Tema 1: Implantación de arquitecturas web

La arquitectura web y algunos modelos

Una aplicación web, o web en general necesita de una estructura que permita su acceso desde diferentes lugares (máquinas). Esta estructura es lo que se denomina Arquitectura Web (realmente este nombre se da también al diseño de toda la estructura).

La gran mayoría de las arquitecturas web en la actualidad se basan en un modelo **cliente/servidor**: una comunicación asimétrica en la que uno de los extremos ofrece uno o más servicios y el otro hace uso de él. Éste es el modelo sobre el que centraremos el curso, pero no hay que olvidar otros modelos como **P2P (peer-to-peer), B2B (bussiness to bussiness), etc.**

El término **servicio** es muy amplio y muchas veces confuso. Por ejemplo se puede considerar una web a la que acudimos a comprar productos un servicio en si misma, pero a la vez dicho servicio está compuesto de servicios de seguridad, de sesión, de transacciones, etcétera.

La estructura de una Arquitectura Web actual sigue el siguiente **modelo**:

1. Una **capa cliente**: es generalmente el navegador Web ejecutándose en el

ordenador del usuario final. Existen otras opciones más básicas pero en la actualidad la potencia y diversidad de los navegadores existentes (así como su gratuidad) han relegado las demás opciones a la práctica desaparición.

2. Un servidor Web (**capa de negocio**): La capa cliente puede acceder a diferente lógica y procedimientos que existen en la capa de negocio. Aquí la lógica puede ser mucho más compleja que en la capa anterior. Los componentes de esta capa pueden ser desde simples archivos HTML hasta Servlets de Java. Existen muchas tecnologías que pueden usarse en este nivel: por ejemplo scripting web como PHP, ASP o JSP a lenguajes de programación como TCL, CORBA y PERL.
3. Una **capa de datos**: Se compone de un sistema de almacenamiento acceso a datos que se utilizan para confeccionar la página Web. Generalmente es un gestor de bases de datos relacionales (SGDB) pero pueden ser ficheros de texto plano, ficheros XML, etc. Una opción cada vez más usada es la creación de ficheros XML a partir de datos almacenados en una base de datos y su presentación mediante alguna de las opciones vistas el curso pasado como por ejemplo XSLT.

La capa de negocio puede estar a su vez dividida en dos partes si el sistema es suficientemente grande o complejo. Puede dividirse en una capa de presentación y una capa de lógica de negocio.

- La **capa de presentación** se encarga de componer las páginas integrando la parte dinámica en la estática. Además también procesa las páginas que envía el cliente (por ejemplo datos en formularios). Algunas soluciones para esta subcapa son los ASP de Microsoft o los JSP de Java. Esta parte la realiza generalmente un servidor web.
- La **capa de lógica de negocio** lleva a cabo operaciones más complejas. Se corresponde con la implantación de un servidor de aplicaciones. Realiza muchos tipos de operaciones entre los que destacan:
    - Realizar todas las operaciones y validaciones.
    - Gestionar el flujo de trabajo (workflow) incluyendo el control y gestión de las sesiones y los datos que se necesitan.
    - Gestionar todas las operaciones de accesos a datos desde la capa de presentación.

En el caso de estar usando páginas web estáticas (no cambian en función de diversas variables) no existiría la capa de datos ya que estos van incorporados en los propios archivos de marcas que serán las conforman las páginas web.

Este supuesto es cada vez menos común. Debido a la introducción de dinamismo en las páginas, la estructura vista anteriormente se ha visto alterada sensiblemente:

- Los navegadores Web son capaces de interpretar diferentes elementos dinámicos autónomamente o mediante plugins (javascript, flash, etc.)
- Los servidores Web también pueden interpretar código para generar las páginas web. Así se pueden introducir pequeños programas que alteren el contenido o aspecto final de una página web dependiendo de diferentes elementos como el usuario que accede o la información solicitada en cada momento. El servidor web necesita de algún módulo adicional para poder interpretar este código. Generalmente se empotra en el propio servidor web para lenguajes de script o se incorpora en un servidor a parte (de aplicaciones) para los lenguajes más potentes. Algunos lenguajes que típicamente se usan en las páginas dinámicas en el servidor son PHP, Python, Ruby o Java. Estos lenguajes también permiten el acceso a la capa de datos y la intercalación de estos datos entre los elementos de la página final.

Un ejemplo del modelo completo estaría compuesto por un servidor Apache y un Tomcat que se conecte a una base de datos. Un ejemplo del simplificado sería un servidor LAMP.

Dónde se ejecute el código de dinamización de la página determinará si el lenguaje de programación es de entorno cliente o de entorno servidor como se estudiará en los otros dos módulos de contenido informático de este curso del ciclo.

A pesar de que el modelo Cliente/Servidor es el más extendido, el W3C describe cuatro modelos de arquitectura de servicios web. Estos conceptos los veremos por encima por no tener una gran influencia en el resto del curso:

1. El Modelo Orientado a Mensajes: Se centra en definir los mensajes, su estructura, la forma de transportarlos, etc. sin referencias al por qué de cada mensaje ni a su significado.

2. El Modelo Orientado a Servicios: Es el más extendido y el más complejo de todos. Aunque usa mensajes, su definición no se basa en ellos si no en qué se proporciona a los receptores de dicho servicio. El servicio lo lleva a cabo un agente y lo usa otro agente, utilizando mensajes para comunicarse. En este modelo se usan metadatos para formar de y acordar muchos aspectos de la comunicación en si misma.

3. El Modelo Orientado a Recursos: Este modelo se basa en la creación de recursos y su accesibilidad mediante redes.

4. El Modelo de Políticas: se basa en definir los comportamientos de los agentes que utilizan los servicios definiendo cómo accederán a los recursos

Estos modelos son en muchos casos complementarios y la definición de una arquitectura web puede hacerse desde diferentes enfoques.

En un caso práctico lo más habitual es que nuestra arquitectura se base en una definición estándar.

Un modelo simple para el despliegue de aplicaciones Web

En la actualidad la mayor parte de la información y lógica de un negocio debe ser accesible desde diferentes lugares. Aquí entran en juego las aplicaciones web.

Todos pensamos en tiendas online como modelo de una aplicación web, pero hay otros muchos como por ejemplo una aplicación de compra venta de activos entre dos bancos en el sector entre negocios (B2B).

Se puede imaginar inmediatamente que la **seguridad** es un aspecto clave en este tipo de aplicaciones, pero no el único. Muchas veces la **velocidad** y **estabilidad** de la comunicación y del servicio en sí pueden ser tanto o más críticas.

Cuando uno va a desarrollar e implantar una aplicación web debe tener en cuenta varios factores. Lo primero debe ser hacerse una **idea general de la aplicación** y de las diferentes **soluciones** que podemos utilizar. Se deben tener en cuenta las tres capas. Un error muy común es el uso de un único conjunto de tecnologías constantemente. Por supuesto conocer una tecnología es un punto a favor de su uso, pero muchas veces vamos a llevar a cabo una solución manifiestamente mejor únicamente por no haber considerado usar otras y afrontar su aprendizaje.

El siguiente aspecto a considerar sería el **coste**. Cuánto nos va a costar y qué presupuesto tenemos.

Estos factores hay que considerarlos **antes de firmar** ningún contrato e incluso antes de dar un presupuesto aunque sea orientativo.

