas de red y no son accesibles. Además en la otra tabla se muestran todos los *W.S.* que se están monitorizando y de ellos cuantos son funcionales, cuantos tienen algún aviso y hay que revisar, cuantos están en un estado desconocido por el sistema y cuantos se encuentran en estado crítico o caído.

Bajo la gráfica y estas tablas hay otra que contiene la información del estado de uso de disco de cada una de las máquinas del sistema indicando el porcentaje de utilización de modo que sea sencillo ver que máquinas se están quedando sin espacio de almacenaje y hay que ampliar para que puedan continuar prestando servicio indicando los MB y GB utilizados y libres para que el administrador sepa exactamente el estado de uso de los discos en sus servidores.

Finalmente hay una última tabla que muestra el estado de los *WS* que están siendo monitorizados por la herramienta, en ella se muestra primero la *IP* o dominio del nodo encargado de prestar el servicio y luego siguiendo un sencillo código de colores se muestran en verde los servicios que están disponibles en ese nodo, en rojo los que están inhabilitados o caídos y en gris los que son inaccesibles para la herramienta.

2.5 Site24x7 (04, s.f.)

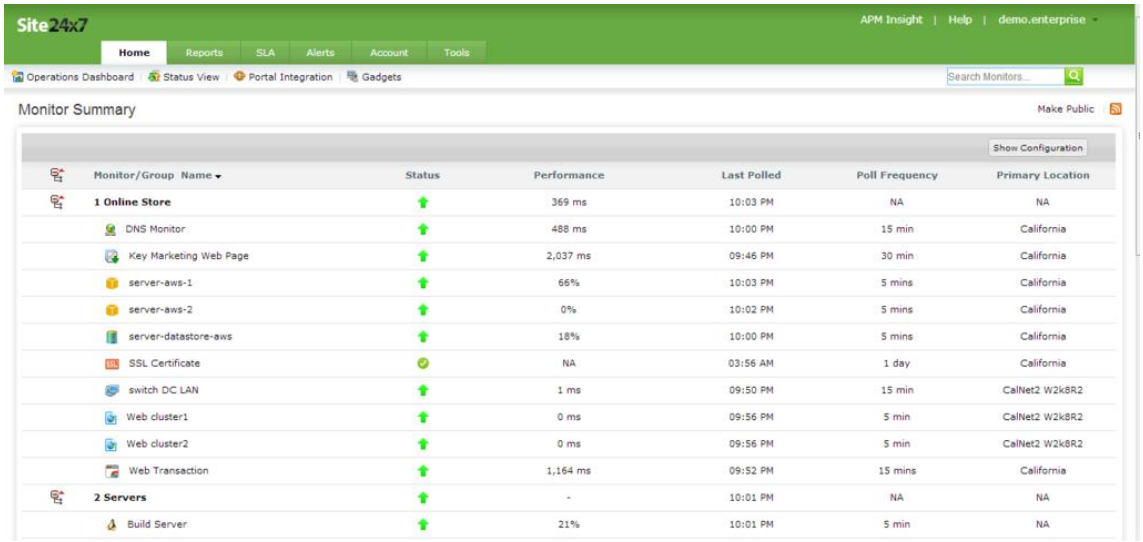

Ilustración 4 – Dashboard de Site 24x7

En esta ilustración puede verse de una forma general las principales funciones de site24x7. En esta vista se ven todos los servidores que se están monitorizando en la herramienta así como información general de ellos. Desde el estado hasta su localización geográfica.

En esta primera (Ilustración 4) imagen se ve el nombre y el tipo de servidor que se está monitorizando, tras esto su estado (si está funcionando o si por el contrario se encuentra caído) indicado con una flecha verde hacia arriba si está *OK* y con una roja hacia abajo si está *KO*, también se muestra el comportamiento del servidor de acuerdo al tipo de servidor del que se trate, ya sea a través de porcentajes o el tiempo de respuesta en milisegundos.

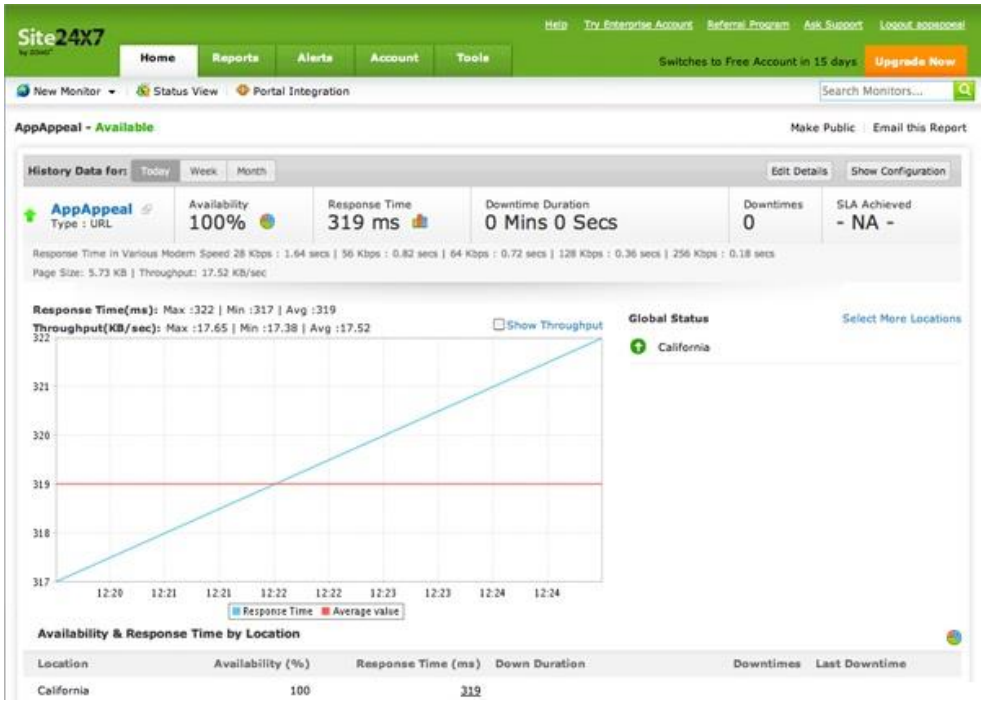

Ilustración 5 – Visualización de un servidor en Site 24x7

En la siguiente ilustración (Ilustración 5) se ve el detalle de uno de los servidores monitorizados, primero se puede elegir entre el periodo de tiempo en el que se quiere ver las estadísticas, el día de hoy, la última semana o el último mes.

Luego se muestra el nombre del servidor que se está visualizando así como su estado con los mismos símbolos que en la ilustración anterior, es decir, con una flecha verde hacia arriba si está *OK* o con una roja hacia abajo si está *KO*. Tras esto se muestra el tanto por ciento del tiempo que el servidor ha estado disponible en el periodo de tiempo que se está observando, el tiempo de respuesta en milisegundos (ms), el tiempo en que el servidor ha estado caído, así como el número de veces que ha caído.

Por último se puede comprobar una gráfica que indica la evolución del tiempo de respuesta en el periodo de tiempo seleccionado y su vez una comparación en el tiempo medio de respuesta también para ese tiempo, de modo que se pueden comprobar a la vez de un solo vistazo el estado de carga del servidor a través de los tiempos de respuesta, y también la evolución de estos con respecto a la media.

Al final de esta vista se dice la localización física del equipo, así como disponibilidad en tanto por ciento, el tiempo de respuesta, el tiempo que ha estado caído en el periodo de tiempo, además se ve el número de veces que el servidor no ha estado disponible así como el momento de la última vez que dejó de estar disponible.

2.6 Paessler (05, s.f.)

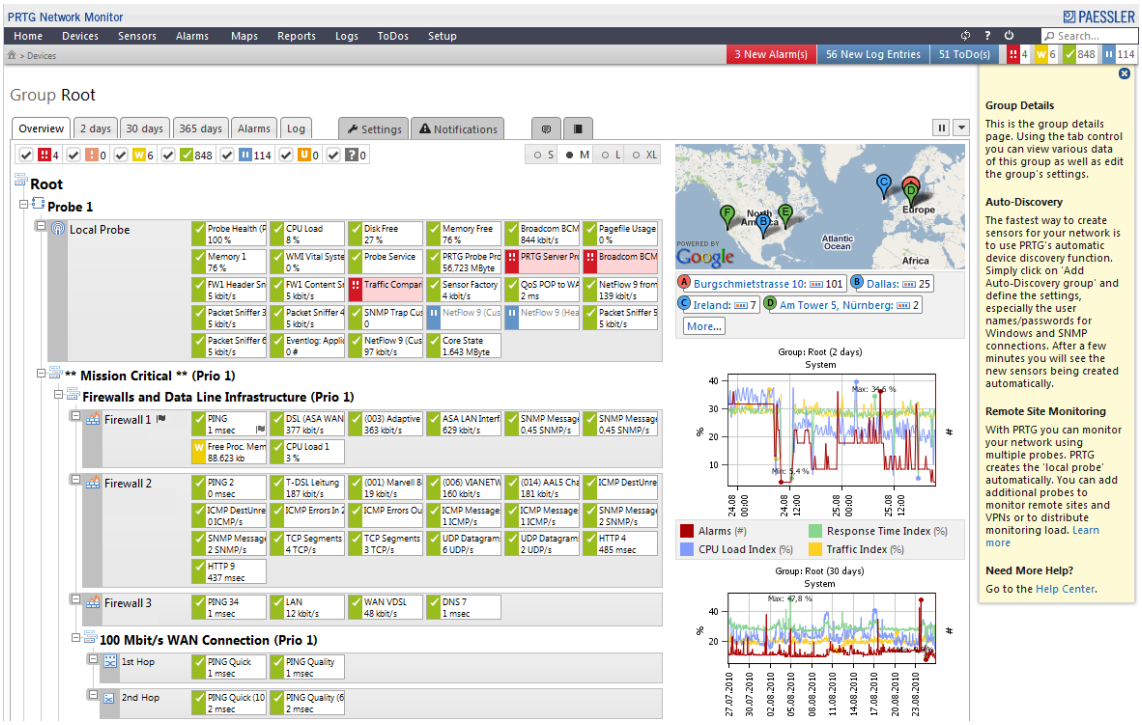

Ilustración 6 - Dashboard de Paessler

Esta herramienta está centrada fundamentalmente en la monitorización de red, esto quiere decir que se centrará en tiempos de respuesta en estado de conectividad con los distintos nodos de la red, así como tráfico y localización geográfica de los distintos elementos que componen el sistema que se quiere controlar.

En esta herramienta la monitorización es llevada a cabo por lo que ellos llaman *sensores*. Los sensores se manejan a través de una jerarquía en árbol con la que se le permite una navegación sencilla e intuitiva al administrador agrupando los sensores para dispositivos, localizaciones o servicios similares. El administrador puede crear grupos anidados que contengan varios dispositivos y que estos a su vez tengan varios sensores asociados.

En esta vista puede observarse los distintos grupos de los que se ha hablado anteriormente que contienen los sensores, que al final son la base de la herramienta. El primer caso de la ilustración es un dispositivo que contiene una serie de sensores que serán los que se evalúen. En verde con un ✓ significa que el sensor no presenta ningún tipo de inconveniente y que por tanto eso que se está monitorizando funciona correctamente, por el contrario si el color es rojo con dos !! significa que hay algún tipo de problema con el sensor y que lo que se está monitorizando tiene algún tipo de error, en caso de estar funcionando pero con algún tipo de aviso para comprobar estará señalizado con una W de color amarillo y por último está la

posibilidad de || en color azul que significa que ese sensor se encuentra pausado y que por tanto no está recogiendo datos.

En un lateral también se muestran gráficas con el número absoluto de alarmas, el tanto por ciento de carga de las CPU, así como los porcentajes de tráfico para la carga del índice y de tiempo de respuesta para las peticiones que se le realizan al sistema que se está monitorizando.

2.7 Tabla comparativa con las distintas soluciones y la propuesta

Para esta tabla se va a utilizar el siguiente código para indicar si una de las propuestas cumple completa, parcialmente o no cumple con una de las características de las que dispone la solución propuesta para este proyecto. En caso de tener la característica de forma completa se indicará con el símbolo ✓ en verde, si lo cumple parcialmente se indicará con el símbolo ≈ en amarillo, y si no lo cumple en absoluto se indicará × con en rojo.

	Gratuito	Visualización gráfica de carga del servidor	Visualización gráfica de evolución de uso frente al tiempo
Conad	✓	✓	✓
New Relic	≈ (Sin algunas funcionalidades)	✓	✓
Google Analytics	≈ (Sin algunas funcionalidades)	✓	✓
Nagios	×	✓	✓
Site24x7	×	×	✓
Paessler	≈ (Hasta 100 *sensores*)	✓	✓

Tabla 1: Comparativa soluciones – Gratuito – Carga – Evolución en el tiempo

	No necesita librerías en el servidor	Registro de conexiones ssh	Calendario
Conad	✓	✓	✓
New Relic	×	×	×
Google Analytics	×	×	×
Nagios	×	×	×
Site24x7	×	×	×
Paessler	×	✓	×

Tabla 2: Comparativa soluciones – Librerías – ssh - Calendario

	Visualización gráfica de uso para servidor y usuarios	Histórico desde el principio de los tiempos	Actualización bajo demanda
Conad	✓	✓	✓
New Relic	×	×	×
Google Analytics	×	×	×
Nagios	×	×	×
Site24x7	×	×	×
Paessler	×	×	×

Tabla 3: Comparativa soluciones – Gráfica uso – Histórico – Actualización bajo demanda

	Multiservidor	Multilenguaje	Distintas visualizaciones gráficas para los mismos datos
Conad	✓	✓	✓
New Relic	✓	✓	×
Google Analytics	✓	✓	×
Nagios	✓	✓	×
Site24x7	✓	✓	×
Paessler	✓	✓	×

Tabla 4: Comparativa soluciones – Multiservidor – Multilenguaje – Multiple visualización

De las tablas comparativas puede extraerse que la herramienta *CONAD* está diseñada para cubrir una serie de características bastante específicas, (y otra no tanto) que el resto de plataformas y herramientas no cubren, por ejemplo sólo *Paessler* además de *CONAD* registran conexiones *ssh* que se hacen al servidor cuando es una de la características principales que debía cumplir el sistema y es la base misma de todo el proyecto.

Puede observarse que pese al hecho de ser muy completas, el resto de herramientas analizadas salen perdiendo en características con *CONAD*, esto es debido a que para este proyecto hay un nicho de mercado poco explorado como es la monitorización de conexiones *ssh*. Pero aunque no sea una cosa explorada es de gran utilidad, la primera y por la que fue diseñada la aplicación es para las aulas informáticas de la universidad, con la monitorización que se proporciona se puede saber por ejemplo de forma muy intuitiva que usuarios se conectan a que máquina, con qué frecuencia y la duración de dichas conexiones.

Por todos esto, y como puede observarse en las tablas comparativas el crear una herramienta con estas características era necesario e innovador en un terreno tan manido como es la monitorización de sistemas, que además es uno de los más importantes en el área IT de cualquier empresa. Actualmente una empresa no puede permitirse los gastos directos e indirectos que pueden suponer el dejar de dar servicio a sus clientes durante un espacio prolongado de tiempo y tiene que ser capaz de prevenir en la medida de lo posible, los problemas que puedan ir encontrando, y para ello se pueden basar en datos históricos, y es por eso que en *CONAD* se almacenan y se clasifican utilizando un calendario para favorecer su acceso en cualquier momento.

De todas las características vistas en las tablas, una remarcable y que hay que explicar es la de actualización bajo demanda, en determinadas circunstancias podría considerarse una desventaja, pero dado que no se están realizando constante consultas a los servidores que se monitorizan no afectan para nada en su rendimiento dado que los históricos se obtienen en una única consulta cuando es necesario. Además gracias a que la actualización sólo se realiza bajo demanda la aplicación no necesita una gran cantidad de recursos ni un servidor potente para poder ponerla en funcionamiento.

3.- Análisis, diseño, implementación e implantación

3.1 Introducción

Para la realización de este proyecto de entre las diferentes metodologías de desarrollo conocidas durante tanto los estudios de la ingeniería como la vida laboral se ha optado por una metodología ágil, que a diferencia de las metodologías anteriores se basa según su manifiesto en los individuos y sus interacciones frente a los procesos y las herramientas, en hacer un software funcional frente a una documentación más elaborada y comprensible, en la colaboración con el cliente frente a la negociación de un contrato y sobre todo y más importante que sea un software que responda al cambio en lugar de ser *planeado* **(06, 2016)**.

Las razones para optar por esta metodología es porque es un producto que en lugar de ser planificado a priori es mucho más flexible y se adapta mucho mejor, no sólo a las necesidades del cliente, sino que además está más preparado a los distintos cambios que todo producto software sufre durante su vida funcional. En esta introducción se van a tratar los diferentes puntos del manifiesto que se han enunciado antes.

Individuos e interacciones frente a procesos y herramientas es importante en este caso porque al final quien va a tener que monitorizar servidores utilizando el producto final van a ser personas que interactúan con la herramienta, por lo tanto es en esos individuos en los que hay que pensar a la hora de desarrollar la herramienta, la mentalidad no es de cómo va a funcionar la aplicación sino de cómo la van a utilizar los usuarios finales, y en formas de hacerles más sencillo y útil el uso de la herramienta.

