Tema 1 Implantación de arquitecturas web.

# 1.- Aspectos generales de arquitecturas web.

La arquitectura World Wide Web (WWW) **de Internet provee un modelo de programación sumamente poderoso y flexible**.

Las aplicaciones y los contenidos son localizados por los navegadores (web-browsers), que **envían requerimientos de objetos a un servidor y este responde con el dato codificado según un formato estándar**.

Los **estándares WWW** especifican muchos de los mecan**ismos necesarios para construir un ambiente de aplicación** de propósito general:

* **Modelo estándar de nombres**: todos los servidores se denominan según un Localizador Uniforme de Recursos (Uniform Resource Locator: URL).
* **Contenido**: a todos los contenidos en la WWW se les especifica un determinado tipo para que los browsers los interpreten correctamente.
* **Formatos de contenidos estándar**: los navegadores soportan un conjunto de formatos estándar, por ejemplo HTML, ECMA, JavaScript, etc.
* **Protocolos estándar**: permiten que cualquier navegador pueda comunicarse con cualquier servidor web. El más comúnmente usado es HTML (Protocolo de Transporte de HiperTexto), que opera sobre el conjunto de protocolos TCP/IP.

Esta infraestructura permite a los usuarios acceder a una gran cantidad de aplicaciones y servicios de terceros. **Los aspectos generales a destacar en una arquitectura web** son los siguientes:

* **Escalabilidad**.
* **Separación de responsabilidades**.
* **Portabilidad**.
* Utilización de componentes en los servicios de infraestructura.
* Gestión de las **sesiones del usuario**.
* Aplicaciones de **patrones de diseño**.

El **esquema de funcionamiento de los servicios web** requiere de 3 elementos fundamentales:

1. **Proveedor del servicio web**: es quien lo diseña, desarrolla e implementa y lo pone disponible para su uso.
2. **Consumidor del servicio**: es quien accede al componente para utilizar los servicios que presta.
3. **Agente del servicio**: sirve como enlace entre proveedor y consumidor para efectos de publicación, búsqueda y localización del servicio.

La **arquitectura web** es un **modelo compuesto de 3 capas**:

1. **Capa de Base de Datos**: donde está toda la información que se pretende administrar mediante el servicio web. Emplearía una plataforma como MySQL, PostgreSQL, etc.
2. **Servidores de aplicaciones web**: ejecutando aplicaciones como Apache, Tomcat, Resin, etc.
3. **Los clientes del servicio web**: Mozilla Firefox, Google Chrome, Opera, Edge…

## 1.1.- Evolución de los servicios web.

La **evolución del uso de Servicios web** en las organizaciones está **fuertemente ligada** al desarrollo de Internet. Factores que han impulsado el uso de los servicios web:

* **El contenido se está volviendo más dinámico**: proporcionan contenidos “instantáneos”.
* **El ancho de banda es menos costoso**: actualmente un servicio web puede entregar tipos variables de contenidos como vídeo o audio.
* **El almacenamiento es más barato**: debe ser capaz de manejar cantidades masivas de datos.
* **El éxito de la computación extendida**: Con cientos de millones de dispositivos como teléfonos móviles, el dispositivo más usado actualmente está dejando de ser el ordenador.
  + A medida que las plataformas se hacen más diversas, tecnologías como XML se volverán más importantes.
  + Un servicio web debe servir a todo tipo de dispositivos, plataformas y navegadores.

**Estos factores, unidos a los beneficios** proporcionados por los servicios web en la organización **y los buenos productos** disponibles para su desarrollo, **han hecho que su utilización se extienda** sin mayores obstáculos.

**Cuando se empiezan a usar servicios web** en una organización, estos **se desarrollan e implementan como servicios simples, que poco a poco** se van integrando hasta **llegar a servicios web mucho más complejos**.

En los orígenes del mundo web nos situábamos ante un entorno estático.