Por ejemplo una compañía de venta de vehículos industriales quiere una aplicación web para publicar sus datos de ventas y que los comerciales puedan acceder a ella remotamente. Necesitaremos una base de datos en la que se almacenen los diferentes vehículos y sus ventas. También hará falta una lógica que mantenga todo el sistema actualizado y permita modificaciones. Además necesitaremos una capa de cliente con autenticación para que los diferentes vendedores puedan acceder al sistema, consultar y actualizar los datos.

Después de la evaluación se puede decidir no afrontar el proyecto por muchos motivos. Además de los costes ya mencionados podría darse el caso de que no tengamos los conocimientos o la infraestructura para llevar a cabo el proyecto.

Una vez vayamos a realizar el proyecto deberemos poner en práctica los conocimientos adquiridos durante el estudio de este ciclo, pero muchas veces es difícil hacerse una idea de cómo hacerlo; aquí nos orientan con un ejemplo y esta pequeña guía nos puede ser muy útil para organizar el front-end.

Qué es una aplicación web

Es una aplicación que se va a ejecutar a través de internet. Constará de dos partes (al menos) una en el lado servidor y otra que se ejecutará en la máquina del cliente en un navegador web. Las aplicaciones web se engloban en el concepto superior de aplicaciones distribuidas. El servidor pone a disposición del cliente diferentes recursos. Ejemplos de aplicaciones web son el correo electrónico web, las tiendas online, las redes sociales, etc.

Fases de un proyecto de una aplicación web

Se pueden considerar cuatro fases en el proyecto:

1. **Concepto**: Durante esta fase se debe obtener una idea clara y concreta de qué quiere el cliente. Además hay que obtener una idea general de cómo se llevará a cabo y de si es viable o no. Hay que determinar las limitaciones reales con que nos podemos encontrar. Por ejemplo la conexión a internet existente en la zona puede no ser suficiente para obtener los resultados previstos. Otro ejemplo de problema pueda ser que la tecnología necesaria sea demasiado cara. Es vital que al terminar esta fase se tenga una documentación que defina claramente los límites y objetivos del proyecto.

2. **Diseño**: Esta fase se centra en responder a cómo haremos la aplicación. Hay que concretar las tecnologías (tanto software como hardware) que usaremos y cómo se van a comunicar entre ellas. También hay que determinar los distintos módulos que usaremos y sus interfaces. Es muy importante realizar un plan de proyecto realista en el que se dividan las tareas y responsabilidades y se calculen

los tiempos para cada elemento así como su secuencia y dependencias. También hay que obtener una especificación funcional en la que se detallen tanto el funcionamiento como el flujo de la aplicación.

3. **Desarrollo**: En esta fase se debe desarrollar el proyecto en sí. Es muy importante llevar a cabo pruebas tanto unitarias como de integración así como gestionar una documentación del desarrollo y un control de versiones.

4. **Pruebas e implantación**: Cuando el proyecto está totalmente terminado es necesario probarlo intensivamente antes de ponerlo en producción. Es necesario tener en cuenta tanto nuestra aplicación como su comunicación con otros sistemas informáticos. Cuanto más se parezca el sistema de pruebas al real mejor. El último paso es la instalación y puesta en marcha del sistema. Es un momento crítico.

Este modelo es una simplificación de los que se utilizan habitualmente en ingeniería de software.

> Investiga en Internet las fases habituales en un proyecto de desarrollo de software y cuáles son los diferentes ciclos de vida que se usan. Discute en clase los descubrimientos con ayuda del profesor.

Una fase común a todos los proyectos informáticos y que no se incluye aquí es el **mantenimiento**. Este concepto incluye dos partes. El mantenimiento del servicio y corrección de errores y las mejoras. La primera consiste en asegurarse de que todo sigue funcionando y solucionar los posibles errores y “caídas” del servicio. La segunda consiste en ampliar el proyecto. Ambos casos suelen considerarse contratos a parte y por ello no se incluyen en el ciclo.

El orden correcto para el desarrollo es empezar de abajo a arriba. Es decir, primero la capa de datos, luego la de negocio y por último la presentación al cliente. Muchas veces se tiene la tentación de hacerlo al revés. Esto es debido a que no se han identificado bien las necesidades y objetivos de proyecto o no se ha realizado un diseño concreto. Es un error que nos conducirá a muchas más modificaciones y errores en nuestra aplicación.

Servidores web

Un **servidor web** es un programa o conjunto de ellos que proporciona un servicio a través de una red. La comunicación con un servidor web suele hacerse mediante el protocolo http (hypertext transfer protocol) que está englobado en la capa de aplicación del modelo OSI.

Muchas veces servidor web se usa como referencia también al **hardware** que lo aloja, pero esto es inexacto porque el mismo hardware puede albergar muchas otras funcionalidades o puede darse el caso de que un mismo hardware contenga varios servidores web (a veces simulados).

El objetivo de un servidor web es proporcionar los medios para permitir la comunicación entre dos o más programas o grupos de software sin importar la tecnología usada para crear y operar cada uno de ellos.

En la actualidad el servidor web más extendido con mucha diferencia es **Apache**. Por ello será en el que centraremos este curso. Existen muchos otros servidores web. Una forma fácil de consultar la lista y ver una comparativa muy general es visitando la Wikipedia. En esta otra se pueden leer las principales características de cada uno.

Los servidores web se engloban en un conjunto de sistemas más general que se denomina **modelo distribuido** porque el sistema no es unitario, está repartido entre diferentes máquinas o conjuntos de hardware. Este modelo tiene que afrontar algunos problemas que hay que tener siempre en cuenta:

1. La latencia y poca fiabilidad del transporte (por ejemplo la red).
2. La falta de memoria compartida entre las partes.
3. Los problemas derivados de fallos parciales.
4. La gestión del acceso concurrente a recursos remotos.
5. Problemas derivados de actualizaciones de alguna/s de las partes.

¿De qué se encarga cada nivel de la pila del modelo OSI? ¿Qué protocolos se incluyen en cada capa? Identifícalos y explica muy brevemente qué hace cada uno.

Servicios Web

Un **servicio web** es un concepto abstracto que debe implementarse mediante un **agente**: un pedazo de software que envía, recibe y procesa mensajes mientras que el servicio es el concepto de qué hace. El agente solo debe ajustarse a la definición de una interfaz (dos realmente, una hacia dentro (pila OSI) y otra hacia fuera) y puede modificarse o incluso rehacerse en otro lenguaje de programación sin ningún problema. El diseño se realiza siguiendo normas de modularidad para permitir estas modificaciones.

Es de vital importancia que el servicio web esté bien definido para posibilitar la comunicación entre ambos extremos. Por ello hay muchos estándares sobre servicios web que permiten la comunicación de un cliente genérico (por ejemplo un navegador web) con diversos servicios.

Generalmente la definición de un servicio se realiza en una API que especifica cómo comunicarse con el servicio.

El proceso para usar el servicio es como sigue:

1. El cliente y el servidor deben ser conscientes de la existencia del otro. En el caso más habitual es el cliente el que informa al servidor de su intención de usar el servicio pero también puede ser el servidor el que inicie el contacto. Si es el cliente el que comienza, puede hacerlo o bien conociendo previamente cómo localizar el servidor o usando el servicio para descubrir servicios (Web Service Discovery).
2. Ambas partes deben ponerse de acuerdo sobre los parámetros que regirán la comunicación. Esto no significa que discutan, solo que las normas y protocolos deben ser las mismas en ambas partes.
3. Los agentes de ambos lados empiezan a intercambiar mensajes. El servidor web necesita componer las páginas en caso de que lleven elementos multimedia e incluso necesitará realizar otras acciones si la página se crea dinámicamente.