Hacer un software que funcione frente a una documentación exhaustiva, la explicación es bastante intuitiva, de nada sirve tener el mejor diseño del mundo con la mejor documentación en la que se vea pormenorizado hasta el último detalle de la aplicación si luego a la hora de la verdad no funciona o los que se describe es irrealizable. En algunos casos, merece la pena sacrificar una descripción más exhaustiva en la documentación y que funcione, aunque esto no significa que por ello la documentación no deba ser completa y cumplir con todo lo que se requiere a un proyecto de esta envergadura.

Anteponer la colaboración de los clientes a la negociación de los términos de un contrato, dado que lo que se busca es que el producto software que se entregue sea lo más útil para el cliente y que se adapte lo mejor posible a las necesidades de este, no tiene sentido hacer un diseño cerrado en el que se hable con el cliente al principio y en el que no participe, actualmente se hace imprescindible su colaboración no sólo como se ha dicho para que se adecue más a sus necesidades sino también para que el cliente sienta que forma parte del proceso de creación de la aplicación. Esto no sólo hace que se implique más que aporte ideas y que el resultado final sea lo más parecido posible a lo que el cliente tiene en mente, además de estas ventajas hace que a la hora de la implantación del nuevo producto esta sea menos traumática para el cliente dado que es una herramienta que ha ido viendo crecer y evolucionar y por tanto con la que está al menos en parte familiarizado con ella.

Y por último y más importante enfocar el producto a la respuesta frente al cambio en lugar de seguir un plan predefinido. Como se ha dicho antes es importante la colaboración del cliente, por tanto no es práctico para conseguirla el hablar antes de empezar con el proyecto con él, acordar unos términos y unas características inamovibles y no volver a verles hasta que se haya finalizado el desarrollo, y dado que el cliente va a cambiar de parecer con bastante frecuencia y que el entorno de la informática es un campo en constante cambio no es lógico

diseñar y producir cosas “estáticas” y resistentes al movimiento y la evolución, porque al final se está creando un producto que está obsoleto antes siquiera de existir. Antes lo normal era la programación de productos estáticos y robustos poco preparados para afrontar el cambio, este tipo de productos están muy bien para entornos cerrados y aislados poco influenciados por eventos externos.

El producto que se va a desarrollar es una aplicación web, para reafirmar los argumentos anteriores, el entorno web es un entorno cambiante y muy sujeto a cambios, no sólo del entorno sino en cuanto a cambios por parte del cliente, la competencia evoluciona y del mismo modo debe hacerlo el producto software para no quedarse atrás. Además no solo evolucionan las necesidades que debe cubrir la aplicación, también cambian las tecnologías en las que se desarrolla haciéndolas más flexibles.

Con todo esto también se justifica que el desarrollo se haya realizado de la siguiente manera, en este caso primero se creó un núcleo estable sin interfaz gráfica que se encarga de la obtención de las conexiones *ssh* a través del comando *last* y su almacenamiento en la base de datos. Tras esto en diferentes evolutivos se le han ido añadiendo el aspecto gráfico, la posibilidad de gestionar otros servidores, así como el resto de funcionalidades que se han descrito con anterioridad a lo largo de esta documentación. Es decir, la aplicación se ha ido realizando de forma incremental con una base bien definida a la que se le han ido añadiendo todas las características con las que contará la aplicación.

3.2 Análisis

3.2.1 Introducción

Ahora se va a proceder a hacer el análisis del sistema a desarrollar y de los requisitos que tiene que cumplir. Esta aplicación surge de la idea de obtener y procesar información del comando *last* de Linux que guarda un registro de las conexiones *ssh* que se producen a un servidor.

Partiendo de esta base se busca crear un producto que permita de una forma intuitiva mostrar la información obtenida. Se dará la opción de actualizar la información que tiene almacenada la aplicación bajo demanda del usuario de modo que estará actualizada cuando el usuario lo necesite sin estar consumiendo recursos el resto del tiempo.

También la aplicación contará con un calendario que permita ver históricos por meses o días de la información de conexiones almacenada, gracias a este calendario se podrá ver de forma sencilla la información que se tiene en la base de datos en un mes, además clicando en cualquier día del calendario se podrán ver los datos para ese día.

A través de esta aplicación web podrán almacenarse más de un servidor y de todos ellos se obtendrá información cada vez que el usuario actualice los datos. En la vista principal se verán distintos tipos de gráficas comparativas con las que podrá comprobarse que servidores tienen una mayor cantidad de conexiones y cuales menos, además se tiene que permitir ver para un servidor concreto de los que se gestionan estas mismas gráficas comparativas, pero en lugar de entre servidores se mostrará la relación entre los usuarios.

Además de estas características se va a permitir que el usuario final de la aplicación pueda seleccionar el idioma en que se le va a mostrar la información de modo que va a ser un programa multilenguaje que permita a través de una pestaña traducirlo, además de a través de la interfaz de la aplicación se va a permitir y gestionar el cambio de idioma a través de las *URL's*.

Ahora a partir de estas especificaciones se van a mostrar los requisitos funcionales y no funcionales que se extraen de ellas. Los requisitos funcionales son aquellos que describen como se comporta el sistema y sus componentes mientras que los no funcionales son aquellos relacionados con el resto de necesidades que tiene que cubrir la aplicación, como por ejemplo requisitos técnicos o de arquitectura.

A continuación se muestra una tabla de muestra de recogida de requisitos, con la información necesaria para la definición de cada uno de los requisitos tanto funcionales como no funcionales.

Nombre del Requisito			ID	RR-YYY
Prioridad	Alta/Media/Baja	Procedencia	Cliente	Eq. Desarrollo
Claridad	Alta/Media/Baja	Necesidad	Esencial	Deseable
Verificabilidad	Alta/Media/Baja	Estabilidad		
Descripción				

Tabla 5: Plantilla de tabla de requisitos

Los campos de la tabla deben ser completados siguiendo las siguientes definiciones:

- Nombre del Requisito: Este campo contiene el nombre corto de cada requisito de modo que describa brevemente el propósito del mismo.
- ID: Cada requisito dispondrá de un identificador único para permitir su rastreabilidad durante todo el proceso. El identificador tiene el siguiente formato, "RR-XXX", donde:
  - RR es el tipo de requisito, puede tener los valores "RF" si es un Requisito Funcional o "RNF" si es un Requisito No Funcional.
  - YYY es un número de tres cifras que comenzará por 001 y que identificará unívocamente a cada requisito de los distintos tipos.
- Prioridad: La prioridad para de un requisito se mide de acuerdo a lo que faciliten el desarrollo de las fases que le sigan y dependan de él. Puede tener los valores alto, medio o bajo.
- Procedencia: Se ha incluido la procedencia del requisito. Muchos de los requisitos vienen de la propuesta inicial del cliente pero también los hay que han surgido del equipo de desarrollo de la aplicación, para marcar la procedencia se indicará con una X junto al valor válido.
- Claridad: Este campo determina si un requisito no es ambiguo, por lo que será alta si la descripción del requisito no es vaga ni mal interpretable, y sólo tiene una posible y única interpretación. Los posibles valores que va a poder adquirir son alta, media o baja.
- Necesidad: En este campo se mide la importancia de la implementación de este requisito para poder garantizar el completo funcionamiento del sistema. En caso de ser un requisito esencial se marcará con una "X" a su lado. En cambio sí se trata de un requisito deseable, se le asignará un valor entre el 1 y el 3, siendo el 1 el nivel más alto de la característica.
- Verificabilidad: Se puede comprobar sí un requisito se ha incluido en la aplicación. Este campo mide como de sencillo es realizar esa comprobación. Los posibles valores que puede adquirir son alta, media o baja.
- Estabilidad: Un requisito puede ser necesario solo durante un periodo de tiempo concreto o durante toda la vida útil del proyecto. En este campo se pretende

establecer la longevidad del periodo de la vida del proyecto durante la cual el requisito tiene que estar presente.

- Descripción: Este campo contiene una explicación clara y concisa del propósito del requisito.

3.2.2 Requisitos funcionales

CONEXIÓN SSH			ID	RF-001
Prioridad	Alta	Procedencia	Cliente	Eq. Desarrollo X
Claridad	Alta	Necesidad	Esencial X	Deseable
Verificabilidad	Alta	Estabilidad	Durante toda la vida útil del proyecto.	
Descripción	El sistema tiene que poder conectarse por medio de ssh a otro servidor.			

Tabla 6: Requisito RF-001

OBTENER CONEXIONES			ID	RF-002
Prioridad	Alta	Procedencia	Cliente X	Eq. Desarrollo
Claridad	Alta	Necesidad	Esencial X	Deseable
Verificabilidad	Media	Estabilidad	Durante toda la vida útil del proyecto.	
Descripción	El sistema tiene que obtener las conexiones ssh del servidor al que se conecte.			

Tabla 7: Requisito RF-002

ACTUALIZACIÓN DE LAS CONEXIONES			ID	RF-003
Prioridad	Alta	Procedencia	Cliente	Eq. Desarrollo X
Claridad	Alta	Necesidad	Esencial X	Deseable
Verificabilidad	Media	Estabilidad	Durante toda la vida útil del proyecto.	
Descripción	El sistema tiene que obtener información actualizada de las conexiones ssh al servidor que se está monitorizando.			

Tabla 8: Requisito RF-003

OBTENER CONEXIONES DE UN MES			ID	RF-004
Prioridad	Media	Procedencia	Cliente X	Eq. Desarrollo
Claridad	Alta	Necesidad	Esencial	Deseable X
Verificabilidad	Alta	Estabilidad	Durante toda la vida útil del proyecto.	
Descripción	Se tiene que poder obtener la información de conexiones dividida por meses para luego poder mostrarla.			

Tabla 9: Requisito RF-004

OBTENER CONEXIONES DE UN DÍA			ID	RF-005
Prioridad	Media	Procedencia	Cliente X	Eq. Desarrollo
Claridad	Alta	Necesidad	Esencial	Deseable X
Verificabilidad	Baja	Estabilidad	Durante toda la vida útil del proyecto.	
Descripción	Se tiene que poder obtener la información de conexiones de un día concreto para luego poder mostrarla.			

Tabla 10: Requisito RF-005

GESTIÓN DE MULTIPLES SERVIDORES			ID	RF-006
Prioridad	Alta	Procedencia	Cliente	Eq. Desarrollo X
Claridad	Alta	Necesidad	Esencial	Deseable X
Verificabilidad	Alta	Estabilidad	Durante toda la vida útil del proyecto.	
Descripción	Hacer que la aplicación pueda gestionar varios servidores en lugar de solamente uno.			

Tabla 11: Requisito RF-006

OBTENER CONEXIONES DE UN MES DE TODOS LOS SERVIDORES MONITORIZADOS			ID	RF-007
Prioridad	Media	Procedencia	Cliente	Eq. Desarrollo X
Claridad	Alta	Necesidad	Esencial	Deseable X
Verificabilidad	Media	Estabilidad	Durante toda la vida útil del proyecto.	
Descripción	Se tiene que poder obtener el total de conexiones de cada servidor divididas por meses para luego poder compararlas.			

Tabla 12: Requisito RF-007

OBTENER CONEXIONES DE UN DÍA DE TODOS LOS SERVIDORES MONITORIZADOS			ID	RF-008
Prioridad	Media	Procedencia	Cliente	Eq. Desarrollo X
Claridad	Alta	Necesidad	Esencial	Deseable X
Verificabilidad	Baja	Estabilidad	Durante toda la vida útil del proyecto.	
Descripción	Se tiene que poder obtener el total de conexiones de cada servidor en un día concreto para luego poder compararlas.			

Tabla 13: Requisito RF-008

CREAR UN NUEVO SERVIDOR			ID	RF-009
Prioridad	Alta	Procedencia	Cliente	Eq. Desarrollo X
Claridad	Alta	Necesidad	Esencial X	Deseable
Verificabilidad	Alta	Estabilidad	Durante toda la vida útil del proyecto.	
Descripción	El sistema tiene que permitir insertar los datos para la creación de un nuevo servidor a monitorizar.			

Tabla 14: Requisito RF-009

EDITAR UN SERVIDOR EXISTENTE			ID	RF-010
Prioridad	Alta	Procedencia	Cliente	Eq. Desarrollo X
Claridad	Alta	Necesidad	Esencial X	Deseable
Verificabilidad	Alta	Estabilidad	Durante toda la vida útil del proyecto.	
Descripción	El sistema tiene que permitir la modificación de los datos almacenados para un servidor de los que se están monitorizando.			

Tabla 15: Requisito RF-010

ELIMINAR UN SERVIDOR EXISTENTE			ID	RF-011
Prioridad	Alta	Procedencia	Cliente	Eq. Desarrollo X
Claridad	Alta	Necesidad	Esencial X	Deseable
Verificabilidad	Alta	Estabilidad	Durante toda la vida útil del proyecto.	
Descripción	El sistema tiene que permitir de los datos almacenados para un servidor de los que se están monitorizando, así como el historial de todas sus conexiones.			

Tabla 16: Requisito RF-011

IDIOMA SEGÚN URL			ID	RF-012
Prioridad	Baja	Procedencia	Cliente	Eq. Desarrollo X
Claridad	Alta	Necesidad	Esencial	Deseable X
Verificabilidad	Alta	Estabilidad	Durante toda la vida útil del proyecto.	
Descripción	El sistema tiene que permitir de los datos almacenados para un servidor de los que se están monitorizando, así como el historial de todas sus conexiones.			

Tabla 17: Requisito RF-012

3.2.3 Requisitos no funcionales

INCLUIR NAVBAR PARA PODER VOLVER SIEMPRE AL HOME DE LA APLICACIÓN			ID	RNF-001
Prioridad	Alta	Procedencia	Cliente	Eq. Desarrollo X
Claridad	Alta	Necesidad	Esencial X	Deseable
Verificabilidad	Alta	Estabilidad	Durante toda la vida útil del proyecto.	
Descripción	Tiene que existir un elemento permanente y común a todas las vistas que permita volver al home desde cualquier parte de la aplicación.			

Tabla 18: Requisito RNF-001

ACTUALIZAR SERVIDOR ACCESIBLE EN EL HOME			ID	RNF-002
Prioridad	Alta	Procedencia	Cliente	Eq. Desarrollo X
Claridad	Alta	Necesidad	Esencial X	Deseable
Verificabilidad	Media	Estabilidad	Durante toda la vida útil del proyecto.	
Descripción	La capacidad de actualizar la información de conexiones del servidor tiene que ser accesible desde el home de la aplicación.			

Tabla 19: Requisito RNF-002

POSIBILIDAD DE CAMBIAR DE IDIOMA DESDE CUALQUIER PARTE DE LA APLICACIÓN			ID	RNF-003
Prioridad	Baja	Procedencia	Cliente	Eq. Desarrollo X
Claridad	Alta	Necesidad	Esencial	Deseable X
Verificabilidad	Media	Estabilidad	Durante toda la vida útil del proyecto.	
Descripción	La posibilidad de cambiar el idioma de la aplicación debe ser accesible desde cualquier parte de la aplicación.			

Tabla 20: Requisito RNF-003

MÚLTIPLES GRÁFICAS PARA MOSTRAR LA INFORMACIÓN DE CONEXIONES			ID	RNF-004
Prioridad	Alta	Procedencia	Cliente	Eq. Desarrollo X
Claridad	Alta	Necesidad	Esencial	Deseable X
Verificabilidad	Alta	Estabilidad	Durante toda la vida útil del proyecto.	
Descripción	Con el objetivo de conseguir una mejor interpretación de las conexiones a un servidor se hace necesario que los mismos datos se muestren con distintas representaciones gráficas.			

Tabla 21: Requisito RNF-004

MOSTRAR DATOS DE CONEXIONES POR MESES			ID	RNF-005
Prioridad	Media	Procedencia	Cliente X	Eq. Desarrollo
Claridad	Alta	Necesidad	Esencial X	Deseable
Verificabilidad	Alta	Estabilidad	Durante toda la vida útil del proyecto.	
Descripción	La aplicación debe disponer de una vista para cada mes del que se quiera saber la información de conexiones.			

Tabla 22: Requisito RNF-005

CALENDARIO DEL MES QUE SE ESTÁ VIENDO			ID	RNF-006
Prioridad	Media	Procedencia	Cliente X	Eq. Desarrollo
Claridad	Alta	Necesidad	Esencial X	Deseable
Verificabilidad	Alta	Estabilidad	Durante toda la vida útil del proyecto.	
Descripción	La aplicación debe mostrar el calendario del mes que se está viendo.			

Tabla 23: Requisito RNF-006

VISUALIZACIÓN DE CONEXIONES PARA UN DÍA CONCRETO			ID	RNF-007
Prioridad	Media	Procedencia	Cliente X	Eq. Desarrollo
Claridad	Alta	Necesidad	Esencial	Deseable X
Verificabilidad	Alta	Estabilidad	Durante toda la vida útil del proyecto.	
Descripción	La aplicación debe mostrar la información de conexiones de un servidor para un día concreto.			

Tabla 24: Requisito RNF-007

VISUALIZACIÓN DE CONEXIONES PARA UN DÍA CONCRETO ACCESIBLE DESDE CALENDARIO			ID	RNF-008
Prioridad	Media	Procedencia	Cliente X	Eq. Desarrollo
Claridad	Alta	Necesidad	Esencial	Deseable X
Verificabilidad	Alta	Estabilidad	Durante toda la vida útil del proyecto.	
Descripción	La aplicación debe mostrar la información de conexiones de un servidor para un día concreto tiene que ser accesible desde ese día en el calendario.			