**La Web 2.0 es la transición** que se ha dado **desde las aplicaciones tradicionales hacia aplicaciones** que funcionan a través de la **web** y que están fuertemente enfocadas al usuario final. **Hace uso de nuevas tecnologías que tienen como objetivo**:

* Transformar software de escritorio hacia la web.
* Separar hojas de estilo.
* Potenciar el trabajo colaborativo.
* Dar control total a los usuarios en el manejo de su información.

## 1.2.- Tecnologías asociadas a las aplicaciones web.

Las aplicaciones web emplean páginas dinámicas. Estas se ejecutan en un servidor web y se muestran en el navegador. Cuando una página web llega al navegador, es posible que también incluya algún programa o fragmento de código que se deba ejecutar (en JavaScript) en el navegador.

Nos centraremos en las tecnologías asociadas a las aplicaciones web que se ejecutarán tanto del lado del servidor como del cliente:

|  |  |
| --- | --- |
| **Tecnología** | **Características** |
| **ASP (Active Server Pages)** | * Se ejecutan **del lado del servidor**, y así se forman los resultados que luego se muestran en el navegador del cliente que realizó la solicitud. * **Ejemplo**: los buscadores. * Existen versiones de ASP para Unix y Linux, a pesar de que fue una tecnología desarrollada por Microsoft ofrecida junto a su servidor IIS. * Es una tecnología obsoleta, al igual que CGI. * API clásico era un lenguaje de scripting embebido en lenguaje de marcas, lo que lo hacía muy parecido a PHP. * Además, estaba orientado a la utilización de componentes ActiveX. El lenguaje de scripting utilizado era VBScript y no era orientado a objetos. * Tampoco se daba soporte a los actuales paradigmas de desarrollo web, como MVC. * En la actualidad, **es una plataforma de desarrollo residual**. |
| **ASP .NET Webforms** | * Se introducen en el 2002 como **nuevo paradigma de desarrollo web**. * Se hace posible separar de forma nativa las capas de presentación, aplicación y acceso a datos. * Utiliza cualquiera de los lenguajes de la plataforma (a la cabeza de los cuales, se sitúa C#). * Webforms, además, introduce un ciclo de vida de la aplicación basado en eventos y controles de servidor. |
| **ASP .NET MVC y ASP.NET Core** | * ASP.NET evoluciona hacia el paradigma MVC. Además, ASP.NET se convierte en un **stack de código abierto (no propietario)** mantenido por una extensa comunidad de empresas, investigadores y entusiastas. * **Con ASP.NET MVC** y su **adhesión al código abierto**, llega el **desarrollo guiado por pruebas**, el análisis estático, los framework de pruebas automatizadas, ejecución paralela en el servidor y ejecución paralela no-concurrente, un modelo unificado para desarrollar servicios HTTP y aplicaciones web, inversión de control mediante inyección de dependencias, aplicaciones web en tiempo real, middleware, etc. |
| * En la actualidad, ASP.NET mantiene dos **branches** en desarrollo activo y **de código abierto**:   + **ASP.NET MVC** (versión 4.7), ligado todavía a sistemas Windows y al servidor IIS   + **ASP.NET Core (más nuevo)**: es un stack totalmente reescrito desde cero y orientado a su ejecución **en todo tipo de servidores** (IIS, Apache o nginx) y S.O (Linux, MacOS, Windows…). * Ambas tecnologías constituyen una excelente opción de desarrollo y, desde un punto de vista formal y metodológico, **son pioneras y muy superiores a otras tecnologías más extendidas**. * **En ambos casos, se dispone de**:   + **.NET permite el desarrollo de todo tipo de aplicaciones**, en consola, en ventana, para dispositivos móviles y también de aplicaciones web y servicios.   + **Variedad de lenguajes** (C#, F#...). Se puede programar en cualquiera de ellos (incluso combinándolos) con idéntico rendimiento. La utilización de uno u otro lenguaje solo difiere en la sintaxis específica, ya que **las APIs son compartidas**.   + **Soporte nativo a los actuales paradigmas de programación**, sin necesidad de frameworks.   + **Todos los lenguajes son orientados a objetos y de tipado fuerte**, aunque soportan otros modos, como la tipificación dinámica y la tipificación implícita.   + **La plataforma y los lenguajes son restrictivos**: sólo es posible hacer las cosas de una manera, la correcta. |
| **CGI (Common Gateway Interfaces)** | * La “Interfaz Común de Entrada” es **uno de los estándares más antiguos en Internet** para trasladar información desde una página a un servidor web. * Este estándar **es utilizado para bases de datos**, motores de búsqueda, formularios, generadores de email automático, foros, comercio electrónico, rotadores y mapas de imágenes, juegos en línea, etc. * **Las rutinas de CGI son** habitualmente escritas **en lenguajes interpretados como Perl o por lenguajes compilados como C**. * Fue pionera en el desarrollo web, pero **a día de hoy, se considera obsoleta**. |