Alternativas

Antes de decidirse a instalar nuestro propio servidor web, debemos tener en cuenta que no siempre es la mejor opción. Lo primero que debemos saber es qué quiere el cliente. Dependiendo del tamaño del servicio que vayamos a proporcionar y de la importancia de poder controlar todos los aspectos del servidor, podemos decidir usar otras posibilidades.

Por otro lado la máquina que necesitamos tendrá que tener mucha RAM y capacidad de almacenamiento a parte de soportar grandes cargas de trabajo. La conexión a internet también deberá ser potente y necesitaremos contratar una dirección IP estática.

Lo primero que se debe tener en cuenta es si nos interesa tener nuestro propio servidor web o contratar un servicio de almacenamiento web (web **hosting**). Realmente el término Web Hosting incluye el tener un servidor propio, pero en la actualidad se utiliza para denominar el alquilar espacio en un servidor de otra compañía. Generalmente esta compañía está dedicada a ello específicamente. Las ventajas de este caso son las obvias: no tenemos que preocuparnos de adquirir ni mantener ni el hardware ni el software necesario. Además la fiabilidad del servicio de una empresa especializada suele ser muy alta.

Existen casos en los que incluso hay tecnologías más específicas para nuestras necesidades. Cada vez es más habitual la existencia de sitios web en los que la apariencia no cambia pero el contenido es actualizado constantemente. Para estos casos se puede usar un **gestor de contenidos**. Con ellos se permite al usuario actualizar la información del sitio sin necesidad de que tenga conocimientos web concretos. Existen muchos gestores web, algunos comerciales y caros pero muy ponentes. También hay dos alternativas gratuitas muy extendidas: Joomla y Drupal.

Últimamente se considera incluso una alternativa más sencilla si se quiere una web informativa y bastante sencilla. Wordpress empezó siendo una plataforma para alojar blogs pero cada vez incluye más opciones y posibilidades de configuración permitiendo llevar a cabo páginas bastante atractivas. En la actualidad es un gestor de contenidos que incluso puede instalarse de manera autónoma como se indica aquí.

http://www.frameworx.co.za/

http://www.webdesignerdepot.com/

http://designshack.net/

Son algunos ejemplos de páginas hechas en Wordpress.

> Busca comparativas entre montar tu propio servidor, contratar un hosting externo o usar wordpress. ¿Qué funcionalidades crees que son más importantes? ¿Por cuál te decantarías para crear la web de una panadería? ¿y de un centro deportivo municipal? ¿la de la consejería de deportes de la Comunidad de Madrid la realizarías con el mismo sistema?
>
> Los gestores de contenido como Joomla, Drupal o Wordpress están generando bastantes ofertas de trabajo en la actualidad. Busca en Internet las principales características de cada uno y los motivos que te pueden llevar a decidirte por uno u otro.

¿Qué necesito para montar un servidor web?

Lo primero que necesitas es una **máquina** con una potencia capaz de atender las peticiones que vaya a procesar. Este punto es crítico y difícil de gestionar porque no sabemos cuál será la demanda y muchas veces es complejo estimar la carga de trabajo que soportará. Es muy recomendable que sea una máquina dedicada o que cumpla otras funciones relacionadas con intercambio de información en internet como gestionar correo electrónico o FTP.

También es vital que el **sistema operativo** que elijamos sea estable. No tiene ningún sentido elegir un sistema operativo que deje de estar funcional con facilidad. Es conveniente que lleve cierta seguridad y control de permisos integrado. Los sistemas más habituales son diferentes versiones de UNIX (por ejemplo Solaris) pero las diferentes distribuciones de Linux están tomando una posición fuerte por su bajo (o inexistente) coste. Windows también es una buena opción (sobre todo sus versiones servidor) pero es más caro que Linux y es más difícil de gestionar debido a la diversidad de configuraciones, opciones y funcionalidad que lleven integrados.

Lo siguiente que tendrás que conseguir es una **dirección IP estática**. Por supuesto debe ser una dirección de internet a no ser que tu objetivo sea montar una intranet. Nuestra máquina debe ser accesible desde redes remotas.

> Consulta cómo se puede contratar una dirección IP estática y cuánto cuesta.

Los nombres y direcciones de internet que conocemos se basan en un sistema llamado DNS que lo que hace es convertir esas direcciones legibles para nosotros en direcciones IP y viceversa. Si nuestra dirección IP cambia frecuentemente cuando alguien fuera a acceder a nuestra página esta le aparecería como no disponible a pesar de que todo el resto del sistema estuviera trabajando.

> ¿Qué es DNS? ¿Quién lo gestiona? ¿Cómo se consigue un nombre de dominio?
>
> Para alternativas como Wordpress o un hosting externo, ¿es necesario un nombre de dominio?

Existe la posibilidad de funcionar con una dirección IP dinámica mediante sistemas como http://dyn.com/dns/ que mantienen siempre actualizada nuestra dirección pero solo es recomendado (incluso en la propia página) para servidores con muy poca carga de conexiones y trabajo.

La **conexión a internet 24 horas** se da por garantizada pero también implica ciertos problemas como la adquisición de dispositivos de red que aguanten ese horario sin sobrecalentarse o saturarse.

El **software** del servidor del que ya hemos hablado y sin el que no podríamos trabajar.

Configurar nuestra máquina para que sea accesible pero impidiendo la conexión de desconocidos a partes del sistema críticas o que no queremos publicar. En definitiva, hay que mejorar la seguridad.

Instalación y configuración básica de un servidor web: Apache

Nosotros vamos a optar por instalar un servidor web Apache en un sistema operativo Linux. Es una de las opciones más extendidas y la posibilidad de obtener este software de manera gratuita disminuye mucho los costes pero no es la única razón. Apache destaca sobre otros servidores por:

- Tiene un diseño modular y altamente configurable.
- Es de código abierto por lo que existen muchas extensiones y herramientas de terceros.
- Funciona muy bien con Perl, PHP y otros lenguajes de script.
- Existen versiones para muchos sistemas operativos incluyendo Windows, Linux y Mac OS X.

La familia Apache 2 trajo muchas mejoras con respecto a la primera versión sobre todo en el plano de la flexibilidad, escalabilidad y portabilidad.

Lo más lógico sería instalar Apache en un sistema operativo de tipo servidor (Ubuntu Server) pero por motivos didácticos, vamos a instalarlo en una versión estándar con interfaz gráfica. Es menos seguro por lo que en un sistema en producción deberíamos optar por la otra opción. A pesar de usar un Linux con interfaz gráfica vamos a instalar todo desde la ventana de terminal, por lo que los pasos se podrán aplicar a un servidor.

Apache puede funcionar de forma autónoma, pero en la actualidad (más aún hablando de aplicaciones web) suele necesitar una base de datos y cada vez son más los sistemas que usan PHP. Por ello han proliferado los instaladores AMP (Apache, MySQL y PHP) que instalan y configuran los tres sistemas para que funcionen de manera conjunta. Esto requiere conocer en profundidad la configuración de los tres elementos para poder tener un servidor seguro. Para seguridad sobre PHP y MySQL habrá que referirse a los módulos propios.

XAMPP es la versión más usada para Windows y Linux. Dependiendo del sistema para el que vayan orientados, estos paquetes se llaman WAMP y LAMP genéricamente. La instalación de estos paquetes es tremendamente sencilla.