Tabla 25: Requisito RNF-008

MOSTRAR LISTA SELECCIONABLE CON LOS DISTINTOS SERVIDORES QUE SE MONITORIZAN			ID	RNF-009
Prioridad	Media	Procedencia	Cliente X	Eq. Desarrollo
Claridad	Alta	Necesidad	Esencial	Deseable X
Verificabilidad	Alta	Estabilidad	Durante toda la vida útil del proyecto.	
Descripción	Al permitir múltiples servidores tiene que existir una lista con los disponibles para poder acceder a las visualizaciones descritas anteriormente.			

Tabla 26: Requisito RNF-009

MOSTRAR DATOS COMPARATIVOS DE CONEXIONES POR MESES ENTRE LOS SERVIDORES			ID	RNF-010
Prioridad	Media	Procedencia	Cliente X	Eq. Desarrollo
Claridad	Alta	Necesidad	Esencial X	Deseable
Verificabilidad	Alta	Estabilidad	Durante toda la vida útil del proyecto.	
Descripción	La aplicación debe disponer de una vista para cada mes del que se quiera visualizar la comparativa de servidores monitorizados.			

Tabla 27: Requisito RNF-010

VISUALIZACIÓN COMPARATIVA DE CONEXIONES PARA UN DÍA CONCRETO ENTRE LOS SERVIDORES			ID	RNF-011
Prioridad	Media	Procedencia	Cliente X	Eq. Desarrollo
Claridad	Alta	Necesidad	Esencial	Deseable X
Verificabilidad	Alta	Estabilidad	Durante toda la vida útil del proyecto.	
Descripción	La aplicación debe mostrar la visualización comparativa de servidores monitorizados para un día concreto.			

Tabla 28: Requisito RNF-011

SERVIDOR CON RUBY			ID	RNF-012
Prioridad	Alta	Procedencia	Cliente	Eq. Desarrollo X
Claridad	Alta	Necesidad	Esencial X	Deseable
Verificabilidad	Baja	Estabilidad	Durante toda la vida útil del proyecto.	
Descripción	La aplicación necesita un servidor con la versión 2.1.1 de Ruby para poder hacer el despliegue en él.			

Tabla 29: Requisito RNF-012

BASE DE DATOS POSTGRES			ID	RNF-013
Prioridad	Alta	Procedencia	Cliente	Eq. Desarrollo X
Claridad	Alta	Necesidad	Esencial X	Deseable
Verificabilidad	Baja	Estabilidad	Durante toda la vida útil del proyecto.	
Descripción	La aplicación necesita una base de datos Postgres para poder almacenar la información de conexiones de los servidores.			

Tabla 30: Requisito RNF-013

3.3 Diseño

3.3.1 Herramientas

Para la realización de este proyecto se ha elegido el lenguaje de programación Ruby con el *framework Ruby on Rails*, frente a las alternativas *Phyton* con el *framework Django* y *PHP* con el *framework Symfony*. A continuación se va a exponer una comparativa entre los tres lenguajes y *frameworks* con los que se espera justificar la decisión de una tecnología sobre las otras.

Para la realización de dicha comparación se tomarán como base los documentos siguientes: *PHP* vs *Ruby* vs *Python*: *Three Programing Languages in a Nutshell* (07, s.f.), *PHP* vs. *Python* vs. *Ruby – The web scripting language shootout* (08, 2009) y *Code Wars*: *Ruby* vs *Python* vs *PHP* [Infographic] (09, 2012) en los que se explican detenidamente tanto las ventajas e inconvenientes de cada una de las posibles tecnologías. En lugar de citar cada una de las fuentes se va a proceder a hacer una comparación de todas las tecnologías cogiendo datos y opiniones de cada uno de los documentos pero redactados de una forma completamente distinta para cumplir los objetivos de esta sección de la documentación del proyecto.

Se va a empezar comparando la legibilidad y la usabilidad e intentar decidir cuál de estos lenguajes es el mejor en este apartado. Por un lado *PHP* sigue una aproximación clásica muy similar a las de otros lenguajes como *C*, por lo que para un programador acostumbrado a este tipo de lenguajes es uno que le puede resultar familiar. *Python* por otro lado depende de la sangría y tiene pocas palabras clave lo que es muy recomendable para programadores inexpertos. Por último *Ruby* es el lenguaje más atractivo para programadores experimentados dado que permite programar en pocas líneas de forma elegante y poderosa a la vez que expresivo.

También se va a tener en cuenta la abstracción de la base de datos relacional, en el caso de *PHP*, en un principio estaba pensado y ha utilizado fundamentalmente *MySQL* como base de datos, por lo tanto no ha sido hasta más adelante hasta que se ha creado *PHP Data Objects* (PDO) como capa de abstracción para sistemas de bases de datos basados en *SQL*. En cambio *Python* usando *PEP* y *Ruby on Rails* con *ActiveRecord* tienen incluida la abstracción del *framework* con respecto a la base de datos con lo que es muy sencillo cambiar de una a otra sin tener que cambiar mucho código. En resumen y pese a que los tres lenguajes implementan la abstracción respecto a la base de datos, podría decirse que *PHP* se encuentra un poco por detrás en este campo con respecto a los otros dos.

Pero para mostrar de una forma más visual las diferencias, ventajas y desventajas se adjuntan varias ilustraciones con gráficas y tablas comparativas entre ellas, para mostrar las características descritas con anterioridad y explicar tras esto las conclusiones y la decisión final con respecto al lenguaje de programación a utilizar para la aplicación.

Criterio	Descripción detallada	Mejor	Medio	Peor
Gestión de excepciones	Control de errores y recuperación de ellos	Phyton Ruby	-	-
Disponibilidad	En la mayoría de sistemas existentes.	PHP	Python	Ruby
Legibilidad	Mantenible y resistente cambios de personal.	Python	PHP Ruby	-
Abstracción de la base de datos	Independiente de la BD y mapeo objeto-relacional.	Phyton Ruby	PHP	-
Seguridad	En casos críticos.	Python Ruby	PHP	-
Popularidad	Mercado de trabajo.	PHP	Python	Ruby
Rendimiento	Velocidad y tiempo de ejecución	PHP Ruby Python	-	-
Usabilidad	Creación de prototipos y desarrollo rápidos.	Ruby	Python	PHP
Características funcionales	Dan la posibilidad de utilizar técnicas de programación funcional.	Python Ruby	-	-

Tabla 31: Comparación entre Phyton, PHP y Ruby.

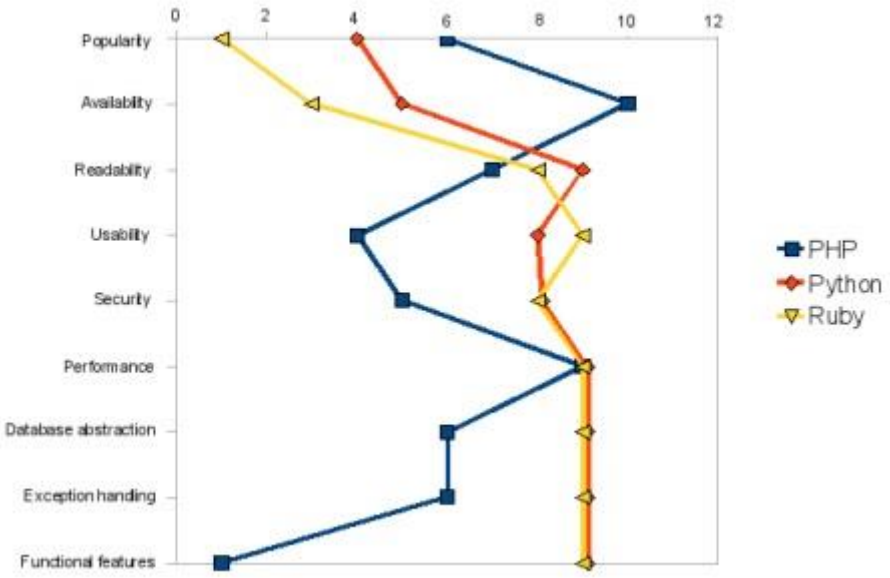

Ilustración 7: Comparación entre Phyton, PHP y Ruby.

Como puede observarse en la tabla y gráfica anteriores, y la tabla comparativa siguiente, la decisión de un lenguaje para el desarrollo de este proyecto estaba entre *Python* y *Ruby*, dado que *PHP* pese a ser el lenguaje más popular, no era el que presentaba unas mejores características para el propósito final del proyecto.

De entre *Python* y *Ruby* al final se ha optado por utilizar *Ruby* como lenguaje de programación para la aplicación. Esta decisión está basada en los argumentos expuestos con anterioridad que como se ha dicho dejaban la elección final entre 2 lenguaje de características

muy similares. Lo peor que tiene *Ruby* frente a sus competidores es que pese a ser muy legible e inteligible porque se basa mucho en el lenguaje natural, con pocas líneas muy descriptivas de lo que hacen, es un lenguaje que tiene una curva de aprendizaje muy pronunciada, es decir, que es muy complicado de aprender a usar, y sobre todo de usar correctamente, pero superado este escollo es el que mejores características tiene.

Dado que el equipo de desarrollo tiene conocimientos previos con *Ruby* y con el *framework Ruby on Rails* el inconveniente de la curva de aprendizaje queda superado por lo tanto la decisión de utilizar este lenguaje para *CONAD* se presenta como la más acertada para este desarrollo.

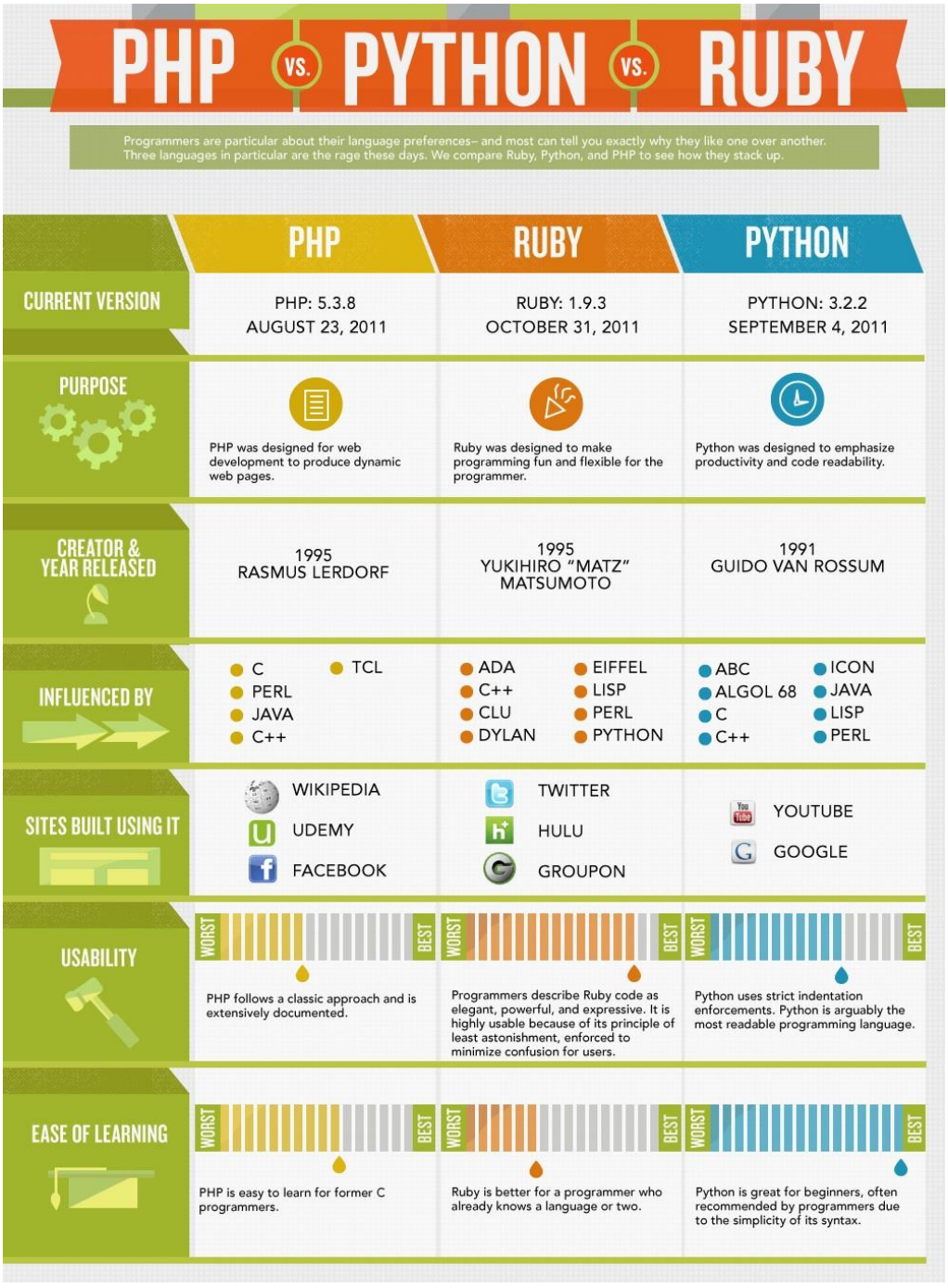

Ilustración 8: Comparación entre Phyton, PHP y Ruby.

Como base de datos en complemento al *framework* de *Ruby on Rails* se ha optado por *PostgreSQL*, dado que como se ha explicado antes está preparado para abstraer bases de datos relacionales, y se ha elegido esta frente a otras opciones como *MySQL* porque en el servidor que se quiere usar para el despliegue (*Heroku*) es la base de datos por defecto y es más cómodo trabajar con ella desde el principio.

3.3.2 Casos de uso

3.3.2.1 Crear un nuevo servidor para monitorizar

Como se puede apreciar en la ilustración 9, para la creación de un nuevo servidor se necesita lo primero iniciar la aplicación, tras eso se debe ir ala desplegable que contiene la lista de servidores y una vez ahí seleccionar la opción de crear un nuevo servidor, esto llevará al usuario al formulario que una vez completado correctamente le permitirá añadir un nuevo servidor al sistema.

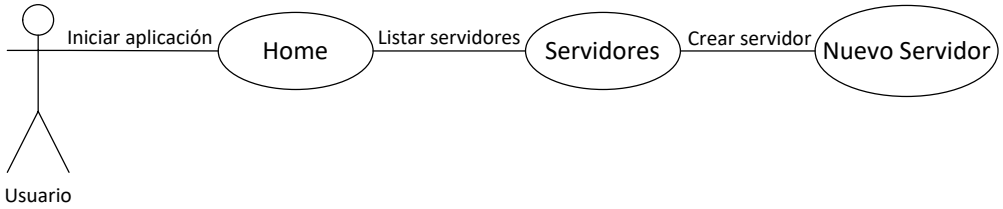

Ilustración 9: Crear un nuevo servidor para monitorizar

3.3.2.2 Modificar un servidor monitorizado

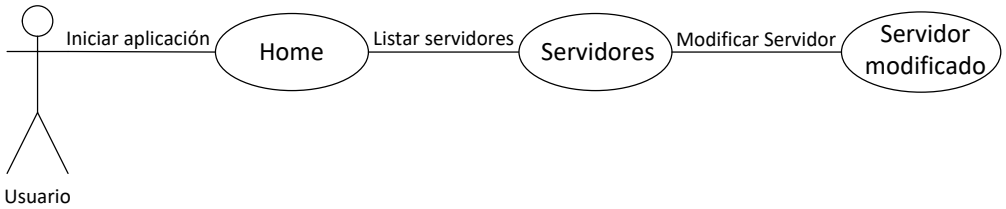

Ilustración 10: Modificar un servidor monitorizado

En la imagen anterior (ilustración 9), se pueden ver los diferentes pasos que debe seguir un usuario para modificar un servidor que ha sido creado previamente, primero se ha de iniciar la aplicación, tras esto se selecciona el servidor que se va actualizar, una vez seleccionado el servidor, habrá que dar al botón de editar que traslada al usuario al formulario de modificación del servidor y una vez completado de forma correcta se realizará la actualización de los datos modificados.

3.3.2.3 Eliminar un servidor monitorizado

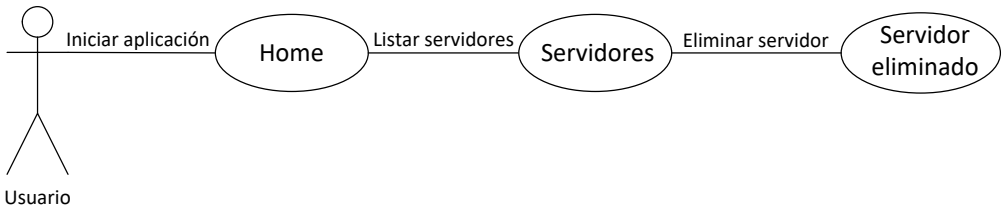

Ilustración 11: Eliminar un servidor monitorizado

3.3.2.4 Actualizar servidores

La ilustración 12 muestra el proceso que ha de seguir un usuario desde que inicia la aplicación hasta que consigue actualizar la información de conexiones de los diferentes servidores que se tienen incluido en la aplicación. Lo primero que ha de realizarse es desde la página de inicio de *CONAD* solicitar la actualización de los servidores, esto produce que se realice una solicitud del comando *last* a cada uno de los servidores que responderán con la salida de dicho comando que se procesará para poder mostrar los nuevos datos obtenidos.