| **CSS** | * Cascading Style Sheets (Hojas de Estilo en Cascada) se usan para formatear las páginas web; se trata de separar el contenido de su presentación. |
| **Java** | * Es un lenguaje que puede **trabajar tanto en el lado del cliente, como en el lado del servidor**. Es muy eficiente y poderoso, y se caracteriza por:   + Una misma aplicación puede funcionar en diversos tipos de ordenadores y sistemas operativos.     - Permite el desarrollo de todo tipo de aplicaciones: en consola, en ventana, para dispositivos móviles y también de apps webs y servicios.   + Pueden ser **aplicaciones independientes** (que corren en una ventana propia) o “applets”, pequeños programas interactivos incrustados en una página web.   + En desarrollo web, el API JSP/Servlets, combinado con beans, **permite, de manera nativa, separar la aplicación en capas**.   + Es de **tipado fuerte y “orientado a objetos”**. Los “módulos” a partir de los que se construyen, se pueden transformar o ampliar fácilmente.   + Desarrollado por la empresa Sun Microsystems, posteriormente liberado bajo licencia GNU GPL.   + **No dispone de soporte nativo para algunos de los paradigmas actuales**. Para ello, es necesario recurrir a frameworks específicos (JavaServerFaces, Spring, Hibernate, NUnit, etc.).   + **Java ofrece desarrollos robustos, fiables, fáciles de diseñar, depurar y modificar**. **Sin embargo, la responsabilidad de un buen diseño recae sobre el equipo de desarrollo** ya que impone pocas restricciones formales. Por ejemplo:     - "Se traga" antipatrones como la definición de una enumeración en una interfaz.     - Sería posible diseñar una aplicación web íntegramente con JSP, al estilo PHP, sin hacer uso de Servlets o Beans. |
| Dependiendo del tipo de aplicación desarrollada, podemos tener:   * **Aplicaciones en ventana o en consola**, que se ejecutan en un único equipo (arquitectura centralizada o arquitectura cliente, dependiendo del tipo de aplicación). * **Aplicaciones web desarrolladas con el API JSP/Servlets**, que se despliegan y **ejecutan en el servidor únicamente**. De igualmente, los servicios HTTP (implementados con SOAP+XML o cualquier otra tecnología similar) se ejecutan en el servidor. * **Aunque a día de hoy no se usa debido a sus limitaciones**, Jvva permite crear **applets**, pequeñas apps embebidas en una página web, que se descargan ye ejecutan en el cliente, de manera similar a como lo haría un script de JavaScript. |
| **JavaScript** | * Se interpreta y se ejecuta en el cliente. * Sire para realizar tareas como mover imágenes por la pantalla, crear menús de navegación interactivos, utilizar algunos juegos… * Se prefiere frente a VBScript (de Microsoft), al ser más aceptado por muchos más navegadores. |
| **PHP** | * **Hypertext PreProessor**: es ejecutado en el lado del servidor. Es muy similar a ASP y puede ser usado en circunstancias similares. * Es muy eficiente, permitiendo el acceso a bases de datos. * Suele utilizarse para crear páginas dinámicas complejas. * Basado en código embebido en lenguaje de marcas. * Desde un punto de vista formal y metodológico, plantea desventajas:   + Es un lenguaje de scripting **embebido en lenguaje de marcas**, por lo que resulta **dificil (o imposible) separar la aplicación en capas**, delimitando claramente sus responsabilidades.   + **Tipificación débil**: dificulta la lectura del código y resulta más propenso a errores.   + **No es orientado a objetos** (aunque simula algunos copmortamientos de la orientación a objetos, de manera muy peculiar).   + **Es un lenguaje interpretado**, lo que afecta, por un lado, al rendimiento, y por otro, limita las posibilidades de depuración.   + **No ofrece soporte nativo para muchos de los paradigmas actuales**, como la arquitectura MVC, el desarrollo guiado por pruebas, el análisis estático o la implementación de determinados patrones.     - Para disponer de alguna de estas características es necesario recurrir a frameworks de terceros (Code igniter, Larabel, Django...) * No obstante, **pese a todas estas desventajas, es posiblemente la plataforma de desarrollo más extendida**, en parte por su arraigo histórico. |
| **VBScript** | * **Visual Basic Scripting**: VBScript es una buena herramienta para cualquier sitio destinado a ser mostrado exclusivamente en el navegador Internet Explorer. * Puede estar diseñado para su ejecución en el lado del cliente o en el del servidor. * Un código que se ejecuta en el lado del servidor no es visible en el lado del cliente. El cliente solo recibe los resultados de lo que se ejecuta en el servidor, pero no el código con el que se obtienen. |