NOTA: En anteriores versiones de estos apuntes, se incluía una guía para realizar una instalación manual de Apache. He decidido quitarla porque este módulo se incluye en un ciclo sobre desarrollo y no sobre administración, porque es demasiado farragosa, porque no la usábamos durante el resto del curso y porque quitaba demasiado tiempo que luego echábamos en falta durante los últimos temas. Considero que un alumno que termina el ciclo debe ser capaz de realizar la instalación manual siguiendo una guía con los conocimientos de Linux adquiridos en Sistemas Operativos y los de Apache que se explican en este módulo.

Comprobación y eliminación de otras versiones

En la mayoría de las guías y tutoriales de internet verás que instalan primero MySQL. Esto es porque facilita la instalación, pero ¿y si tenemos que montar una base de datos en un servidor web que ya está funcionando? Nosotros instalaremos primero la opción más básica de nuestro servidor web.

Antes de nada, más aún en un servidor que vaya a publicar en la web, debemos asegurarnos de tener el control absoluto. Muchas versiones de Linux vienen con paquetes de Apache **preinstalados** así que lo primero será **eliminarlos** si existen. Esto mejora la seguridad porque nos aseguramos de estar instalando la última versión. Vamos a comprobar de manera conjunta si existen instalaciones de Apache, MySQL y PHP para poder dejar una máquina limpia.

```
dpkg --list|grep -e httpd -e apache -e mysql -e php
```

¿Qué hace la línea anterior?

En Ubuntu por defecto no viene instalado. Si está preinstalado el comando anterior devolverá algo. Si es así debemos comprobar las dependencias para eliminar todo. Si viéramos que aparece *ii* al lado del paquete estaría instalado y habría que eliminarlo. Supongamos que nos devuelve lo siguiente

```
     ii  apache2                              2.2.4-3build1            Next generation, scalable,
extendable web se
     ii  apache2-mpm-prefork                  2.2.4-3build1            Traditional model for Apache
HTTPD
     ii  apache2-utils                        2.2.4-3build1            utility programs for webservers
     ii  apache2.2-common                     2.2.4-3build1            Next generation, scalable,
extendable web se
     ii  libapache2-mod-php5                  5.2.3-1ubuntu6           server-side, HTML-embedded
scripting languag
     ii  libdbd-mysql-perl                    4.004-2                  A Perl5 database interface to
the MySQL data
     ii  libmysqlclient15off                  5.0.45-1ubuntu3          MySQL database client library
     ii  mysql-client-5.0                     5.0.45-1ubuntu3          MySQL database client binaries
     ii  mysql-common                         5.0.45-1ubuntu3          MySQL database common files
     ii  mysql-server                         5.0.45-1ubuntu3          MySQL database server (meta
package dependin
     ii  mysql-server-5.0                     5.0.45-1ubuntu3          MySQL database server binaries
     ii  php5-common                          5.2.3-1ubuntu6           Common files for packages built
from the php
     ii  php5-mysql                           5.2.3-1ubuntu6           MySQL module for php5
```

Buscamos los procesos de apache.

```
     # ps -wef|grep apache2
```

Suponemos que nos muestra

```
root      4006     1  0 09:41 ?        00:00:00 /usr/sbin/apache2 -k start
www-data  4025  4006  0 09:41 ?        00:00:00 /usr/sbin/apache2 -k start
www-data  4026  4006  0 09:41 ?        00:00:00 /usr/sbin/apache2 -k start
www-data  4027  4006  0 09:41 ?        00:00:00 /usr/sbin/apache2 -k start
www-data  4028  4006  0 09:41 ?        00:00:00 /usr/sbin/apache2 -k start
www-data  4029  4006  0 09:41 ?        00:00:00 /usr/sbin/apache2 -k start
www-data  4140  4006  0 14:24 ?        00:00:00 /usr/sbin/apache2 -k start
www-data  4141  4006  0 14:24 ?        00:00:00 /usr/sbin/apache2 -k start
www-data  4142  4006  0 14:24 ?        00:00:00 /usr/sbin/apache2 -k start
root      4157  4125  0 14:36 pts/0    00:00:00 grep apache2
```

Paramos Apache

```
# apachectl stop
```

Volvemos a buscarlo para asegurarnos de que ya no se está ejecutando.

```
# ps -wef|grep apache2
root      4162  4125  0 14:36 pts/0    00:00:00 grep apache2
```

Ahora buscamos MySQL

```
# ps -wef|grep mysql
```

Nos muestra

```
     root      3857     1  0 09:41 ?        00:00:00 /bin/sh /usr/bin/mysqld_safe
     mysql         3897    3857    0  09:41  ?               00:00:00  /usr/sbin/mysqld  --basedir=/usr
--datadir=/var/lib/mysql   --user=mysql   --pid-file=/var/run/mysqld/mysqld.pid   --skip-external-locking
--port=3306 --socket=/var/run/mysqld/mysqld.sock
     root      3898  3857  0 09:41 ?        00:00:00 logger -p daemon.err -t mysqld_safe -i -t mysqld
     root      4355  4310  0 18:54 pts/1    00:00:00 grep mysql
```

Paramos MySQL

```
     # /etc/init.d/mysql stop
      * Stopping MySQL database server mysqld      [ OK ]
```

Y comprobamos que ya no está.

```
     # ps -wef|grep mysql
     root      4395  4310  0 18:55 pts/1    00:00:00 grep mysql
```

PHP no hay que pararlo.

Eliminamos todos los paquetes que aparecían.

```
     # 3 remove php5-mysql libapache2-mod-php5 php5-common
     # apt-get remove libdbd-mysql-perl mysql-server mysql-server-5.0 mysql-client-5.0
```

```
# apt-get remove libmysqlclient15off mysql-common
# apt-get remove apache2 apache2-mpm-prefork apache2.2-common apache2-utils
```

Ahora que ya hemos visto cómo hacer una instalación y configuración de Apache 2 de forma manual, vamos a realizar otra instalación más estándar para continuar. Además esta instalación nos permitirá tener una versión limpia.

Instalación

Antes de instalar algo es importante tener actualizada la lista de paquetes (este comando no instala nada, solo actualiza la lista de paquetes)

```
sudo apt-get update
```

para luego actualizar la máquina entera (instalamos las versiones más recientes de las máquinas instaladas)

```
sudo apt-get upgrade
```

y luego ya instalamos

```
apt-get install apache2
```

Lo primero que debemos hacer es ver que ahora Apache está instalado en

```
cd /etc/apache2/
```

Existen diferentes versiones de Apache 2 por lo que es interesante comprobar cuál es la que estamos usando.

```
apache2 –v
```

lo que nos mostrará algo como esto

```
Server version: Apache/2.4.7 (Ubuntu)
Server built:   Jul 22 2014 14:36:38
```

> Configura e instala Apache en tu máquina virtual siguiendo los pasos indicados y comprueba el acceso desde la máquina virtual y desde la máquina anfitrión. Comenta en clase los resultados.

Podemos ver que ya existe un archivo apache2 en /etc/init.d/ que hace que se inicie Apache cada vez que encendamos la máquina.

```
sudo gedit /etc/init.d/apache2
```

Nos muestra un contenido muy complejo pero al que podemos echar un vistazo sabiendo que se basa en establecer qué hacer ante diversas órdenes (las más importantes son start, stop, restart y graceful).

Echa un vistazo al script sin entrar en detalles. ¿Qué es graceful? ¿para qué se usa aquí?