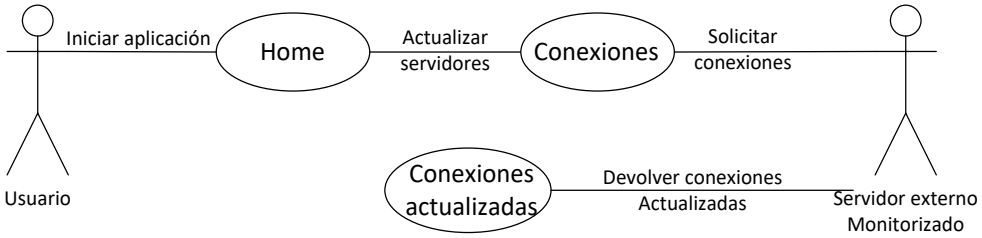

Ilustración 12: Actualizar servidores

3.3.2.5 Visualizar comparativa servidores para un día

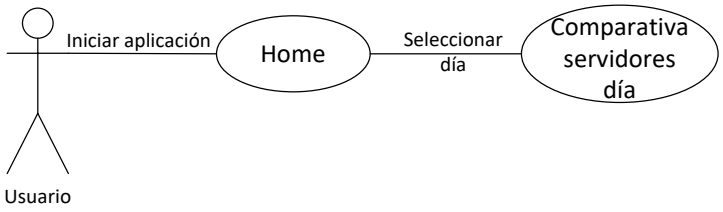

Ilustración 13: Visualizar comparativa servidores para un día

La ilustración 13 muestra los pasos que ha de seguir un usuario para poder visualizar la comparativa de los servidores para un único día en lugar de la visualización de la comparativa del mes que se muestra por defecto, para ello en la página principal (*home*) en el calendario del que se dispone deberá seleccionar el día en el que está interesado y podrá disponer de las diferentes gráficas de la aplicación para únicamente para los datos del día elegido.

3.3.2.6 Visualizar mes de un servidor

Como puede apreciarse en la ilustración 14 para poder visualizar un mes de un servidor concreto en lugar de la comparativa entre todos los registrados hay que desde la página de inicio ir al desplegable con la lista de servidores y una vez ahí selección el servidor que se quiere visualizar y se cargará la vista con la comparativa de uso por usuarios de ese servidor para el mes en el que se encuentra.

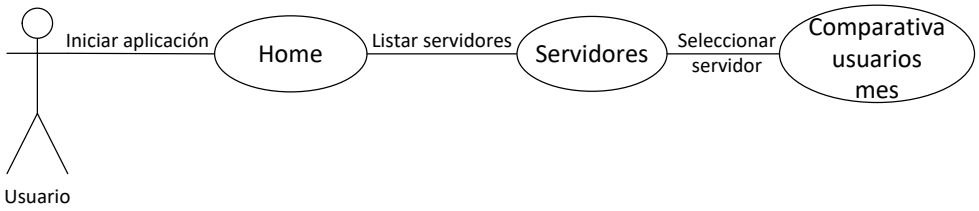

Ilustración 14: Visualizar mes de un servidor

3.3.2.7 Visualizar día de un servidor

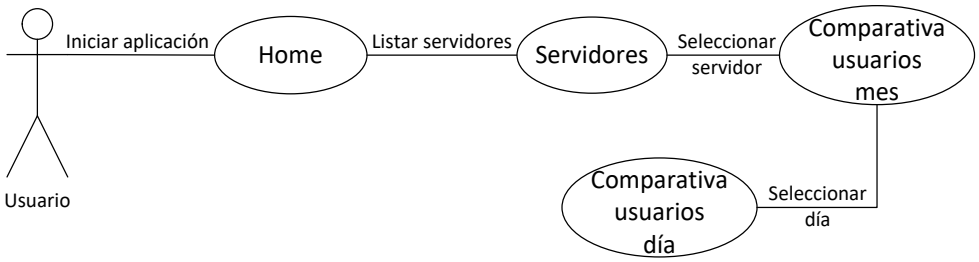

Ilustración 15: Visualizar día de un servidor

En la ilustración 15 se muestra el proceso ha de seguir un usuario para alcanzar la visualización de un día concreto para un servidor y como en el caso anterior, primero desde la página de inicio deberá ir al desplegable de selección de servidores, una vez ahí seleccionar el servidor que va a querer visualizar y por último elegir en el calendario el día en el que quiere ver el detalle de dicho servidor.

3.3.3 Diagrama de navegación

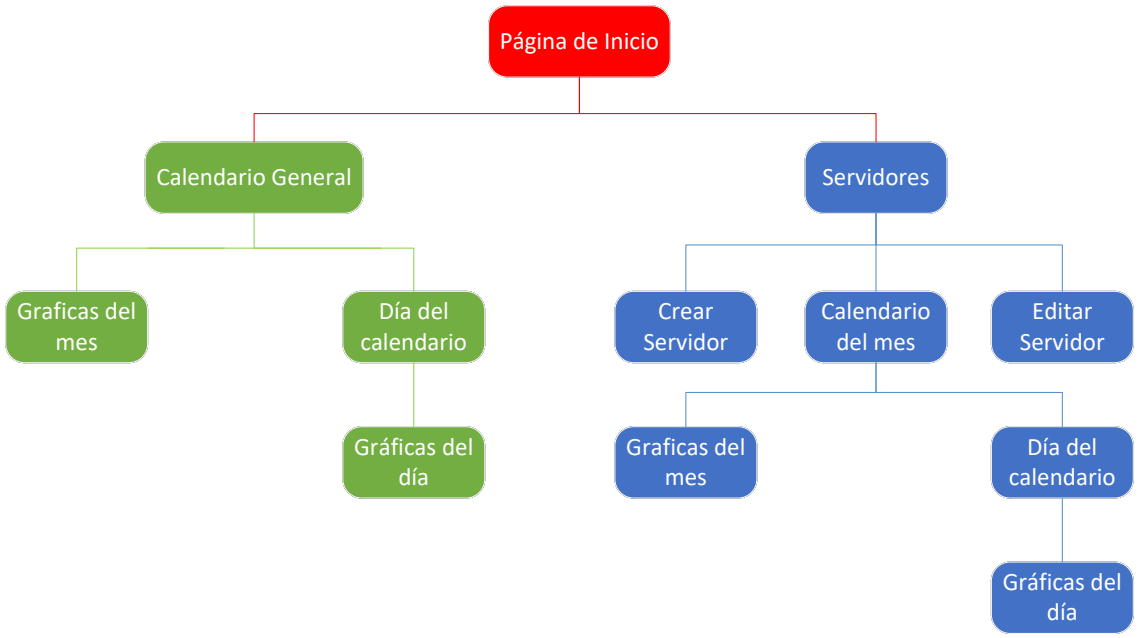

Ilustración 16: Diagrama de navegación

La ilustración 16 muestra el diagrama de navegación de la aplicación web que se ha desarrollado. Primero uno se encuentra en la página de inicio, representado con una caja roja en el diagrama. Desde ahí se tienen 2 opciones de acción, se puede optar por gestionar los servidores o por la visualización de datos comparativos de servidores a través del calendario, si se opta por esta segunda opción se presentan 2 líneas de acción, puede cambiarse el tipo de gráfica que se quiere visualizar o seleccionar un día del calendario, en caso de seleccionar un día se podrá para los datos de ese día seleccionar también el tipo de gráfica en el que se verán los datos.

Si por otro lado se toma la decisión de ir al desplegable de servidores y seleccionar uno se abre otro abanico de posibilidades, se puede editar o crear un nuevo servidor, además de las opciones expuestas con anterioridad pero en lugar de ser comparativas de servidores para una mes/día serán gráficas que muestren la comparación de uso por usuarios para el servidor que se eligió en el desplegable.

En la ilustración 16 se muestra en color rojo la página inicial y las acciones que se pueden tomar desde ella, luego en azul todas las acciones que se pueden realizar relacionadas con el los servidores y en verde todas las acciones que se pueden realizar que tengan como objeto de análisis la comparación entre los diferentes servidores registrados para un periodo de tiempo determinado.

3.3.4 Esquema de la base de datos

En este caso el esquema de la base de datos es muy sencillo debido a que solo hay 2 entidades, los servidores, que es entorno a los cuales gira todo el diseño de la aplicación y por otro lado las conexiones *ssh* que se realizan a dichos servidores y que son con las que luego se dibujan las diferentes gráficas que permiten una interpretación más sencilla de los datos de conexiones en los servidores frente a otras formas de exponer dichos datos.

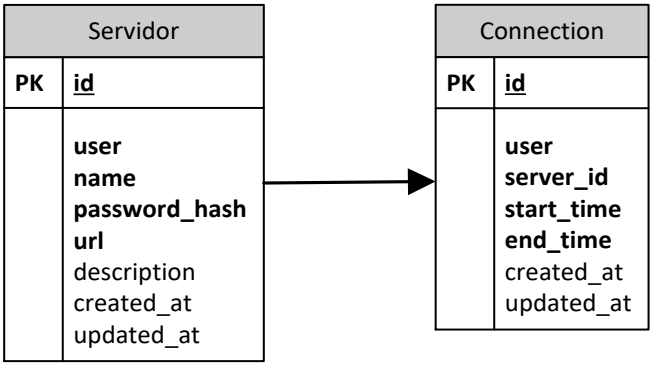

Ilustración 17: Esquema de la BD

Como puede verse en la ilustración 18, el diagrama entidad – relación, de los servidores se almacena un usuario *ssh* que tenga permisos para poder ejecutar el comando *last* y su contraseña, un servidor recibe conexiones *ssh* que quedan registradas en él y que luego se extraen de ese comando, de cada conexión se almacena el usuario que la realiza así como la fecha de inicio y fin de la conexión, con esta información es con la que luego trabaja la aplicación y la que se muestra dentro de ella.

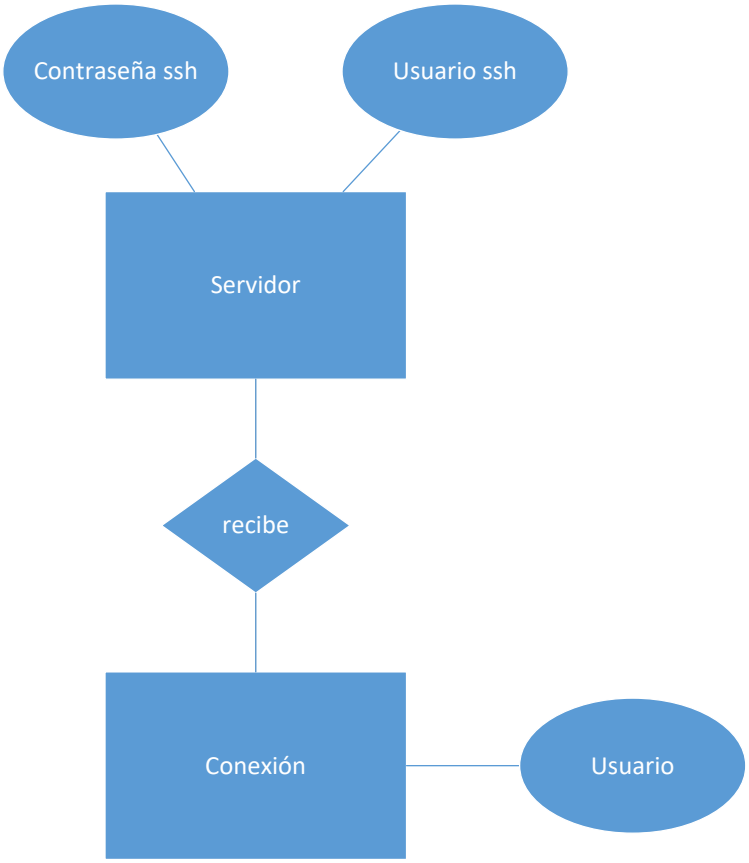

Ilustración 18: Entidad - relación

3.3.5 Implementación

Durante la implementación de este proyecto ha habido que enfrentarse con un gran número de dificultades, la primera y principal, el desarrollo de una aplicación en *Ruby on Rails* dado que es una tecnología no muy extendida y de difícil aprendizaje, por suerte no es del todo desconocida para el equipo de desarrollo, también cuenta con una comunidad muy activa y un gran número de tutoriales y dudas resueltas. Además es un lenguaje que cuenta con un gran número de librerías (*gemas*) que permiten automatizar muchas de las tareas que de otro modo habría que desarrollar a mano.

Dicho esto, la primera dificultad derivada de la implementación propiamente dicha fue la de manejar las conexiones *ssh* a los servidores remotos de modo que pudiese ejecutarse en ellos el comando *last* y obtener el resultado para su posterior procesado en la aplicación. Puede parecer trivial, pero además del propio hecho de gestionarlas y obtener la salida de la conexión para procesarla en la aplicación se planteó otra gran dificultad, como almacenar las contraseñas cifradas en la base de datos pero a la vez obtenerlas abiertas para poder conectar.

No debería darse el caso de que nadie externo a la aplicación tuviese acceso a la base de datos, pero en caso de tener lugar dicho acceso, y para evitar que se comprometan las contraseñas se ha procedido a cifrarlas, y este proceso de cifrado ha traído muchos quebraderos de cabeza, dado que se quiere guardar la contraseña cifrada, pero la conexión *ssh* como es lógico necesita la contraseña abierta. Para poder resolver todos estos problemas al final se ha recurrido a la creación de una clave aleatoria privada de cifrado con la que se ofusca la contraseña. Dado que se quiere evitar que la contraseña pueda obtenerse abierta en

cualquier sitio que no se sea en las conexiones *ssh*, el método que devuelve la contraseñas es privado del modelo *server*, de modo que es el único que tiene acceso a la clave abierta, no lo tendrían siquiera las clases que heredasen de él. Por esta razón la conexión *ssh* se realiza en el server, porque es el único lugar donde se puede usar la contraseña abierta.

Otro de los problemas a los que ha habido que enfrentarse ha sido la realización de las gráficas, para ello se decidió el uso de la librería de *javascript d3js* que permite la realización de gráficas a partir de datos, el primer reto fue el de aprender a realizar gráficas utilizando dicha librería, al final los tipos de gráfica incluidos en *CONAD* son: de líneas, de barras, de barras apiladas y de donut. Además otro de los problemas era nutrir de datos las gráficas para poder dibujarlas, se ha optado por meter los datos dentro de un atributo de la *svg* que contiene a la gráfica, otra opción podría haber sido obtenerlos de un *webservice* a través de una llamada *AJAX*.

También hubo que enfrentarse a permitir que la aplicación sea multilenguaje, además que dicho lenguaje se mantenga durante la navegación. Para la gestión de la elección del lenguaje, se ha optado por un selector en la *navbar* que está visible para cualquier vista y que permite cambiar de idioma en cualquier momento, y además el idioma viene dado por el prefijo de idioma con el que se carga la vista, por ejemplo para visitar la página en español la dirección sería http://domainname.com/es. Después de elegir idioma con el selector siempre se vuelve a la home.

3.3.6 Implantación

Para el despliegue de la aplicación (configuración e instalación) se utiliza una gema de *Ruby* llamada *Capistrano*, dicha gema está ya configurada para realizar el despliegue en cualquier máquina pero necesita de unos cuantos ajustes para poder funcionar. Pero antes de eso lo primero que hay que hacer es configurar el servidor.

Todos los pasos necesarios para la configuración, despliegue e instalación de la aplicación vienen detallados en los diferentes apéndices.

4.- Evaluación

Para la evaluación del sistema se van a tener en cuenta fundamentalmente dos aspectos que se consideran esenciales para valorar que la aplicación sea útil e interesante para su uso e implantación como herramienta de monitorización de conexiones *ssh*. El primer aspecto es la satisfacción de los usuarios finales con respecto al sistema, y el segundo es el rendimiento, y comprobar si se podría haber optimizado el sistema de algún modo.

Para las pruebas de evaluación del sistema se va a buscar una muestra mínima significativa que nos permita que los resultados puedan ser extrapolables al máximo de usuarios finales posible. Para ello se ha hecho una preselección de personas que se piensa que van a tener un nivel destreza con aplicaciones web, alto, medio y bajo. En el bajo entran personas que normalmente no utilizan aplicaciones web en su día a día y que utilizan un navegador para consultas de información en internet o para ver su correo electrónico de vez en cuando.

Para el nivel medio se ha buscado a gente que trabaja normalmente con aplicaciones específicas aunque no sean aplicaciones web, o que utilizan internet y aplicaciones web con cierta frecuencia, aplicaciones como por ejemplo Facebook, Twitter o alguna otra red social, leen medios de comunicación digitales, o realizan compras a través de tiendas en internet.

Por último como representantes de las personas con un nivel alto se ha seleccionado a gente que no sólo está acostumbrada al uso de diferentes aplicaciones web, sino que también esté en constante contacto con ellas, gente que trabaje no sólo con aplicaciones propietarias propias de su trabajo, si no que trabajen con varias diferentes a la vez, o que se dediquen al mundo informático y del desarrollo software aunque dicho desarrollo no esté relacionado directamente con las aplicaciones web en sí mismas.

Por tanto se seleccionaron 10 personas que podrían ser incluidas en cada uno de estos grupos, o que al menos en un primer momento se pensó que pertenecían a dichos grupos. En cualquier caso, con el objetivo de poder verificar la veracidad exactitud de esta hipótesis se realizó un formulario con 10 preguntas y se decidió que los usuarios capaces de responder correctamente a entre 0 – 4 preguntas pertenecerían al grupo bajo, con 5 – 8 respuestas acertadas serían del grupo medio y el resto serían usuarios de nivel alto. De ello se derivaron los siguientes resultados. Como puede apreciarse en la figura de la ilustración 19 antes de que los usuarios respondiesen al formulario se contaba con 10 personas agrupadas en cada nivel de conocimientos.