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

### Anexo: Lenguajes de scripting en el cliente.

Respecto a los lenguajes de script, ni VBScript, ni JScript existen en la actualidad en desarrollos web. **JavaScript es la realidad que se ejecuta en el lado del cliente a día de hoy**.

Es conveniente no confundir Java con JavaScript.

|  |  |
| --- | --- |
| **Java (JSP/Servlets)** | **JavaScript** |
| Se ejecuta en el servidor. | lenguaje estructurado, sin orientación a objetos pura (la simula, también de forma muy peculiar) |
| lenguaje orientado a objetos | interpretado (no compilado) |
| de tipado fuerte | tipificación débil |
| compilado a máquina virtual | scripts se ejecutan en el cliente |

Existen algunas ramificaciones curiosas, como TypeScript, que es **un intento por dotar de cierta tipificación a JavaScript**.

Dada la disparidad en las formas de implementar JavaScript en los distintos navegadores y de la relativa complejidad para realizar determinadas implementaciones, se han popularizado **numerosas APIs que ofrecen la realización de tareas habituales de manera simplificada** (jQuery es un ejemplo).

## 1.3.- Tipos de aplicaciones web.

Cualquier proyecto que se quiera desarrollar en Internet **conlleva el desarrollo de una aplicación web**. Una **aplicación web** es una **plataforma orientada a automatizar los procesos de servicios que se quieren ofrecer a usuarios**.

En función de cómo se presenta la aplicación web junto con el contenido que se pretende mostrar, se ha establecido la siguiente clasificación:

* **Página web estática**: implementada en HTML.
* **Página web animada**: Se realizan con la tecnología FLASH.
  + **Página web Dinámica**: Existen muchos lenguajes de programación que son la base para la mayoría de páginas web dinámicas. **PHP y ASP**: permiten una perfecta estructuración del contenido.
    - Por una parte, crearíamos la estructura de las páginas web
    - Y por otra, almacenaríamos el contenido en determinados archivos.
    - A partir de ahí, crearíamos el **código de llamada**, que insertaría el contenido en la propia página web estructurada.
    - A partir de aquí, se desarrollan aplicaciones para poder gestionar el contenido a través de un panel de control.
* **Portal**: es un sitio web que en su página principal permite el acceso a múltiples secciones.
* **Tienda Virtual o comercio electrónico**: sitio web que publica los productos de una tienda en Internet.
* **Página web con “Gestor de contenidos”**: Se trata de un sitio web cuyo contenido se actualiza a través de un panel de gestión por parte del administrador del sitio.
  + Es muy fácil e intuitivo de usar.
  + En aquellas páginas que requieren de una actualización constante, se suele incorporar este panel de gestión para poder controlarse día a día.