Hay muchas maneras de instalar Apache. Se pueden elegir módulos, configuración y rutas. Nosotros vamos a usar una que utiliza DSO (Dynamic Shared Objects). Esto es una configuración de Apache que permite añadir luego módulos sin necesidad de recompilar todo el servidor y por defecto en la instalación automática ya viene activada. Para nuestro curso es vital. El único pero es un descenso casi imperceptible del rendimiento por lo que considero muy recomendable usar este método.

Ahora abrimos el navegador web y cargamos la página para ver que todo ha ido bien. Escribimos http://localhost y debemos encontrar la página que pone “It Works!”. En la versión de Ubuntu además esta página ha sido modificada para que nos explique la organización de la configuración de Apache en esta distribución.

En la distribución estándar toda la configuración va en un único archivo pero se puede distribuir mediante llamadas a archivos de configuración externos. La configuración que nos encontramos aquí es bastante lógica y veremos que la relación entre ambas formas es sencilla de modificar y adaptar.

Podemos además colocar algún archivo HTML y comprobar que se muestra. Para ello iremos a http://localhost/nombre_archivo.html

Configura e instala Apache en tu máquina virtual siguiendo los pasos indicados y comprueba el acceso desde la máquina virtual y desde la máquina anfitrión.

Aunque el espíritu de un servidor es estar activo todo el tiempo posible, es importante poder pararlo, arrancarlo y reiniciarlo en ciertas situaciones. Todas estas opciones nos las explican en la documentación de Apache. Muchas de estas páginas de la documentación pueden

estar traducidas al español, pero prefiero enlazar las versiones en inglés ya que estarán siempre actualizadas cosa que puede no suceder para otros idiomas.

Hemos hablado antes de la opción “graceful”. ¿Qué sucede si usas “apachectl -k graceful”? ¿es lógico? ¿cuál es la diferencia con usar “graceful-stop”?

¿Cómo se debe parar el servidor para una operación de mantenimiento rutinario? ¿y si lo que queremos es que se carguen algunas modificaciones en los archivos de configuración?

Instalar un servidor con LAMP

Ahora vamos a empezar con una **nueva máquina virtual** a instalar un servidor con Apache, MySQL y PHP ya que como hemos comentado es la opción más habitual. Ni MySQL ni PHP son necesarios para montar un servidor de aplicaciones web.

Esta vez vamos a hacerlo con la opción más fácil para ver diferentes instalaciones. En el capítulo siguiente veremos las configuraciones necesarias para asegurar nuestro servidor.

Para aspectos de mayor seguridad en MySQL y PHP consulta los módulos del ciclo al respecto.

Lo primero es descargar e instalar tasksel.

```
sudo apt-get install tasksel
```

Luego instalamos directamente todo el paquete y seguimos las instrucciones.

```
sudo tasksel install lamp-server
```

Para probar Apache simplemente abrimos el navegador y consultamos localhost.

Para probar PHP, creamos un archivo prueba.php que solo contenga “<?php phpinfo(); ?>”

Para probar MySQL podemos usar cualquier método: conectarnos, instalar PHPMyAdmin, etc.

Si queremos instalar PHPMyAdmin escribimos

```
apt-get install phpmyadmin
```

y luego cargamos http://localhost/phpmyadmin

El directorio en el que se ha instalado Apache es

```
/etc/apache2
```

> Copia una nueva máquina virtual e instala el servidor LAMP. Prueba el correcto funcionamiento de todo.

Servidores de Aplicaciones

Un servidor de aplicaciones es un paquete software que proporciona servicios a las aplicaciones como pueden ser seguridad, servicios de datos, soporte para transacciones, balanceo de carga y gestión de sistemas distribuidos.

El término se acuñó para servidores de la plataforma Java en su versión Enterprise Edition, pero en la actualidad se extiende a muchas otras tecnologías.

Nosotros nos centraremos en Tomcat, un servidor de aplicaciones Java creado por Apache. Existen muchos otros como la integración de .NET en servidores de Microsoft, integración de PHP en un servidor para tener servidores de aplicaciones PHP, Zend Server, también para PHP, Barracuda, WebLogic de IBM, etc.

Apache Tomcat es un servidor de aplicaciones creado para alojar Servlets y Java Server Pages (JSP). Tomcat es gratuito y de código abierto pero no tiene nada que envidiar a otras soluciones comerciales. La versión que usaremos nosotros es la 7.

El funcionamiento de un servidor de aplicaciones necesita de un servidor web. Muchas veces vienen en el mismo paquete, pero realmente son dos partes diferenciadas.

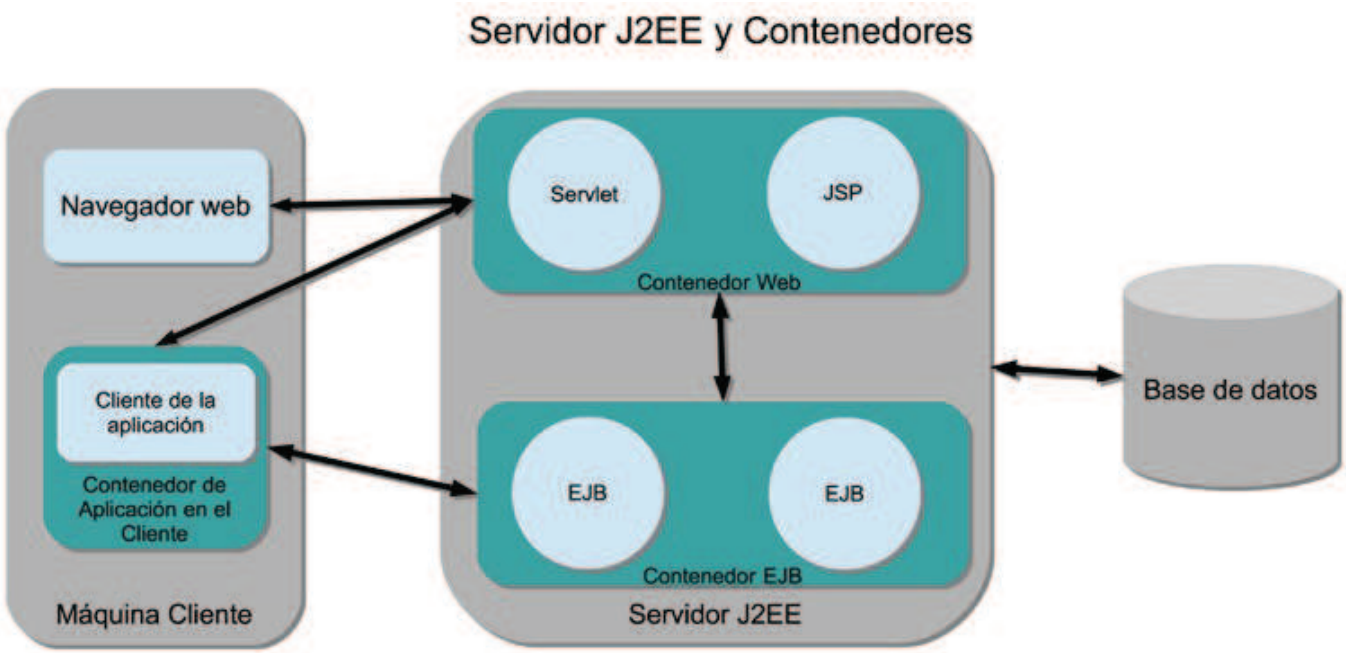
Cuando un cliente hace una petición al servidor web, este trata de gestionarlo, pero hay muchos elementos con los que no sabe qué hacer. Aquí entra en juego el servidor de aplicaciones, que descarga al servidor web de la gestión de determinados tipos de archivo, en nuestro caso servlets y JSP.