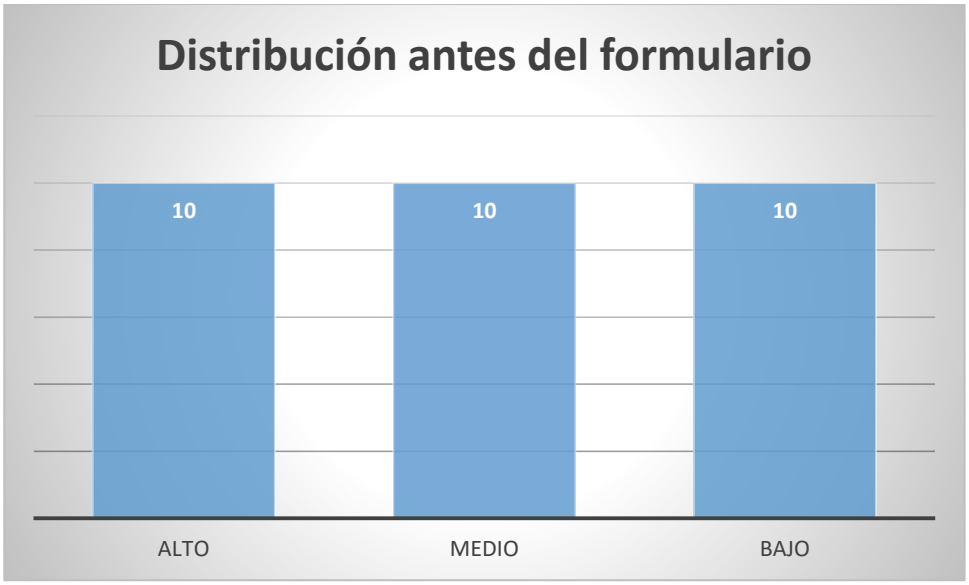

Ilustración 19: Distribución de usuarios antes del formulario

Una vez realizado el formulario y de ver los resultados que están expuestos en la gráfica de la ilustración 20 puede observarse que al final nuestra muestra se ha polarizado mucho, aumentando el número de usuarios de nivel bajo significativamente en incluso aumentando el número de usuarios expertos, pero por el contrario se ha reducido el número de usuarios medios, esto se ha producido probablemente porque el contenido de las preguntas del formulario era de una complicación superior a la deseada en un primer momento.

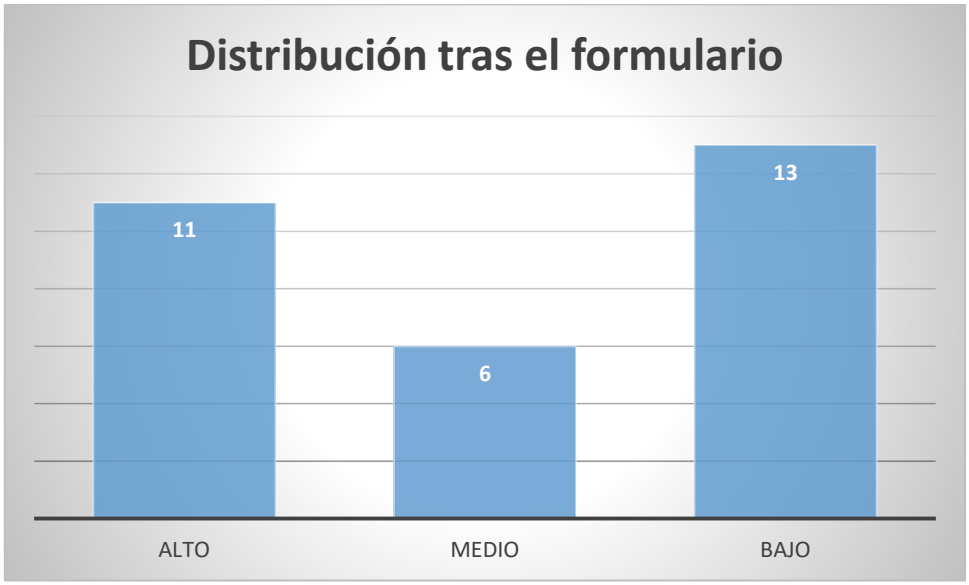

Ilustración 20: Distribución de usuarios tras el formulario

4.1 Satisfacción de usuarios

Una vez establecidos los grupos de prueba lo primero que se decidió hacer para comprobar la satisfacción de los diferentes usuarios fue sin ningún tipo de instrucción previa o preparación ponerles ante la aplicación a ver que les parecía, si eran capaces de hacer algo con ella o si podían imaginarse (aunque fuese de forma aproximada) para qué sirve.

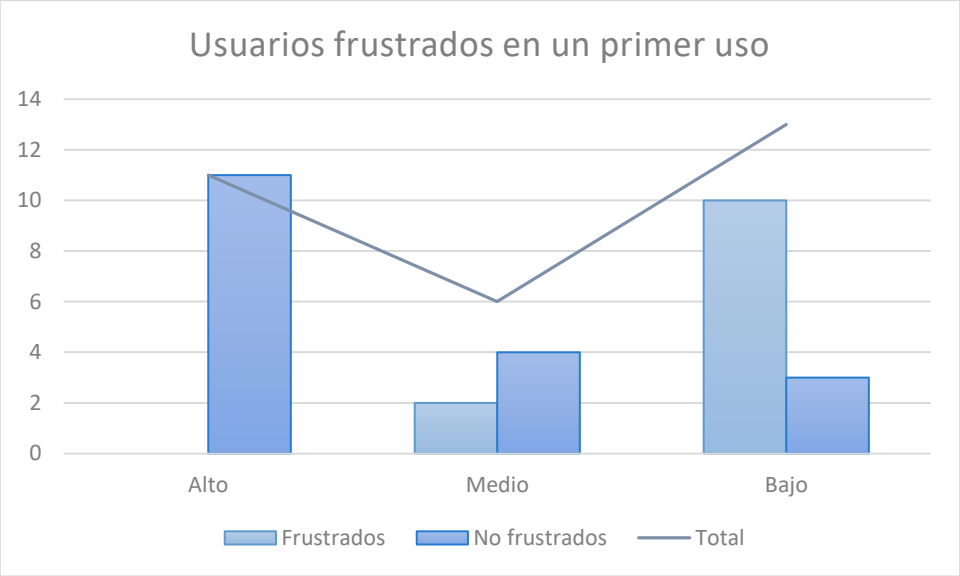

Ilustración 21: Gráfica frustración primer uso

En la gráfica que se muestra en la ilustración 21 puede observarse que la mayoría de usuarios no expertos o de nivel medio se sienten frustrados con la aplicación si se les presenta sin ningún tipo de explicación previa o justificación de lo que están haciendo. Se preguntó la opinión con respecto a la aplicación después de su uso a todos los usuarios, y las respuestas por parte de las personas de nivel medio/bajo fueron que no entendían nada, que qué era eso de los servidores, y que si "que las gráficas muy bonitas" pero que no entendían que representaban o para que servía el botón de actualización. En el caso de los usuarios de nivel alto se han sentido cómodos con ella, aunque también tenían ciertas dudas, como qué tipo de conexiones se recogían en la aplicación o qué se mostraba cuando se seleccionaba un servidor concreto.

Si bien es cierto que la aplicación está dirigida a gente que podría encuadrarse en el perfil alto de nuestra selección de usuarios, es cierto que nos hubiese gustado que la aplicación fuese sencilla de primeras a cualquier usuario independientemente de su nivel de dominio informático, pero dado que todos los miembros de nuestro grupo objetivo se han sentido cómodos con la aplicación podemos considerar la primera prueba un éxito.

Tras esta prueba se ha realizado otra similar de uso de la aplicación pero en este caso a los usuarios se les dieron ciertas nociones básicas sobre la aplicación, cuáles eran sus objetivos, el funcionamiento básico y que representaba cada uno de los componentes de los que consta la aplicación.

En este caso los resultados fueron mucho más positivos que en el primer caso de estudio que se tuvo en cuenta, como puede apreciarse en la ilustración 22 la tasa de

frustración es mucho menor que en el caso anterior, además se ha visto reducida solamente a una pequeña parte de los usuarios de nivel bajo, lo cual sí que representa un éxito, puesto que se puede decir que hemos completado una de las metas de este proyecto que era el conseguir que la aplicación fuese sencilla y usable.

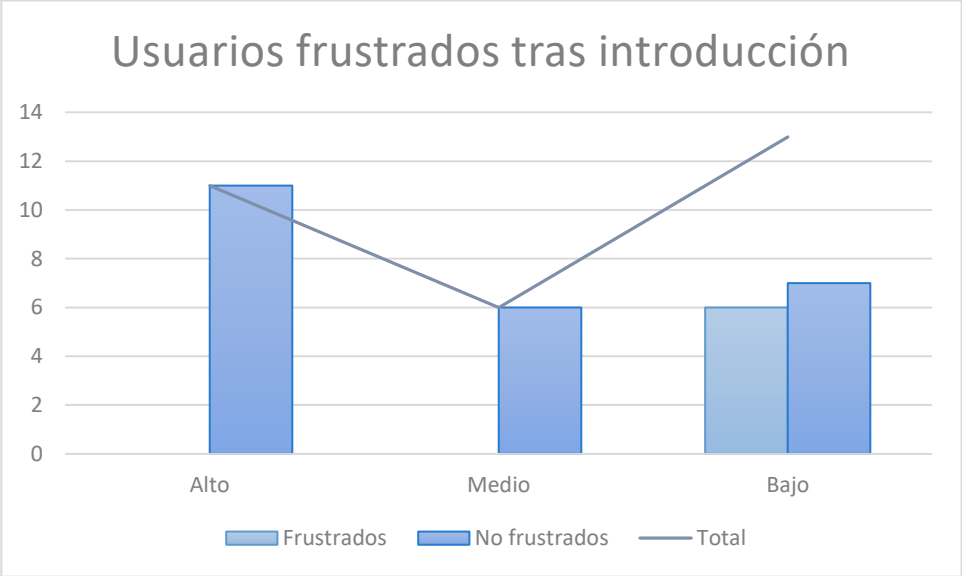

Ilustración 22: Usuarios frustrados tras introducción

El poder decir que nuestra aplicación es sencilla y usable se debe a que la introducción que se les proporcionó a los usuarios es similar a la que puedan tener o adquirir cualquier futuro usuario de la aplicación dado que los usuarios finales serán personas que busquen una aplicación para monitorizar conexiones *ssh* en los diferentes servidores que tengan a su cargo. Por tanto, dudas como qué representan los diferentes gráficos o qué tipo de conexiones son las que se registran en la aplicación no tendrán sentido.

Para la siguiente prueba se seleccionó sólo a los usuarios expertos, y cada uno de ellos se les ofreció la posibilidad de poder introducir un servidor *ssh* al que tuviesen acceso y en el que tuviesen los permisos para poder ejecutar el comando *last*, varios de ellos que sí que disponían de dicho tipo de información los introdujeron y pudieron ver la que la aplicación recogía la información de conexiones de dicho servidores y lo cierto es que se sorprendieron mucho. Tras hacer pruebas y visualizar los datos todos ellos afirmaron que la herramienta es útil y que no pensaban que se pudiese obtener toda esa información sobre el servidor solo con las credenciales de conexión *ssh*.

También hay que reconocer que muchos de los usuarios expertos hicieron muchas preguntas sobre cómo estaba implementada la seguridad de las conexiones, y que como se almacenaban las contraseñas, tras explicarles el proceso y que las contraseñas nunca estaban abiertas y legibles dentro de la aplicación, y que eran guardadas cifradas dentro de la base de datos se quedaron mucho más tranquilos.

La conclusión de estas pruebas fue que todos los usuarios expertos recomendaban la aplicación, que les había sorprendido gratamente y que no solo la utilizarían en caso de necesitar monitorizar las conexiones *ssh* sino que además la recomendarían a otras personas.

4.2 Rendimiento de la aplicación

Respecto al rendimiento de la aplicación se ha intentado optimizar lo máximo posible para que se pueda utilizar en cualquier servidor aunque no se potente. Se pretende que sea una herramienta útil que suponga una mejora y un beneficio para el usuario final, por esto se ha hecho especial énfasis en el rendimiento, puesto que una máquina potente es cara, la intención es que el sistema pueda ser desplegado en una máquina antigua, dado que no requiere que soporte muchas conexiones (en teoría es una herramienta sólo para el administrador de sistemas) y como se dice está optimizado para consumir el mínimo de recursos posibles de la máquina.

Para esto lo primero que se ha hecho es respecto a la parte de *frontend* (la parte visual) en lugar de realizar una llamada *AJAX* con *Javascript* para obtener los datos de forma dinámica cada vez que se pintase una gráfica o se cambiase el tipo de gráfica, lo que se ha hecho es:

Primero unificar los datos que necesita cada una de las gráficas que se van a pintar y que sean interpretables por la librería *3DJS* (la librería de *Javascript* que dibuja las distintas gráficas) y que los mismo datos sirvan tanto para dibujar una gráfica de barras que un donut.

Lo siguiente que se ha hecho es que los datos en lugar de ser requeridos cada vez que se dibuje la gráfica, es que se carguen en la vista con *Ruby* de modo que *Javascript* sólo tenga que leerlos sin hacer peticiones al servidor y sin ralentizar las aplicación.

Ilustración 23: Tiempo de carga gráficas

Como se puede observar en el gráfico de la ilustración 23 y como se ha descrito con anterioridad todo el tiempo que se tarda en visualizar la página se invierte en el cálculo de los valor que se van a representar, una vez que quedan incluidos en la vista ya no se tienen que volver a cargar y por tanto el cambio de una gráfica a otra es instantáneo, pudiendo de este modo cambiar entre las diferentes gráficas sin tener penalización ya sobre el tiempo.

Por último la mejora de rendimiento más reciente es la de dividir la actualización de servidores en hilos de modo que dicha actualización en lugar de llevarse a cabo en un *webservice* cuando se hace la petición, se ejecuta en *background*, y en vez de ejecutarse la actualización de cada servidor de forma secuencial lo que se hace es que se divide en un hilo para cada uno de los servidores de modo que la información de cada uno se obtiene de manera independiente a la de los otros. Con esto se consiguen 2 mejoras en el rendimiento, por un lado que no se bloquea la aplicación mientras se realiza la actualización, y por otro que al no realizarse la actualización de forma secuencial si no que en paralelo, el tiempo que se tarda en realizar es menor.

En la siguiente gráfica se muestran los tiempos de procesado de 30 días antes de después de la inclusión de *sidekiq* para poder procesar en paralelo los *logs* de varios servidores. En el eje vertical se muestran los segundos que se tarda en procesar los 30 días y en el eje horizontal el número de servidores que se tienen registrados en la aplicación. Antes de la inclusión de esta herramienta el procesado de servidores se hacía de forma secuencial con el consiguiente coste en tiempo que se aprecia perfectamente en la ilustración 24. Esto nos demuestra que era una mejora necesaria y que realmente ha tenido un impacto positivo en el rendimiento de la aplicación.

Ilustración 24: Tiempo de procesado 30 días

5.- Planificación y presupuesto.

5.1 Planificación.

En este apartado se va a describir la planificación realizada para este proyecto y se verá reflejada en el diagrama de Gantt que se muestra a continuación, luego se irán describiendo las distintas actividades de las que se compone el proyecto.

Id	Modo de tarea	Nombre de tarea	Duración	Comienzo	Fin	Predecesoras
1		Creación de la aplicac	5 días	lun 12/01/15	vie 16/01/15	
2		Definición de los modelos	5 días	lun 19/01/15	vie 23/01/15	1
3		Conexión ssh y obtención de logs	5 días	lun 26/01/15	vie 30/01/15	1
4		Gestión de contraseñas ssh	5 días	lun 02/02/15	vie 06/02/15	1;2
5		Creación de las vistas y formularios	5 días	lun 09/02/15	vie 13/02/15	1
6		Función que obtenga los datos necesarios para la representación	5 días	lun 16/02/15	vie 20/02/15	3;5
7		Hacer que la información se muestre por mes natural y no los	5 días	lun 23/02/15	vie 27/02/15	6
8		Inclusión del calendario a las vistas	5 días	lun 02/03/15	vie 06/03/15	5
9		Inclusión de un gráfico de lineas	5 días	lun 09/03/15	vie 13/03/15	5
10		Inclusión de un gráfico de barras	5 días	lun 16/03/15	vie 20/03/15	5

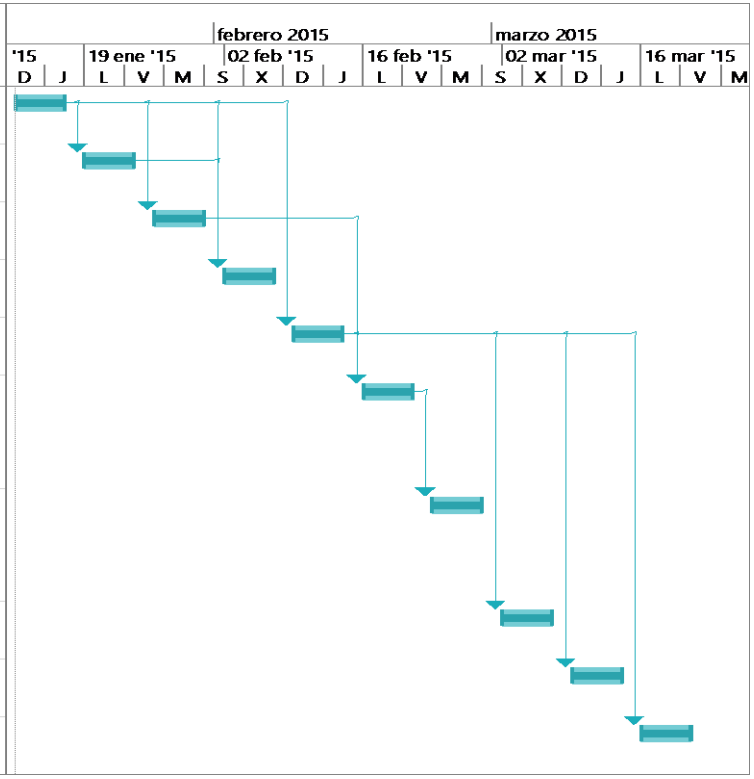

Ilustración 25: Diagrama de Gantt, tareas 1-10

5.1.1 Creación de la aplicación Rails

La primera tarea a llevar a cabo en el proyecto es la creación de la aplicación Rails genérica a partir de la cual se desarrollará el proyecto. Para esta tarea se ha programado una semana porque incluye la generación de un entorno de pruebas así como de un sistema de integración continua que permitirán que el desarrollo ágil sea posible.