**Reflexiona**: ¿En algún momento te has parado a pensar qué cantidad de aplicaciones web hay disponibles en Internet para sustituir a las que tienes pensado instalar en el equipo?

## 1.4.- Arquitecturas web. Modelos.

La **arquitectura de un sitio web** **comprende los sistemas de organización y estructuración** de los contenidos **junto con los sistemas de recuperación de información y navegación** que provea el sitio web.

Centraremos el estudio de los modelos de arquitectura web relacionados, **en función de cómo implementan cada una de las capas establecidas en una aplicación web**.

1. **Capa de presentación**: encargada de la navegabilidad, validación de los datos de entrada, formateo de los datos de salida, presentación de la web… **Es la capa que se presenta al usuario**.
2. **Capa de negocio**: es la que recibe las peticiones del usuario y desde donde se le envían las respuestas. En ella, **se verifica que las reglas establecidas se cumplen**.
3. **Capa de acceso a datos**: formada por determinados gestores de bases de datos que se encargan de **almacenar, estructurar y recuperar los datos solicitados**.

**La evolución experimentada** por los **medios informáticos** en los últimos años **ha convivido con** otra evolución paralela, **la evolución de la arquitectura de las aplicaciones web**.

El modelo arquitectónico de las aplicaciones de Internet ha sufrido dos grandes saltos, con algún paso intermedio. **Los distintos modelos de aplicación, se podrían clasificar del siguiente modo**:

* **Modelo 1**: en este caso, las apps web se diseñan en un modelo web **CGI**, basadas en la ejecución de profesos externos al servidor web, **cuya salida por pantalla era el HTML que el navegador recibía en respuesta a su petición**.
  + Presentación, negocio y acceso a datos **se confundían en un mismo script**.
* **Modelo 1.5**: Aplicado a la tecnología , se da con la aparición de las JSP y los servlets.
  + Las responsabilidades de **presentación** recaen en las **páginas JSP**.
  + Los **beans** (abreviatura científica del botánico Wilian Jackson Bean. Es un componente software que tiene la particularidad de ser reutilizable y así evitar la tediosa tarea de programar los distintos componentes uno a uno) incrustados son los responsables del **modelo de negocio y acceso a datos**.
* **Modelo 2**: con la incorporación del patrón Modelo Vista Controlador (MVC) en este tipo de aplicaciones, se aprecia la **incorporación de un elemento controlador de la navegación de la aplicación**.
  + **El modelo de negocio queda encapsulado en los Javabeans** que se incrustan en las páginas JSP.
* **Modelo 2X**: como respuesta a la necesidad de desarrollar apps multicanal, es decir, aplicaciones web que pueden ser atacadas desde distintos tipos de clientes remotos (PDA, móvil, o cualquier navegador HTMl estándar).
  + Para publicar la misma aplicación para distintos dispositivos, debemos **emplear plantillas XSL para transformar los datos XML**.