Si un cliente hace una petición al servidor pidiendo un JSP, esta llega al servidor Web que lee un archivo XML que le proporciona el servidor de aplicaciones y determina que el archivo lo gestionará el servidor de aplicaciones.

En el archivo XML también se incluye la dirección del servidor de aplicaciones y el servidor web le envía la petición mediante HTTP.

Contenedores

El término *contenedor* es otro bastante ambiguo como ya nos sucedió con *servidor*. En muchos casos se usa para referirse al propio servidor de aplicaciones e incluso al servidor web. Sin embargo, la acepción más extendida es otra.



Hay que tener en cuenta que los conceptos de los que estamos hablando surgieron principalmente de J2EE.

Los contenedores en los servidores de aplicaciones son una forma de aislar la ejecución de cada aplicación o de cada **instancia** de una aplicación del resto de instancias y de otras aplicaciones. Para cada ejecución proporcionan seguridad, soporte para transacciones, conexión remota y la gestión de los recursos precisos para la ejecución de la aplicación.

En referencia a la imagen anterior, conviene concretar algunos conceptos:

- El servidor J2EE es el programa que proporciona contenedores EJB y Web.
- El contenedor Enterprise JavaBeans (EJB) se encarga de la ejecución de los EJBs.
- El contenedor web se encarga de la ejecución de servlets y JSPs.
- El contenedor del cliente de la aplicación encarga de la ejecución de los componentes de las aplicaciones en la máquina del cliente.
- El contendor de applets se encarga de ejecutar los applets en el clientes. Está compuesto por un navegador web y un plugin Java.

Los contenedores de Tomcat se denominan Catalina.

Tomcat

Tomcat es un servidor de aplicaciones que puede funcionar por si mismo. De hecho es capaz de procesar peticiones en HTTP y servir archivos HTML con bastante eficiencia, pero no tan bien como lo hace Apache. Generalmente si lo que queremos es un servidor web con funcionalidad adicional lo mejor es disponer de ambos servidores trabajando conjuntamente, pero para casos donde casi todo va a ser lógica en java con Tomcat funcionando autónomamente es suficiente.

En este primer acercamiento a Tomcat lo instalaremos en una máquina virtual nueva. Más adelante veremos como integrarlo con Apache para que cada cual se encargue de hacer lo que gestiona mejor. En ese caso, Apache recibirá todas las peticiones y enviará a Tomcat lo que le corresponda a él.

En una máquina virtual de Java solo puede ejecutarse una instancia de Tomcat.

> ¿Qué hace la máquina virtual de Java? ¿En qué se diferencia de las máquinas virtuales que se crean con programas como Oracle Virtual Box o VMware?

Instalando Java

Existen dos versiones de Java muy extendidas en entornos Linux. Una es la oficial de Oracle y la otra se denomina OpenJDK y es un versión de código abierto.

OpenJDK

Esta versión es muy recomendable cuando se va a utilizar con otros sistemas de código abierto o de software libre por lo que nosotros **usaremos esta instalación**.

Actualizamos los repositorios e instalamos Java

```
sudo apt-get update
sudo apt-get install openjdk-7-jdk
```

comprobamos la versión de Java

```
java –version
```

lo que debería mostrar algo como

```
java version "1.7.0_65"
OpenJDK Runtime Environment (IcedTea 2.5.2) (7u65-2.5.2-3~14.04)
OpenJDK 64-Bit Server VM (build 24.65-b04, mixed mode)
```

Hay que establecer las variables de entorno de Java

```
sudo gedit /etc/environment
```

Y añadimos al principio las rutas de instalación de Java: el primero es el directorio donde están java y javac y el segundo el de jre.

NOTA: Ten en cuenta que a continuación indico dos ejemplos, pero que los nombres de las carpetas en las rutas dependen de la arquitectura de la máquina en la que se esté ejecutando el sistema.

```
     JAVA_HOME=/usr/lib/jvm/java-7-openjdk-i386
     JRE_HOME=/usr/lib/jvm/java-7-openjdk-i386/jre
```

o

```
     JAVA_HOME=/usr/lib/jvm/java-7-openjdk-amd64
     JRE_HOME=/usr/lib/jvm/java-7-openjdk-amd64/jre
```

Y en el mismo archivo añadimos al final del path

```
     :$JAVA_HOME:$JRE_HOME
```

El archivo entero debería quedar parecido a esto:

```
     JAVA_HOME=/usr/lib/jvm/java-7-openjdk-i386
     JRE_HOME=/usr/lib/jvm/java-7-openjdk-i386/jre
     PATH="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:$JAVA_HOME:
$JRE_HOME"
```

Aunque no es estrictamente necesario, reiniciamos la máquina virtual.

Java de Oracle

Tomcat es un servidor de aplicaciones programadas en Java por lo que antes de instalar Tomcat, debemos tener Java instalado y funcionando.

Lo primero es añadir unos repositorios que contienen el instalador de Java

```
sudo add-apt-repository ppa:webupd8team/java
```

Actualizamos el software de los repositorios con

```
sudo apt-get update
```

Para luego instalar Java

```
sudo apt-get install oracle-java7-installer
```

Podemos comprobar que tengamos funcionando la versión correcta

```
java –version
```

Debe aparecer en pantalla algo similar a esto:

```
java version "1.7.0_05"
```

```
     Java(TM) SE Runtime Environment (build 1.7.0_05-b05)
     Java HotSpot(TM) Client VM (build 23.1-b03, mixed mode)
```

Por ultimo, podemos comprobar el correcto funcionamiento yendo a http://www.java.com/es/download/installed.jsp en el navegador web.

Ahora hay que establecer las variables de entorno para que Tomcat pueda encontrar Java. Para ello editamos el archivo *environment*

```
     sudo gedit /etc/environment
```

Y añadimos al principio las rutas de instalación de Java: el primero es el directorio donde están java y javac y el segundo el de jre.

```
     JAVA_HOME=/usr/lib/jvm/java-7-oracle/
     JRE_HOME=/usr/lib/jvm/java-7-oracle/jre/
```

Y en el mismo archivo añadimos al final del path

```
     :$JAVA_HOME:$JRE_HOME
```

El archivo entero debería quedar parecido a esto:

```
     JAVA_HOME=/usr/lib/jvm/java-7-oracle/
     JRE_HOME=/usr/lib/jvm/java-7-oracle/jre/
     PATH="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:$JAVA_HOME:
$JRE_HOME"
```

Aunque no es estrictamente necesario, reiniciamos la máquina virtual.

Instalar Apache Tomcat

NOTA: Aunque Tomcat está en la versión 8 de manera estable, todavía no es la versión más extendida y no ha sido agregada a los repositorios estándar de Ubuntu. Además para el propósito de este curso no nos es necesario, por lo que voy a mantener Tomcat 7 como la versión de los apuntes un curso más. Si alguien quiere probar Tomcat 8 puede encontrar instrucciones de instalación en Internet como éstas o estas otras.

Es necesario tener Java instalado y configurado para el correcto funcionamiento de Tomcat. Los pasos son los mismos que ya vimos.

Ahora instalamos Tomcat

```
sudo apt-get install tomcat7
```

NOTA: En algunos casos, Tomcat no va a funcionar hasta que no configuremos las variables de entorno como se explica a continuación.