5.1.2 Definición de los modelos

Tras la creación de la aplicación dado que Rails nos proporciona un modelo Modelo-Vista-Controlador (MVC) será necesaria la creación de los modelos que van a ser utilizados para nuestra aplicación, desde los servidores a las conexiones, deberán ser definidas como modelos de datos con sus atributos.

5.1.3 Conexión ssh y obtención de logs

Tras la creación de la base de la aplicación va a ser necesaria la creación de un mecanismo que conecte CONAD con el resto de servidores de lo que va a obtener la información a través del comando 'last' y su posterior tratamiento a un formato inteligible para la apliación.

5.1.4 Gestión de contraseñas ssh

Dado que la conexión ssh se tiene que realizar con la contraseña sin cifrar hay que encontrar una forma de almacenar las contraseñas cifradas pero que a la vez puedan ser utilizadas por el servidor para conectarse a otras máquinas y obtener de ellas los logs a través del comando 'last'. Para ello se utilizará el cifrado AES, y se almacenará una clave privada en el servidor que se generará en caso de no haber ninguna definida, con la que se cifrarán las contraseñas en la base de datos y con la que se descifrarán cuando se necesite establecer la conexión con el servidor.

5.1.5 Creación de las vistas y formularios necesarios

Ahora será necesario crear las vistas básicas para nuestra aplicación, una vista general en la que se mostrarán más adelante el calendario y las gráficas, así como el resto de vistas/formularios que nos permitirán interactuar con la aplicación e incluirle información como el usuario y contraseña del usuario ssh necesario para un servidor.

5.1.6 Función que obtenga los datos necesarios para la representación gráfica

Creación de la una función que analice los datos obtenidos de conexiones de los distintos servidores de modo que luego sean representables a través de una gráfica (o varias) que permitan una interpretación más visual de los datos que se han ido recogiendo en la aplicación.

5.1.7 Hacer que la información se muestre por mes natural y no los últimos 30 días

Lo sencillo en principio es mostrar la información de los últimos 30 días desde la fecha actual pero se ha decidido que sería más interesante y dado que se tiene pensada la inclusión de un calendario mostrar los datos por meses naturales que se correspondan con la información que posteriormente mostrará el calendario.

5.1.8 Inclusión del calendario a las vistas

Para facilitar la visualización de los datos en la aplicación se ha tomado la determinación como se comentó en el punto anterior de utilizar meses naturales y para la

selección tanto de meses como de días se ha pensado que la forma más natural es a través de un calendario.

5.1.9 Inclusión de un gráfico de líneas

La primera gráfica a incluir es una gráfica de líneas que muestra la evolución de las conexiones en el tiempo, para un mes natural o por horas para un día según lo que se seleccione.

5.1.10 Inclusión de un gráfico de barras

También se quiere crear un gráfico de barras que muestre una comparativa para el mes de los distintos servidores que se están monitorizando para comprobar de esta manera su uso y carga.

Id		Modo de tarea	Nombre de tarea	Duración	Comienzo	Fin	Predecesoras
11	✓		Inclusión de un gráfico de barras	5 días	lun 23/03/15	vie 27/03/15	5
12	✓		Inclusión de un gráfico de donut	5 días	lun 30/03/15	vie 03/04/15	5
13	✓		Transiciones entre los gráficos	5 días	lun 06/04/15	vie 10/04/15	5;8;9;10;11;12
14	✓		Visualización de datos por servidor	5 días	lun 13/04/15	vie 17/04/15	3;5;6;7;8;9;10;11;
15	✓		Formularios de creación y edición de	5 días	lun 20/04/15	vie 24/04/15	3;5;6;7;8;9;10;11;
16	✓		Selector de servidores	2 días	lun 27/04/15	mar 28/04/15	14;15
17	✓		Botones para seleccionar los distintos gráficos	1 día	mié 29/04/15	mié 29/04/15	13
18	✓		Inclusión de paths para idiomas	2 días	jue 30/04/15	vie 01/05/15	
19	✓		Traducciones de la página	5 días	lun 04/05/15	vie 08/05/15	18

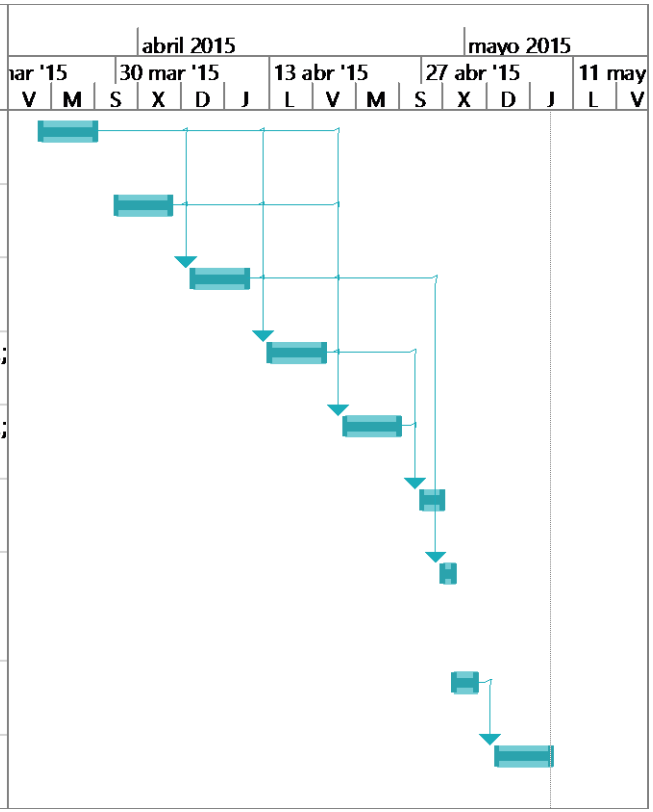

Ilustración 26: Diagrama de Gantt, tareas 11-19

5.1.11 Inclusión de un gráfico de barras apiladas

Este gráfico muestra los totales de conexiones frente al tiempo de manera parecida al gráfico de líneas pero en lugar de superponer unas líneas a las otras en este caso se apilan las barras que representan a cada servidor de modo que de forma sencilla se intuye la proporción de conexiones de cada servidor con respecto al total de las que se han producido en un día.

5.1.12 Inclusión de un gráfico de donut

Este gráfico muestra la parte proporcional del total de conexiones que se han realizado durante el mes para cada uno de los servidores de manera que pueda apreciarse que servidores se utilizan más o menos durante el mes, es fundamentalmente una gráfica comparativa entre servidores.

5.1.13 Transiciones entre los gráficos

Preparar la transición para poder cambiar entre los gráficos descritos anteriormente, permitir el paso de unos a otros de forma dinámica utilizando javascript para evitar la recarga de la página cada vez que se quiera visualizar una representación distinta de entre las ofrecidas por la aplicación.

5.1.14 Visualización de datos por servidor

Permitir la visualización de datos por servidor, adaptar las gráficas y las vistas de forma que al igual que se muestran ahora por servidores, se puedan visualizar por los usuarios del servidor seleccionado.

5.1.15 Formularios de creación y edición de servidores

Adaptar las vistas de los formularios de servidores para poder seleccionarlos de modo se pueda elegir el que se quiera de los existentes y una vez ahí poder editarlo o que si se quiere poder acceder al formulario de creación de uno nuevo.

5.1.16 Selector de servidores

Como se ha dicho en el apartado anterior para la selección de los servidores o la creación de uno nuevo se va a crear un desplegable con los existentes y un enlace para la inclusión de nuevos.

5.1.17 Botones para seleccionar los distintos gráficos

Vincular las transiciones entre los distintos gráficos a las acciones de botones en lugar de que se produzcan cada vez que se cumpla un tiempo determinado y en un orden concreto. Desvincularlos de temporizadores o estados anteriores.

5.1.18 Inclusión de paths para idiomas

Crear nuevos ‘paths’ que permitan el cambio entre los distintos idiomas con los que cuenta la aplicación.

5.1.19 Traducciones de la página

Traducir la página del inglés que es el idioma en el que está desarrollada al español para poder utilizarse de forma bilingüe.

5.2 Presupuesto.

De acuerdo a la planificación expuesta con anterioridad, la duración estimada de realización del proyecto software es de 6 meses, por lo tanto hay que considerar los gastos que va a haber que cubrir durante dicho periodo.

5.2.1 Gasto de personal

Para este proyecto será necesaria la contratación de un jefe de proyecto y de un desarrollador Ruby on Rails junior. El jefe de proyecto deberá ser la persona encargada de gestionar y dirigir el proyecto para que se realice de manera adecuada y de acuerdo a la planificación. A su vez el desarrollador se encargará de programar la aplicación siguiendo las especificaciones del jefe de proyecto.

El cálculo exacto del coste del desarrollador y también del jefe de proyecto se basa en la información obtenida de la página del BBVA para el cálculo de costes de personal para pymes (10, s.f.)

Como se extrae de la referencia el coste para la empresa de un empleado además del salario bruto es de entre el 31% al 35% de esa misma cantidad en concepto de Seguridad Social, además de esto también hay que tener en cuenta otras cuestiones tales como despidos, vacaciones o bajas, por lo que vamos a decir que el coste añadido para la empresa sobre el salario bruto sería aproximadamente de un 40%.

5.2.1.1 Desarrollador

Dado que para este proyecto no se va a realizar ninguna tarea que requiera de un desarrollador senior, la idea sería contratar a un junior, dado que el desarrollo va a estar dirigido y supervisado por el jefe de proyecto no es necesario un perfil con más experiencia.

De acuerdo a mi experiencia y al salario que percibía como junior en la empresa MO2O de 18.000€ brutos al año, por lo tanto es el salario que se considera apropiado para este perfil y para los conocimientos necesarios para el desarrollo de esta aplicación. Y dado que la duración estimada para este proyecto es de 6 meses el coste del salario bruto del desarrollador será de 9.000€ para este periodo, con lo que aplicándole el coste adicional aproximado del 40% para la empresa por persona, el coste de personal final por este empleado será de 12,600€.

5.2.1.2 Jefe de proyecto

Del mismo modo que se ha calculado el coste para el desarrollador se obtiene el coste para el jefe de proyecto, al igual que se obtuvo el coste de un desarrollador a partir del salario en MO2O el coste de un jefe de proyecto sería de unos 35.000€ brutos al año, y como en el caso anterior al ser un periodo de 6 meses el coste quedaría en 17.500€ más el 40% de costes añadidos para la compañía, siendo de este modo de 24.600€ para la duración del proyecto que se va a desarrollar.

5.2.2 Gasto de material

Para que ambos perfiles descritos anteriormente puedan desempeñar sus funciones deberán contar con el material necesario, en este caso será de un ordenador portátil para cada uno de ellos, así como una pantalla auxiliar, un teclado y un ratón.

De acuerdo a pccomponentes (11, s.f.), el coste de un ratón y un teclado es de 13,95€, a esto habría que añadirle un ordenador portátil de un coste aproximado de unos 1000€ que se le dejará elegir a cada uno mientras no supere este coste, así como una pantalla (12, s.f.) de 24" que saldría por 146€. En total el coste por equipo sería de 1.000€ del portátil, 146€ de la pantalla y 13,95€ del ratón y el teclado, es decir, de 1.159,95€. A todo esto habría que añadir el coste de un cuaderno, unos bolígrafos, un portaminas y un borrador, con un valor de unos 5€.

5.2.3 Costes indirectos

Parte de los costes indirectos serían el alquiler de unas oficinas, suponiendo que en este caso se cuenta con unas oficinas de 200m^2 en Las Tablas, con un coste de 1.730€ al mes. El coste de amueblar la oficina para que cuente además de los puestos de los que disponga de los 2 nuevos necesarios para los integrantes de este proyecto el coste es de unos 320€.

Ilustración 27: Oferta puesto oficina

Además de esto hay que incluir el coste proporcional de agua y electricidad que consume cada uno de los empleados, así como el gasto de café, leche y azúcar que se dispensará a coste de la empresa a los empleados, aproximadamente 2 cafés al día y un litro de leche a la semana, lo que hacen unos 4€ en café de 0,40€ por cápsula de *Nespresso*, que junto con los 0,6€ de la leche suman un gasto de 4,6€ por persona a la semana.

También hay que añadir a todo esto los gastos de agua, electricidad y teléfono de la oficina. En este caso se estima un coste de electricidad de unos 220kWh, más el término de potencia para el periodo de 3,300kW, el coste del *kWh* es de 0,141422€ y el del término de potencia es de 0,116968€/día. El teléfono e internet con Movistar empresas saldría a 33,90€/mes, con velocidad real de 300Mb de bajada y 30 Mb de subida y llamadas a fijos nacionales gratis y sin horarios, con 550 minutos gratis de llamadas a móviles, al precio anterior habría que sumarle 14,38€ de la cuota de línea y el 21% de IVA. Por último el consumo de agua de agua aproximado para la oficina será de unos 20m^3. Consumo de agua:

Servicio/Concepto	Volúmenes	Precio unitario	Importe base (€)	Tipo IVA
Aducción				
Consumo	10	0,2965	2,965	10%
Cuota de servicio	-	-	3,915	10%
Distribución				
Consumo	10	0,1335	1,335	10%
Cuota de servicio	-	-	1,78	10%
Depuración				
Consumo	10	0,3115	3,115	10%
Cuota de servicio	-	-	1,75	10%
Alcantarillado				
Consumo	10	0,1094	1,094	10%
Cuota de servicio	-	-	0,595	10%

Tabla 32: Coste mensual de agua

5.2.4 Coste total.

El coste total del proyecto para el periodo de duración estimado, de acuerdo a lo que se ha especificado con anterioridad será de 38.443,20403€. Ahora se procede a desglosar exactamente de donde proviene la cifra citada con anterioridad en la siguiente tabla:

Concepto	Periodo	Precio mensual	Importe total (€)
Personal			
Desarrollador	6 meses	2.100	12.600
Jefe de proyecto	6 meses	4.100	24.600
Material			
Material de oficina	1 vez	-	5
Ordenador, pantalla, teclado y ratón	1 vez	-	1.159,95

Indirectos			
Café, leche y azúcar	6 meses	40	240
Agua	6 meses	19,8	118,8
Mueble	1 vez	-	320
Electricidad	6 meses	34,62188	207,73128
Teléfono e internet	6 meses	58,4188	350,5128
Total			38.443,20403

Tabla 33: Coste total

6.- Conclusiones y trabajos futuros

En esta sección se va a proceder a explicar detenidamente las conclusiones que se han alcanzado durante la realización de este proyecto, así como las cosas que se han aprendido en el desarrollo, los conocimientos adquiridos durante el estudio de la carrera que han ayudado a completar esta aplicación, también se van a explicar las complicaciones que se han tenido que encarar y como se han resuelto.

Además también se expondrán otras líneas de desarrollo que se pueden seguir a partir de la aplicación realizada en este proyecto para mejorarla en un futuro.

6.1 Conclusiones

Ahora se expondrán las conclusiones a las que se ha llegado durante la realización del proyecto tanto a nivel de producto, describiendo si se han cumplido todas las expectativas para la aplicación, como a nivel del proceso, relatando los problemas encontrados y las soluciones que se han ido aplicando a cada uno de ello y por último las conclusiones personales sobre el proyecto, en los que se relatarán los conocimientos adquiridos con anterioridad que se han aplicado y los que han sido necesarios obtener para poder completar las tareas realizadas.

6.1.1 Producto

Al final se han cubierto todos los objetivos propuestos para este proyecto software, lo primero que se quería hacer era la obtención de datos de conexiones *ssh* de los servidores que se registrasen en la aplicación, para ellos se ha utilizado una gema de *ruby* (que es como se denomina a las librerías en este lenguaje) que se encarga de realizar conexiones *ssh* con otro servidores, y luego a través del comando *last* ejecutado en el servidor destino se procesa la salida para obtener los datos que luego muestra nuestra aplicación.

Otro de los objetivos a alcanzar en este proyecto era la gestión de varios servidores utilizando la herramienta generada durante este proyecto, y para ello se ha habilitado el almacenamiento de distintos servidores con distintos usuarios y contraseñas para poder conectarse a ellos a través de *ssh*, además se almacenan las contraseñas cifradas en la base de datos para evitar que se vean comprometidas si alguien consigue acceso a la base de datos de la aplicación.