Ahora podemos abrir un navegador web y acceder a http://localhost:8080

Nos muestra la siguiente pantalla en la que es muy importante el siguiente párrafo:

Tomcat7 veterans might be pleased to learn that this system instance of Tomcat is installed with CATALINA_HOME in /usr/share/tomcat7 and CATALINA_BASE in /var/lib/tomcat7, following the rules from /usr/share/doc/tomcat7-common/RUNNING.txt.gz.

En el que nos indican la ubicación de dos variables de entorno muy necesarias en la configuración y uso de Tomcat. En la instalación manual, ambas apuntarían al directorio en el que descomprimimos Tomcat. Vamos a establecerlas como variables de entorno:

```
sudo gedit /etc/environment
```

en el añadimos (a continuación de las de Java)

```
CATALINA_HOME=/usr/share/tomcat7
CATALINA_BASE=/var/lib/tomcat7
```

Y las añadimos al path, quedaría como sigue:

```
JAVA_HOME=/usr/lib/jvm/java-7-openjdk-i386
JRE_HOME=/usr/lib/jvm/java-7-openjdk-i386/jre
CATALINA_HOME=/usr/share/tomcat7
CATALINA_BASE=/var/lib/tomcat7
PATH="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:$JAVA_HOME:
$JRE_HOME:$CATALINA_HOME:$CATALINA_BASE"
LANGUAGE="es:en"
LANG="es_ES.UTF-8"
LC_NUMERIC="es_ES.UTF-8"
LC_TIME="es_ES.UTF-8"
LC_MONETARY="es_ES.UTF-8"
LC_PAPER="es_ES.UTF-8"
LC_IDENTIFICATION="es_ES.UTF-8"
LC_NAME="es_ES.UTF-8"
LC_ADDRESS="es_ES.UTF-8"
LC_TELEPHONE="es_ES.UTF-8"
LC_MEASUREMENT="es_ES.UTF-8"
```

CATALINA_HOME indica el directorio de instalación de Tomcat.

CATALINA_BASE indica el directorio de una instancia de Tomcat. Si tenemos más de una instancia, CATALINA_BASE será diferente para cada una. En algunas instalaciones (aunque no en esta) ambas variables de entorno apuntan al mismo directorio.

Reiniciamos la máquina para que las variables se carguen.

El archivo que carga Tomcat por defecto es */var/lib/tomcat7/webapps/ROOT/index.html*

Tomcat está instalado en */etc/tomcat7*

Como podemos ver esta versión de Tomcat ya lo instala como servicio y hace que se inicia automáticamente al encender la máquina. El archivo donde está configurado esto es mucho más complejo que el que hicimos nosotros en la instalación manual; puedes consultarlo en

```
gedit /etc/init.d/tomcat7
```

Instalando paquetes adicionales

Ya tenemos Tomcat instalado y funcionando, pero como pudimos leer en la página de inicio por defecto de Tomcat, no se han instalado ni la documentación, ni los ejemplos ni la aplicación de administración. En un servidor de producción esta puede ser la configuración correcta, pero para nuestro propósito didáctico es muy recomendable instalar los paquetes.

```
sudo apt-get install tomcat7-docs
sudo apt-get install tomcat7-examples
sudo apt-get install tomcat7-admin
```

Ahora podemos acceder a cada uno de ellos mediante el correspondiente enlace de la página de inicio.

Instala Tomcat y los paquetes adicionales en un máquina virtual nueva.

Ya he comentado antes cómo viene apareciendo en las noticias tecnológicas que Java en inseguro en la red y que se debe desactivar en los navegadores. Esto puede parecer el fin de Java, pero… desactiva Java en tu navegador y prueba a ejecutar los ejemplos que acabamos de instalar ¿y ahora qué?

Si pinchas en el enlace que aparece para ver el código de un ejemplo verás que contiene código pero si muestras el código de la página resultante de ejecutar el ejemplo en el navegador podrás observar que no, ¿qué indica esto?

NOTA: Al igual que sucedía en Apache, existía un apartado sobre la instalación manual de Tomcat que he decidido eliminar por los mismos motivos.

Usuarios de Tomcat

Lo primero es configurar los usuarios de Tomcat. Para ello debemos editar el archivo *tomcat-users.xml* que está en el directorio *conf*

```
sudo gedit /etc/tomcat7/tomcat-users.xml
```

Lo que estamos haciendo es añadir un usuario administrador así que el fichero, en la parte final debería quedar parecido a lo siguiente, con el nombre de usuario y contraseña que queramos, por supuesto.

```
     <tomcat-users>
          <role rolename="manager-gui"/>
          <role rolename="manager-script"/>
          <role rolename="manager"/>
          <role rolename="admin-gui"/>
          <role rolename="admin-script"/>
          <role rolename="admin"/>

          <user username="sergio" password="sergio" roles="manager-gui,admin-gui,manager,admin,manager-
script,admin-script"/>
     </tomcat-users>
```

Con este usuario y contraseña que ya hemos creado podemos cargar el manager, desde la página de inicio de Tomcat o directamente desde http://localhost:8080/manager/html

Se recomienda probar ejemplos de Tomcat para ver que todo vaya bien siguiendo el enlace en la página principal o en http://localhost:8080/examples/

Iniciar y parar Tomcat

Para iniciar Tomcat usaremos

```
     sudo /etc/init.d/tomcat7 start
     sudo /etc/init.d/tomcat7 stop
     sudo /etc/init.d/tomcat7 restart
```

según lo que queramos hacer.

Si quisiéramos que funcionara como servicio, sería:

```
sudo service tomcat7 start
```

y para pararlo

```
sudo service tomcat7 stop
```

Aplicaciones web

Una aplicación web se diferencia de una estándar en que se accede a ella a través de una red como Internet o una intranet por ejemplo. En muchos casos es una aplicación que se escribe en lenguajes soportados por los navegadores web (por ejemplo Javascript + HTML) y que necesita de un navegador web para ejecutarse.

El ejemplo más sencillo que se nos ocurre de una aplicación web es un programa que permita acceder a datos de una empresa desde el exterior de ésta. Aquí ya queda patente el principal problema que tienen las aplicaciones web: la **seguridad**. Muchas veces el problema es la necesidad de encontrar un compromiso entre la seguridad y la eficiencia en la aplicación; demasiada seguridad puede ralentizar el uso, aumentar el tráfico, etc.

Las principales **desventajas** de este tipo de aplicaciones son las que se derivan del uso de una red y del acceso simultáneo de varios (a veces muchos) usuarios. Las principales **ventajas** son el poder usar un navegador web como cliente (algo de lo que disponen todos los ordenadores en la actualidad) y la simplificación de las actualizaciones por no tener que actualizar los ordenadores uno a uno. Vamos a concretar un poco más:

Ventajas:

- No es necesario ningún tipo de distribución, instalación o actualización complejo de la aplicación. Simplemente el uso de un navegador compatible nos permitirá usar la aplicación. Muchas veces se crea la aplicación para un único navegador web lo que en mi opinión es un error. No es necesario cubrir todo el espectro pero dar al menos dos o tres opciones sería recomendable.
- No se necesitan máquinas clientes especialmente potentes. Prácticamente cualquier ordenador en la actualidad es capaz de ejecutar un navegador web. Si la aplicación es más pesada en el cliente se pueden necesitar recursos un poco más altos.
- Son fáciles de integrar con otras funcionalidades de servidor como el correo electrónico.
- Generalmente permiten eliminar los problemas derivados del uso de diferentes plataformas informáticas (arquitecturas, sistemas operativos, etc.).

- El uso de HTML5 aumenta mucho la funcionalidad que puede ejecutarse nativamente en un navegador web.