Para seleccionar las fechas en las que se mostrarán las conexiones recogidas por la aplicación se ha dispuesto de un calendario en el que se pueden seleccionar no solo el mes en el que se quieran visualizar las conexiones si no también el día, además no solo se pueden ver las conexiones para el conjunto de todos los servidores que haya dados de alta en la aplicación, también se puede realizar el visionado de datos para un servidor concreto viendo en las gráficas el número de conexiones por usuario a dicho servidor, o incluso si se selecciona un día concreto, las que se han realizado por usuario y hora del día, para poder así monitorizar mejor las horas de mayor y menor uso de los distintos servidores que se tiene bajo la supervisión en la aplicación web.

Además otro de los objetivos que la aplicación debía cumplir era la de disponer de diferentes gráficas para la visualización de datos de una forma sencilla e intuitiva, para ello se han generado 4 gráficos diferentes que aportan información útil para el usuario de la

aplicación de una forma rápida y eficiente, la primera de las gráficas es la denominada de líneas, en la que cada línea de un color distinto representa cada uno de los servidores que hay registrados en CONAD y se muestran frente al tiempo para ver su evolución y su carga a lo largo del mes, o del día concreto en caso de seleccionarse dicha opción. La segunda es una gráfica de barras que muestra los totales de conexiones para cada servidor en el periodo de tiempo seleccionado, ya sea un mes o un día. Como una combinación de ambas surge la tercera de las gráficas, la gráfica de barras apiladas, esta nos muestra los totales para cada servidor para cada día u hora según si estamos visualizando un día o un mes, pero los muestra apilados unos encima de otros de forma que uno puede hacerse una idea aproximada de que servidores han tenido mayor tráfico *ssh* para un día u hora determinados. Por último el cuarto gráfico es un gráfico de donut que lo que muestra es la proporción del total de conexiones para un periodo de tiempo que se ha realizado para cada servidor de los que hay disponibles en la aplicación, es muy útil para poder comprobar que servidores tienen un mayor uso y tráfico con respecto al resto.

El último de los objetivos planteados para este proyecto era la posibilidad de poder tener la aplicación en diferentes idiomas de forma que su uso no quedase limitado a hispano-parlantes o anglo-parlantes, si no que al máximo número de personas posible, para ello se han creado diferentes *paths* y un selector de idiomas, de modo que cada usuario pueda utilizar la aplicación con el lenguaje que le sea más cómodo sin perder información por problemas con el idioma por defecto de la aplicación.

6.1.2 Proceso

Durante la realización de la aplicación han surgido múltiples problemas con respecto a lo que se había planificado para el desarrollo y que se han tenido que ir solventando durante el proceso de creación de la aplicación, ahora se va a proceder a explicarlos, así como las soluciones que se le han ido aplicando a cada uno de ellos has conseguir llegar a lo que es ahora mismo CONAD.

El primer problema al que hubo que enfrentarse tras la elección del *framework* de desarrollo (en este caso *Ruby on Rails*) fue aprender lo necesario para la creación de una aplicación de cero, dado que en otros desarrollos anteriores con dicho *framework* se realizaron sobre aplicaciones que ya estaban creadas con anterioridad y sobre las que se realizaron cambios y mejoras.

Tras esto otro problema que surgió fue la elección de la librería (gema) apropiada para la realización de las conexiones *ssh* necesarias para la obtención de los *logs* del comando *last* gracias a los cuales se obtienen las conexiones en las que luego se basa toda la aplicación. Se eligió *net-ssh* que permitía obtener simplemente la salida del comando *last* de una forma cómoda y sencilla, pero a su vez esta gema planteaba otro problema mucho más complicado, dicho problema es que para la realización de las conexiones *ssh* la gema necesita las contraseñas “abiertas”, pero evidentemente almacenarlas de esta manera en la base de datos supondría una grave vulnerabilidad, dado que si alguien consiguiese las contraseñas y usuarios de la base de datos podría conectarse a todos los servidores almacenados en la aplicación, por tanto había que encontrar una solución que permitiese utilizar las contraseñas pero que no las almacenase de forma abierta, para ellos se ha realizado un cifrado *AES*, que se aplica en la

creación de un nuevo servidor, o en su edición en caso de modificarse las contraseña, y que se descifra por código únicamente cuando se realiza la descarga del log del comando *last*, aunando así la funcionalidad y la seguridad en la aplicación.

Otra de las partes complicadas del proyecto ha sido la inclusión del calendario, no solo por encontrar una gema que proporcionase el tipo de calendario que se necesitaba, sino también porque hacía falta que permitiese pulsar en un día y que llevase a ese día, también hubo que cambiar la lógica de la aplicación de modo que en lugar de mostrar los 30 últimos días desde la fecha actual mostrase meses naturales. Para conseguir que el calendario permitiese ir al día seleccionado fue necesario modificar la gema para permitir que las celdas del calendario aceptasen clases y atributos diferentes a los predefinidos.

Una vez añadido el calendario se hizo necesario incluir las diferentes gráficas que luego permitirían ver los datos de una forma más atractiva e inteligible, para ello se seleccionó la librería d3js, la cual me era desconocida, lo que implicó no solo el desarrollo de las diferentes gráficas si no también aprender el manejo de la librería, de modo que a base de modificar ejemplos de gráficas que me parecían interesantes para la aplicación y adaptando la recepción de datos para que pudiesen utilizarse para dibujar las diferentes gráficas logrando así que no hiciese falta recargar la página cada vez que se seleccionaba una gráfica distinta a la que ya se estaba mostrando.

Por último el reto al que hubo que enfrentarse fue al de permitir múltiples idiomas en la aplicación ya que pese a que *Ruby on Rails* está preparado para poder funcionar con varios lenguajes al mismo tiempo ha sido necesaria una configuración que lo permitiese, además del modo que se requería para esta aplicación que es a través de *URLs* que contengan el nombre reducido del idioma en el que se quiere ver el contenido de la página, de este modo se permite poder enviarle a alguien la dirección en el idioma del destinatario o cambiar al idioma que se quiere desde la *URL* aunque no se esté entendiendo el idioma que se está visualizando en la aplicación en el momento en el que el usuario ha accedido a ella, además se ha tenido que realizar la traducción de todos los texto a los 2 idiomas disponibles para la aplicación por ahora.

6.1.3 Personales

Las primeras asignaturas que han sido necesaria para la realización de este proyecto han sido las de Ingeniería del software, dado que es una de las asignaturas que enseñan cómo se ha de gestionar un proyecto software de principio a fin, los diferentes documentos de los que se ha de componer, así como de la correcta redacción de los mismos, de las secciones que tiene que contener y como hay que realizarlos de forma apropiada. Además también es muy importante la asignatura de Sistemas Informáticos que en el fondo es la que sería una asignatura práctica de lo que luego va a ser la realización de un proyecto software de verdad similar a lo que ha sido la realización de este Proyecto Fin de Carrera, ya que no sólo se encargaba de la parte de documentación sino que además contaba con una parte práctica en la que se tenía que realizar una aplicación completa de acuerdo a unos requisitos de cliente.

En este caso al tratarse de una aplicación web han sido imprescindible los conocimientos sobre conexiones y protocolos adquiridos en las asignaturas de telemática y redes de ordenadores, dado que sin estos conocimientos no se hubiese podido realizar la

configuración de los ficheros *nginx* para permitir la conexión de la aplicación a internet y poder manejar dominios y subdominios en una misma máquina , así como tener varias aplicaciones diferentes ejecutándose en distintos subdominios o puertos. También ha ayudado a conocer los protocolos *ssh* imprescindibles para poder realizar la aplicación, así como el *ssl* que hace falta para las conexiones https y conocer cómo hay que configurar los certificados necesarios para poder hacerlo funcionar, también ha sido necesaria para poder configurar las entradas de DNS para poder manejar dominios.

También han sido necesarias las asignaturas de bases de datos, dado que *Ruby on Rails* utiliza *ActiveRecord* para la gestión de los modelos del patrón Modelo Vista Controlador que implementa, y *ActiveRecord* está pensado en principio para modelos relacionales como SQL, en este caso la base de datos es *PostgreSQL* con lo que el conocer modelos relacionales de bases de datos y también algo de lenguaje *SQL* y cómo funcionan estas bases de datos facilitando bastante la labor de trabajar con los datos y de diseñar los modelos de la aplicación.

Del mismo modo todas las asignaturas que han implicado el aprender a programar sobre todo con paradigmas orientados a objetos como por ejemplo ADA (Análisis y diseño de algoritmos), Programación o Paradigmas de la programación, han ayudado a que pese a no desconocer un lenguaje como *Ruby* no haya sido demasiado complicado el aprendizaje ya que guarda similitudes con otros lenguajes estudiados con anterioridad en la carrera como Java.

Pero sin lugar a dudas las asignaturas relacionadas con Sistemas Distribuidos han sido de las que mejor han venido para el desarrollo de esta aplicación dado que se trata de una aplicación web basada en una arquitectura de Cliente – Servidor y sin los conocimientos necesarios al respecto hubiese sido mucho más complicado comprender exactamente en que consiste el *framework Ruby on Rails* y no se le hubiese podido sacar todo el partido necesario para el correcto desarrollo de la aplicación cumpliendo con las necesidad y requisito solicitados a la conclusión del proyecto. A la hora de procesar las conexiones de los distintos servidores la aplicación se sirve de un sistema de procesos que se ejecutan en paralelo en el *background*, para poder conocer y entender los conceptos que hay detrás de este sistema fueron necesarias asignaturas como Sistemas Operativos para saber de qué manera se llevaban a cabo dichas tareas, además también era imprescindible conocer los conceptos como las partes críticas del código o los bloqueos que podían producirse si se utilizan mal los recursos, también para saber cómo paralelizar tareas correctamente contaba con los conocimientos que me proporcionó Arquitectura de los computadores II que estaba orientada precisamente en esa dirección.

Como nuevos conocimientos adquiridos, he tenido que aprender la configuración y despliegue de aplicaciones *Rails* en servidores, así como la configuración de *nginx* para poder redirigir las peticiones dentro del servidor a los diferentes servicios y aplicaciones que se tengan instalados en él. También ha sido necesario aprender a trabajar con máquinas virtuales dentro de un servidor. A parte de esto que va enfocado fundamentalmente con la parte de configuración y despliegue de aplicaciones, con respecto al desarrollo de aplicaciones en sí, ya tenía ciertas bases en el desarrollo en *Ruby on Rails*, pero este proyecto me ha ayudado a aprender a crear un proyecto de cero con este *framework*, así como me ha ayudado a mejorar

mi habilidad desarrollando en *Ruby*, además para la realización de la parte de *front* de la aplicación he tenido que aprender a utilizar varias librerías de *javascript*, la primera y fundamental es *jQuery*, que es en la que se suelen basar el resto de librerías, además para la visualización general se ha utilizado *bootstrap*, que es un paquete de estilos y *javascript* básico que mejora las los componentes mínimos de *HTML5* e incluye algunos nuevos. Pero de todo lo aprendido durante el Proyecto Fin de Carrera lo más complicado ha sido aprender a utilizar la librería *d3js*, la librería de *javascript* utilizada para la generación de las gráficas que se pueden visualizar en la aplicación, fue complicado porque no sólo se tenían que mostrar las diferentes gráficas, si no que debían trabajar todas con los mismo datos y a la vez se tenía que permitir una transición no traumática (con recarga de página) entre los diferentes tipos de gráficas disponibles.

6.2 Trabajos futuros

En esta sección se van a describir las posibles mejoras que podrían realizarse sobre el proyecto presentado en este documento así como también se van a exponer otros ámbitos en los que este proyecto podría ser útil más allá del original por el cual se creó.

Primero, se podría hacer que la actualización se realizase de una forma automática teniendo lugar cada vez que se cumpliese un periodo preestablecido, de este modo el usuario dispondría siempre de información más o menos actualizada cuando entrase en la aplicación sin necesidad de llevar a cabo ninguna acción, aunque como se ha explicado con anterioridad a su vez tampoco se ha investigado mucho en ello para este proyecto dado que produce que no se tenga información actualizada en el momento que el usuario final realmente quiera obtenerla, en cualquier caso se podría investigar sobre una manera de mantener ambos sistemas de una forma simultanea de forma que el usuario final pueda tener información más o menos actualizada y al mismo tiempo si necesita que la información se procese en el momento pueda solicitarla.

Otra posible mejora que se le podría incluir a este sistema sería la inclusión de mecanismos de autenticación, de este modo un usuario cualquiera no tendría acceso a todos los servidores incluidos en la aplicación, se podría implantar un sistema de roles y permisos que permitiesen incluir servidores que sólo pudiesen ser vistos por el usuario que los creo, de esta manera no sería necesaria una instancia de la aplicación para cada usuario que quisiese utilizarla, en cualquier caso se acordó que para esta primera versión del proyecto la autenticación se realizaría a través de http de modo que *ruby* no tomaría parte. De todos modos existen gemas que se encargan de la autenticación como *devise* y otras que se encargan de la gestión de roles y permisos como por ejemplo sería el caso de *can-can-can*.

Otra mejora que vendría muy bien para esta aplicación sería el incluir filtros en la visualización de las gráficas de modo que por ejemplo, si se quieren visualizar sólo los datos de determinado servidor o servidores se vea la información solo de esos en la gráfica y no de todos, y en el caso de estar en la visualización de un servidor concreto, que el filtro se pudiese realizar sobre los diferentes usuarios que han utilizado el servidor en el periodo de tiempo que se esté mostrando en el momento.

Además de las mejoras habría que tener en cuenta otro ámbitos en los que se podría utilizar la aplicación aparte de su función original que era la monitorización de conexiones *ssh*

para los ordenadores de la aulas informáticas, de forma que pudiese comprobarse que ordenadores tienen un uso mayor de modo que se pueda tener un seguimiento más de cerca porque será más susceptible de tener una avería, o necesitará ser sustituido con mayor celeridad en caso de tener problemas. Otro posible ámbito de uso distinto al original podría ser la monitorización de servidores pero en lugar de centrarse en el uso de forma comparativa con otros servidores que cumplen características similares se puede enfocar por ejemplo en servidores *git*, para ver las conexiones que se le realizan y estudiar comportamientos como horas a las que la gente suele hacer *push* al repositorio o la frecuencia con la que la gente sube o baja su código del repositorio, el número de conexiones media que realiza un usuario de *git* cuando quiere trabajar con el repositorio, ya sea para obtener información o para actualizarlo.

Otro posible ámbito relacionado con el anterior podría ser más enfocado a seguridad, dado que la aplicación *CONAD* monitoriza conexiones *ssh* a un servidor, si el servidor tiene información sensible se puede ver las conexiones que se han realizado a él quién las ha realizado y a qué horas de modo que sería sencillo tener controlado el uso de esos registros restringidos de forma que si hay algún uso indebido de ellos se pueda encontrar de manera rápida y eficaz a la personas o personas responsables.

Una mejora que ya conllevaría realmente un desarrollo completo sería el poder incluir otro tipo de conexiones a la monitorización de modo que no solo fuesen *ssh* sino que también pudiesen ser http o de errores, pero esto supondría la generación de librerías que se conectasen con la herramienta cuando se produjesen errores o conexiones, o si no definir un API, para que la gente pueda conectar sus aplicaciones web a la herramienta, pero como se ha dicho esto supondría cambios de base en el planteamiento del proyecto aunque bien es cierto que gran parte de lo existente podría ser reutilizable.

Otra posible mejora también podría ser la eliminación de datos después de un tiempo determinado para evitar llenar la base de datos con información obsoleta o que no se va a volver a consultar, se puede programar una tarea que limpie los registros de conexiones con una antigüedad mayor a un valor que se haya determinado con anterioridad.

Apéndice 1: Creación de usuario en un servidor Unix para albergar la aplicación

Primero se añade una nueva cuenta en el servidor en el que se va a alojar la aplicación, esta cuenta va a ser la encargada de realizar todas las acciones en nombre de nuestra aplicación web. Para este manual se va a crear un usuario con el nombre 'conad' que parece el más indicado para evitar ambigüedad en caso de que el servidor albergue más aplicaciones web, en todas las líneas de comando que se adjuntan en el manual se puede sustituir conad por el nombre de usuario que se haya elegido.

Se va a instalar *Nginx* que es imprescindible para poder hacer funcional la aplicación tras el despliegue.

```
$ apt-get update
$ apt-get install nginx
```

Ahora, lo primero que hay que hacer es comprobar si existe el grupo admin al que pertenecerá nuestro usuario, y si no existe se crea:

```
$ sudo groupadd admin
```

Después se crea un usuario que pertenezca a dicho grupo y que tenga carpeta home:

```
$ sudo useradd -m -g admin conad
```

A este usuario se le puede asignar una contraseña de la sieguiente manera:

```
$ sudo passwd conad
```

Se modifica el fichero sudoers

```
$ sudo visudo
```

y se sustituye:

```
%admin ALL=(ALL) ALL
```

por:

```
%admin ALL= NOPASSWD: ALL
```

Con este cambio en el fichero /etc/sudoers se consigue que los usuarios del grupo admin no tengan que insertar la contraseña cuando ejecuten el comando sudo, esto es necesario porque durante el proceso de despliegue automático se realizan acciones que requieren de permisos de superusuario.

Apéndice 2: Instalación y configuración de PostgreSQL para el proyecto.

En la base de datos es donde se van a ir almacenando las conexiones a los distintos servidores así como la información *ssh* de los servidores a lo que se va a conectar para poder realizar la monitorización. Se ha seleccionado una base de datos relacional que se adapta mejor a la abstracción de los modelos del MVC con las entidades en la base de datos. Para la instalación de la base de datos postgresql en *Ubunutu* lo primero que hay que hacer es actualizar los repositorios.

```
$ sudo apt-get update
```

Tras esto se procede a instalar los paquetes propios de la base de datos.

```
$ sudo apt-get install postgresql postgresql-contrib
```

Por defecto *Postgres* utiliza el concepto de "roles" para ayudar a la autorización y autenticación. Para gestionar las bases de datos en aplicaciones es recomendable crear un "role" propio para la aplicación que pueda crear bases de datos pero que no sea superusuario. Además por defecto *Postgres* crea al usuario postgres que es superusuario, para poder trabajar con la base de datos hay que hacer login en la terminal como dicho usuario.

```
$ sudo -i -u postgres
```

Una vez que se actúa como posgres ya es posible interactuar con la base de datos, ahora se va a proceder a la creación de un *role* llamado "conad" como nuestra aplicación.

```
$ createuser --interactive
Enter name of role to add: conad
Shall the new role be a superuser? (y/n) n
Shall the new role be allowed to create databases? (y/n) y
Shall the new role be allowed to create more new roles? (y/n) n
```

Tras esto se va a proceder a la creación de la base de datos, esto se debería hacer con "rake db:create" pero esta tarea no se ejecutaría nada más que una vez y por tanto no se incluye en el despliegue automático por lo que se tiene que hacer durante la implantación. Se va a crear la base de datos para la aplicación, en este caso se va a llamar "conad"

```
$ createdb conad
```

Apéndice 3: Instalación de Rbenv para la instalación y gestión de ruby en el servidor.

Para la instalación de *rbenv* como gestor de versiones de Ruby, primero hay que añadirle al servidor una serie de librerías que van a ser necesarias posteriormente para la compilación e instalación de *Ruby*. Para este manual se va a explicar como instalar la versión '2.2.0' ya que es la que necesita nuestra aplicación para poder funcionar. Además de las librerías necesitadas por Ruby se añade la *nginx* que junto con la gema *unicorn* serán los encargados de hacer funcionar la aplicación en el servidor.

```
$ sudo apt-get update
$ sudo apt-get install git-core curl zlib1g-dev build-essential libssl-dev
libreadline-dev libyaml-dev libsqlite3-dev sqlite3 libxml2-dev
libxslt1-dev libcurl4-openssl-dev python-software-properties libffi-
dev
```

Ahora en la sesión de usuario en la que se va a realizar el despliegue, en este caso 'conad' se ejecutan los siguientes comandos:

```
$ cd
$ git clone git://github.com/sstephenson/rbenv.git .rbenv
$ git clone https://github.com/sstephenson/ruby-build.git
~/.rbenv/plugins/ruby-build
$ echo 'export PATH="$HOME/.rbenv/bin:$PATH"' >>
~/.bash_profile
$ echo 'eval "$(rbenv init -)"' >> ~/.bash_profile
```

Para los 'echo' dependerá de la terminal que se use, teniendo que sustituir '.bash_profile' por '.bashrc' en caso de ser un Ubuntu Desktop o por '.zprofile' en caso de usar *zsh*.

Tras la ejecución de estos comandos hay que reiniciar la terminal en la que se está trabajando para que los cambios tengan efecto. Después de esto se realiza la instalación de la versión '2.2.0' de Ruby.

```
$ rbenv install 2.2.0
```

Ahora se procede a la instalación de la gema *bundler* que se encargará de gestionar la instalación de todas las gemas que necesita la aplicación, este proceso tendrá lugar durante el despliegue automático.

```
$ gem install bundler
```

Por último se configura la versión de *Ruby* por defecto que va a utilizar la aplicación de modo que no haya problemas de compatibilidad de versiones o no sepa que *Ruby* es el que tiene que utilizar la aplicación.

```
$ echo "2.2.0" > .ruby-version
```

Apéndice 4: Instalación de Redis para su uso en el proyecto.

La instalación de *redis* es necesaria para poder ejecutar la aplicación en un servidor, es quien se encarga de controlar los distintos hilos de trabajo del servidor. En la aplicación se lanza un hilo para obtener las conexiones de cada servidor que se tiene registrado cada vez que se pulsa el botón de actualizar. Para instalar *redis* lo primero es actualizar los repositorios del servidor:

```
$ sudo apt-get update
```

Tras esto se añaden los paquetes necesarios para la instalación de *redis*.

```
$ sudo apt-get install build-essential tcl8.5
```

Ahora se va a proceder a la instalación de *redis* propiamente dicha:

```
$ cd
$ wget http://download.redis.io/releases/redis-3.0.2.tar.gz
$ tar xzf redis-3.0.2.tar.gz
$ cd redis-3.0.2
$ make
$ make test
$ sudo make install
$ cd utils
$ sudo ./install_server.sh
```

Y con esto quedaría *redis* instalado y funcionando en el servidor, después se eliminan los ficheros de *Redis*

```
$ cd
$ rm -rf redis-*
```

Apéndice 5: Configuración anterior al primer despliegue con Capistrano

Antes de realizar el despliegue en el servidor, conectados como el usuario encargado de la aplicación (para nuestras guías 'conad') hay que que crear una serie de archivos y de directorios para que se realice correctamente.

```
$ cd
$ mkdir app
$ mkdir app/shared
$ mkdir app/shared/config
$ mkdir app/shared/public
$ mkdir app/shared/public/images
$ mkdir app/shared/public/javascripts
$ mkdir app/shared/public/stylesheets
```

Con los comandos anteriores se crean todos los directorios que se van a necesitar para el despliegue automático de la aplicación con *Capistrano*. Ahora vamos a crear los ficheros necesarios también para que la aplicación funcione. Para ello en este manual se va a utilizar 'vim' pero se puede utilizar cualquier editor de texto de terminal con el que el usuario se sienta más cómodo.

```
$ vim app/shared/config/database.yml
```

```
staging:
  adapter: postgresql
  encoding: utf8
  database: conad_staging
  username: conad
  password: conad
  host: localhost
```

```
$ vim app/shared/config/private_key.aes
```

```
a07d2e7cc2d6bc3cb8b5161bafef2738
```

```
$ vim app/shared/config/secrets.yml
```

```
# Be sure to restart your server when you modify this file.

# Your secret key is used for verifying the integrity of signed cookies.
# If you change this key, all old signed cookies will become invalid!

# Make sure the secret is at least 30 characters and all random,
# no regular words or you'll be exposed to dictionary attacks.
# You can use `rake secret` to generate a secure secret key.

# Make sure the secrets in this file are kept private
# if you're sharing your code publicly.

development:
  secret_key_base:
b8ab76da473a150aa6247e28e2500373525508a582d2608285c01b283
0c00608cbacdc33e51e7e73aaab48af3d5b481996b40bda427ad1eaf3a2
f531be3ef976

test:
  secret_key_base:
d101d94d4d70ac8de11e6e5e17698e8b2f1c135c2ae752624ea85fdd7ce
c04b14ca2a2403f65cbf46829fd7d17c1a4ed9f1a0a0c1b2c20a041b8c0
a1b396a8dc

# Do not keep production secrets in the repository,
# instead read values from the environment.
production:
  secret_key_base: <%= ENV["SECRET_KEY_BASE"] %>

staging:
  secret_key_base: <%= ENV["SECRET_KEY_BASE"] %>
```

Apéndice 6: Configuración de Capistrano para el despliegue automático

El primer cambio que se debería realizar es en el fichero 'config/deploy'. En él se definen los parámetros por defecto para los despliegues independientemente del entorno en el que se realice.

```
#set :application, 'conad'
set :repo_url, 'git@bitbucket.org:sendorf/conad.git'
```

La única parte de este fichero que debería modificarse es el repositorio del que se va a obtener el código para los despliegues, se sustituirá *'git@bitbucket.org:sendorf/conad.git'* por la dirección del repositorio correspondiente.

El siguiente paso es la creación del fichero de entorno para el despliegue, en este caso se va a crear un entorno de *staging* (entorno de pruebas), a partir de este punto los siguientes pasos se pueden repetir para tantos entornos se requieran, otro entorno típico para trabajar es el de producción (*production*). Nota importante: Para que la aplicación funcione dentro de la carpeta 'config/environtments' tiene que haber un fichero .rb con las variables de entorno para dicho entorno, en este caso dentro de 'config/environments' habrá un fichero llamado 'staging.rb'.

Ahora se va a explicar que debe contener el archivo 'config/deploy/staging.rb' como mínimo para que el despliegue pueda ser satisfactorio.

```
set :application, 'conad'

set :tmp_dir, "/home/conad/tmp/capistrano"

role :app, %w{conad@10.192.45.9}
role :web, %w{conad@10.192.45.9}
role :db,  %w{conad@10.192.45.9}

# Extended Server Syntax
# ======================
# This can be used to drop a more detailed server definition into the
# server list. The second argument is a, or duck-types, Hash and is
# used to set extended properties on the server.

set :deploy_to, '~/app'
set :whenever_environment, 'staging'
```

Se procede ahora a explicar cada uno de los valores que se deberían escribir de acuerdo a las necesidades y valores que se tenga en el servidor de despliegue. 'set :application' es para fijar el nombre de la aplicación, en este caso el valor asignado es conad. Los tres roles definen el cliente *ssh* para cada uno de esos roles dentro del servidor, se tiene que poner el nombre de usuario de que se ha creado con anterioridad, en este caso 'conad' @ y la *ip* del servidor de despliegue. Y por último se definen la carpeta de despliegue, en este caso se está definiendo una carpeta 'app' dentro del home de usuario. Por último y dado que se utiliza la gema *whenever* para la gestión de tareas *cron* se define el entorno, para este caso 'staging'.

Ahora se va a proceder a explicar la creación de los ficheros necesarios para el correcto funcionamiento de la aplicación después de realizar el despliegue. Se necesitan los archivos de configuración de inicio de *Unicorn*, de *Unicorn* propiamente dicho y de *Nginx*. Primero vamos a explicar el de *Nginx*.

Dentro de la carpeta con el nombre del entorno de despliegue en este caso 'config/staging' va a haber 3 ficheros de configuración imprescindibles para poder poner en marcha el sistema, que son los 3 nombrados con anterioridad. Primero vamos a mostrar un ejemplo de 'unicorn.rb'.

```
root = "/home/conad/app/current"
working_directory root
pid "#{root}/tmp/pids/unicorn.pid"
stderr_path "#{root}/log/unicorn.log"
stdout_path "#{root}/log/unicorn.log"

listen "/tmp/unicorn.conad_staging.sock"
worker_processes 4
timeout 600
```

Para que este fichero funcione sólo habrá que modificar el valor de `root' en este caso apunta al 'home' del usuario creado para almacenar y gestionar la aplicación, además también incluye en la ruta 'app' porque en el fichero 'staging.rb' definimos 'app' como carpeta de destino del despliegue, pero esto también deberá modificarse y adaptarse a la configuración que se esté haciendo, y por último 'current' que tendrá un enlace simbólico a la última versión del código.

Ahora se procede a describir el fichero unicorn_init.sh, que es un script que se encarga de lanzar *Unicorn* y haciendo que funcione, es lo que permite que la aplicación pueda utilizarse como un servicio de Unix proporcionando las opciones de 'start', para inicia la aplicación, 'stop' para detenerla y 'restart' para reiniciar la aplicación como servicios básicos que más se van a utilizar, aunque también permite otros como 'force-stop'.

```
#!/bin/sh
set -e

# Feel free to change any of the following variables for your app:
TIMEOUT=${TIMEOUT-120}
APP_ROOT=/home/conad/app/current
PID=$APP_ROOT/tmp/pids/unicorn.pid
CMD="cd $APP_ROOT; bundle exec unicorn -D -c
$APP_ROOT/config/staging/unicorn.rb -E staging"
AS_USER=conad
set -u

OLD_PIN="$PID.oldbin"
```

En este fichero sólo se tiene que modificar la parte superior, como antes se tiene que configurar 'root', que será la misma que para el fichero anterior. Luego en 'CMD' habrá que poner la ruta relativa del fichero 'unicorn.rb', en este caso, 'config/staging/unicorn.rb-E staging' la última parte (-E staging) es para indicar el entorno en el que se tiene que ejecutar unicorn.rb, que en este caso será 'staging' pero que será otro si queremos hacer el despliegue para un entorno distinto. Tras esto solo queda definir el usuario en 'AS_USER', aquí habrá que poner la cuenta de usuario que se ha creado con anterioridad y que es con la que se va a realizar el despliegue.

El tercer fichero de la carpeta con el nombre del entorno dentro de la carpeta 'config' es 'nginx.conf' que se encarga de que las peticiones http que se realicen sobre el servidor sean respondidas por la aplicación.

```
upstream unicorn_conad_staging {
  server unix:/tmp/unicorn.conad_staging.sock fail_timeout=0;
}

server {
  server_name conad.*;
  root /home/conad/app/current/public;

  try_files $uri/index.html $uri @unicorn_conad_staging;
  location @unicorn_conad_staging {
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_set_header Host $http_host;
    proxy_redirect off;
    proxy_pass http://unicorn_conad_staging;
  }

  error_page 500 502 503 504 /500.html;
  client_max_body_size 4G;
  keepalive_timeout 10;
}
```

Los valores de este archivo van a ser los siguientes, para 'upstream', unicorn_[usuario]_[entorno], en nuestro caso si se han seguido las configuraciones por defecto de los manuales el valor será 'unicorn_conad_staging'. Dentro de las llaves hay que darle el valor a 'server unix:/tmp/unicorn.[usuario]_[entorno].sock ', en este caso 'server unix:/tmp/unicorn.conad_staging.sock '. Para este caso después de darle valor a 'server unix', se le va a dar a 'server_name', esta variable sirve para que cuando se haga una petición al servidor, las *URL's* que cumplan con el nombre o nombres de este campo se procesen a través de *CONAD*, para nuestro ejemplo, se van a procesar todas las URL's que comiencen por 'conad.' y cualquier subdominio, para eso, el valor es "conad.*", pero se puede configurar para otras URL's. Ahora lo siguiente que hay que configurar es '@unicorn_[usuario]_[entorno]' que en este ejemplo será '@unicorn_conad_staging' tanto para '$uri' como para 'location'.

Apéndice 7: Configuración posterior al primer despliegue con Capistrano

El despliegue con *Capistrano* se realiza con el comando 'cap [entorno] deploy' en este caso el entorno es 'staging' con lo que se realizaría con 'cap staging deploy'. Tras el primer despliegue en el servidor hay que llevar a cabo aún una serie de configuraciones para su correcto funcionamiento. Para realizar esto hay que iniciar sesión como administrador ('root').

Se crea el enlace simbólico a la configuración que se ha subido con el despliegue. Se crea apuntando a la carpeta 'current' para que se actualice cada vez que se realiza el despliegue

```
$ ln -s /home/[usuario]/app/current/config/[entorno]/nginx.conf
/etc/nginx/sites-enabled/unicorn_[usuario]_[entono]
```

Ahora se crea el enlace simbólico del script de iniciación de la aplicación en la carpeta init.d para que sea considerado un servicio del sistema.

```
$ ln -s /home/[usuario]/app/current/config/[entorno]/unicorn_init.sh
/etc/init.d/unicorn_[usuario]_[entorno]
```

Por último se hace que el servicio se inicie cada vez que el servidor se reinicia de modo que no haya que hacerlo de forma manual.

```
$ update.rc.d unicorn_[usuario]_[entorno] defaults
```

Bibliografía

01. (s.f.). *New Relic*. Recuperado el 10 de Enero de 2015, de New Relic: http://newrelic.com/

02. (s.f.). *Google Analytics*. Recuperado el 26 de Marzo de 2015, de Google Analytics: http://www.google.com/analytics/

03. (s.f.). *Nagios*. Recuperado el 20 de Abril de 2015, de Nagios: http://www.nagios.org/

04. (s.f.). *Site24x7*. Recuperado el 24 de Abril de 2015, de Site24x7: https://www.site24x7.com/

05. (s.f.). *Paessler*. Recuperado el 26 de Abril de 2015, de Paessler: http://www.paessler.com/

06. (8 de Agosto de 2016). *Waters, Kelly*. Recuperado el 5 de Mayo de 2015, de What Is Agile? (10 Key Principles of Agile): http://www.allaboutagile.com/what-is-agile-10-key-principles/

07. (s.f.). *1stwebdesigner*. Recuperado el 10 de Mayo de 2015, de 1stwebdesigner: http://www.1stwebdesigner.com/php-vs-ruby-vs-python/

08. (18 de Julio de 2009). *Purer, Klaus*. Recuperado el 10 de Mayo de 2015, de klau.si: http://klau.si/sites/default/files/php-vs-python-vs-ruby.pdf

09. (16 de Noviembre de 2012). *Renee*. Recuperado el 10 de Mayo de 2015, de udemy blog: https://blog.udemy.com/modern-language-wars/

10. (s.f.). *Cómo se calculan todos los costes de personal*. Obtenido de BBVA: http://www.bbvacontuempresa.es/a/se-calculan-todos-los-costes-personal

11. (s.f.). *Gigabyte KM6150 Kit Teclado + Ratón USB - Teclado*. Obtenido de PcComponentes: http://www.pccomponentes.com/gigabyte_km6150_kit_teclado___raton_usb.html

12. (s.f.). *PcComponentes*. Obtenido de Philips 246V5LSB/00 24" LED - Monitor: http://www.pccomponentes.com/philips_246v5lsb_00_24__led.html