Desventajas:

- Generalmente las interfaces de usuario de las aplicaciones web son menos intuitivas y tienen un comportamiento peor que las clásicas.
- Las tecnologías web son muy dinámicas y cambiantes por lo que podemos usar alguna funcionalidad que desaparezca o se modifique drásticamente obligándonos a rehacer la interfaz (¿Flash en un futuro no muy lejano?)
- La ausencia de estándares en archivos “de oficina” puede dificultar el compartir datos e información.
- Dependen totalmente del correcto funcionamiento de la red (Internet y/o intranet).
- Desde el punto de vista de un usuario es preocupante la privacidad y seguridad de sus datos. Uno de los primeros ejemplos de esto es el correo web, pero en la actualidad todas las aplicaciones de Google y muchas otras como Facebook controlan absolutamente todo lo que haces por lo que la ausencia de privacidad es notable.

> La lista de ventajas y desventajas vista es un poco ambigua. Para cada caso piensa en lo que implica la afirmación e intenta buscar un ejemplo que suponga una excepción.

Como ya hemos comentado antes una de las principales preocupaciones a la hora de desarrollar una aplicación web debe ser la seguridad. En muchos casos se deben proteger tanto **información crítica de la empresa** como los **datos privados de los usuarios**.

A la hora de implementar la seguridad de una aplicación web debemos tener en cuenta cinco áreas:

- La **autentificación** de los usuarios: el uso de un método efectivo para asegurar que se conecten los usuarios autorizados y dificultar la suplantación de identidades es algo primordial. ¿Es una aplicación abierta a todo el mundo? ¿se pueden registrar los usuarios por si mismos?

- La **autorización** de cada usuario: muchas veces deben existir diferentes tipos de usuarios y no todos los usuarios deben poder acceder a todos los datos o realizar todas las operaciones (consulta, inserción, modificación o eliminación) sobre los datos. La identificación y creación de diferentes roles es un aspecto a tener en cuenta desde las primeras etapas de diseño de la aplicación.
- La **gestión de los recursos**: es necesario proteger los datos tanto cuando se encuentran en la base de datos como cuando se encuentran en tránsito entre la máquina cliente y el servidor. Generalmente se evitan las conexiones directas de forma remota (en una red abierta) a la base de datos. Además es muy importante la encriptación de los datos tanto en la base de datos como en las comunicaciones. Los datos se utilizan como ejemplo de recurso, pero es aplicable a otros tipos.
- La **entrada de datos**: otro aspecto a tener en cuenta es qué puede escribir el usuario en cada punto de entrada de datos. Cuanto más limitemos los caracteres y tipo de entrada que puedan realizar los usuarios más fácil será impedir que la gente acceda a puntos no deseados de nuestra aplicación e incluso a otras aplicaciones de la misma red.
- **Auditorías y registros**: Para poder controlar y corregir las brechas en la seguridad es muy importante poder saber qué ha pasado. Por ello debemos tener métodos de registro (por ejemplo archivos *log*) de todo lo que suceda en nuestra aplicación. El tamaño de los archivos debe permitirnos revisar situaciones de hace bastante tiempo ya que muchas veces un agujero en la seguridad tarda en detectarse.

Aunque no conocemos cómo afrontar cada problema en detalle, ya deberíamos tener conocimientos como para plantear soluciones para las áreas delicadas que acabamos de ver. Comenta un escenario que controle los problemas generados en cada área.

Para controlar todos estos problemas hay dos **recomendaciones** básicas:

- **Pruebas**: cuanto más probemos una aplicación menos fallos tendrá. Si la aplicación no es muy pequeña, es prácticamente imposible evitar todos los agujeros de seguridad pero hay que intentar minimizarlos. Lo ideal es tener una persona o un equipo

que sepan lo que hacen intentando atacar la aplicación. Este es el motivo por el que muchos hackers han terminado trabajando para compañías muy potentes.

- Usa un marco de trabajo (**Framework**): Si tenemos un equipo de trabajo no queremos que cada miembro "haga la guerra por su lado". Todos los integrantes deben usar uno o varios métodos comunes para gestionar la seguridad de la aplicación. Un marco común mejorará enormemente la seguridad de la aplicación. Un marco de trabajo para aplicaciones web debe tener en cuenta los siguientes aspectos: persistencia de datos, gestión de sesiones y autenticación de usuarios, seguridad, uso de cachés, uso de plantillas (con tipos de datos, etc.) e incluir una interfaz de administración.

Aunque escapa al contenido del curso, en ésta página puedes consultar estos aspectos sobre los frameworks de aplicaciones web y ver una lista con muchos de ellos.

Estructura

La estructura de una aplicación web es **equivalente a la de un servidor web** como vimos al principio del tema. La aplicación debe funcionar en todos los niveles por lo que tendremos una estructura en capas.

La más común es en tres capas (aunque la del medio puede estar subdividida como ya comentamos): presentación, aplicación y datos.

Otra forma de ver una aplicación web es en dos capas: el cliente y el servidor. Dependiendo de dónde se realiza la mayor parte del trabajo la aplicación será de cliente o de servidor pesado. En la actualidad, con la proliferación de dispositivos con poca capacidad y potencia (móviles, tabletas, netbooks, etc.) existen muchas aplicaciones pesadas en el servidor, pero cuando la aplicación requiere mucha lógica se tiende a distribuir el trabajo en los clientes para mejorar el rendimiento.

La estructura de una aplicación web también se usa para referirse a la **distribución en directorios** de todos los elementos que componen dicha aplicación. Aunque más adelante volveremos sobre este punto es importante enfatizar que la estructura de directorios que usemos debe estar contenida dentro de un punto de inicio y que no debe revelar aspectos sobre la distribución del resto de la máquina.

Descriptor de despliegue

El descriptor de despliegue de una aplicación web es un archivo de configuración en el contenedor o servidor de aplicaciones. En J2EE este archivo se escribe usando sintaxis XML. Describe cómo un componente, módulo o aplicación debe desplegarse especificando aspectos como la seguridad, las opciones del contenedor y aspectos de la configuración.

Existe además un descriptor de despliegue propio de cada servidor web. Por ejemplo en Tomcat, este otro descriptor se encuentra en <TOMCAT_HOME>/conf/web.xml. En el caso de la instalación vista anteriormente está en /opt/tomcat/apache-tomcat-7.0.29/ conf/web.xml.

Si estamos usando un entorno de desarrollo (por ejemplo Eclipse) el descriptor de despliegue lo crea el propio entorno, pero siempre es conveniente revisarlo para asegurarnos de que utiliza las opciones que queramos.

Para las aplicaciones web J2EE debe llamarse web.xml y ubicarse en el directorio WEB_INF en el directorio raíz de la aplicación. El esquema al que debe ajustarse el descriptor de despliegue se puede consultar en la página correspondiente de java.

Podemos ver un buen ejemplo de despliegue de aplicación web aquí. Ten en cuenta que si pinchas sobre una etiqueta te mostrará una breve explicación de su uso.

Se pueden consultar ejemplos más completos después de instalar Tomcat en las siguientes rutas:

- <TOMCAT_HOME>/webapps/jsp-examples/WEB-INF/web.xml
- <TOMCAT_HOME>/server/webapps/manager/WEB-INF/web.xml

Vamos a proyectar la página vista antes y a discutir sobre la estructura y diferentes elementos del descriptor de despliegue que nos ponen como ejemplo. Para ello determina previamente qué indica cada una de las etiquetas de segundo nivel (los hijos de <web-app>)