**Para saber más**: webs para resolver dudas sobre la programación en Java.

* [Curso Aprender programación Java desde cero (aprenderaprogramar.es)](https://www.aprenderaprogramar.es/index.php?option=com_content&view=category&id=68&Itemid=188)

## 1.5.- Plataformas web libres y propietarias.

**Una plataforma web** es el entorno de desarrollo de software empleado para diseñar y ejecutar un sitio web. **Consta de 4 componentes básicos**:

1. **El sistema operativo**: bajo el cual opera el equipo donde se hospedan las páginas web.
2. **El servidor web**: software que maneja las peticiones desde equipos remotos a través de Internet. En el caso de sitios dinámicos, el servidor web se encarga de pasar las solicitudes a otros programas que puedan gestionarlas adecuadamente.
3. **El gestor de bases de datos**: se encarga de almacenar sistemáticamente un conjunto de registros de datos relacionados para ser usados posteriormente.
4. **Un lenguaje de programación interpretado**: controla las aplicaciones de software que corren en el sitio web.

**Diferentes combinaciones** de los cuatro componentes señalados **dan lugar a numerosas plataformas web**, aunque, sin duda, hay dos que sobresalen por su popularidad y difusión: LAMP y WISA.

|  |  |
| --- | --- |
| **Plataforma** | **Componentes** |
| **LAMP** | * **Linux**: Sistema Operativo. * **Apache**: servidor web. * **MySQL**: gestor de bases de datos. * **PHP**: lenguaje interpretado PHP, aunque a veces puede ser sustituido por Perl o Python. |
| **WISA** | * **Windows**: Sistema Operativo. * **Internet Information Services (IIS)**: servidor web. * **SQL Server**: gestor de bases de datos. * **ASP o ASP.NET:** lenguaje de scripting para el lado del servidor. |

**Existen otras plataformas**, como por ejemplo:

* la configuración Windows-Apache-MySQL-PHP que se conoce como **WAMP**.
* Windows, con IIS, y con MySQL y PHP: **WIMP**.

**Existen muchas otras plataformas** que trabajan con distintos sistemas operativos (Unix, Mac, Solaris), servidores web (como Lighttpd y LiteSpeed), bases de datos (Postgre SQL) y lenguajes de programación.

## 1.6.- Escalabilidad.

Las aplicaciones web se ejecutan en un entorno donde **el nº de clientes que solicitan el servicio puede variar en gran medida en función del momento**. Por ello, hay una característica de esencial importancia, como es la **escalabilidad**.

Uno de los principales factores que puede afectar al rendimiento de las apps web es el nº de usuarios. **El éxito o el fracaso de un sitio web orientado al usuario común vendrá determinado**, entre otros aspectos, **por el dimensionamiento del sistema** sobre el que se instala y soporta el software que sustenta dicho sitio. En consecuencia, **uno de los requisitos fundamentales** de una aplicación web es que sea **completamente escalable**, **sin que** un aumento de los recursos dedicados a la misma **suponga modificación alguna en su comportamiento o capacidades**.

La **escalabilidad de un sistema web puede ser**:

|  |  |
| --- | --- |
| **Tipo de escalabilidad** | **Características** |
| **Verticalmente** | Consiste en hacer “**upgrades**” a cada nodo de la granja de ordenadores. |
| La separación lógica en capas **se implementa de tal forma que se permita una separación física de las mismas**. |
| **Interponiendo elementos conectores que actúen de middlewares**, es posible distribuir la aplicación de forma vertical (una máquina por cada capa del sistema). |
| **Horizontalmente** | Consiste en **aumentar el número de nodos** (ordenadores). |
| Se trata de clonar el sistema en otra máquina de características similares y balancear la carga de trabajo mediante un dispositivo externo. |
| El **balanceador de carga puede ser**:   * **Balanceador software**: habitualmente encontramos un servidor web Apache junto al módulo **mod­\_jk** (que redirige las peticiones http configuradas previamente entre las distintas máquinas de la granja).   + Examinan el paquete http e identifican la sesión del usuario, **guardando registro de cuál de las máquinas de la granja se está encargando de servir a dicha sesión**.   + Nos permite trabajar (de cara al diseño de la aplicación) **apoyándonos en el objeto sesión** del propio usuario y almacenando información relativa a la sesión del mismo. * **Balanceador hardware**: son dispositivos que, mediante algoritmos de reparto de carga (Round Robin, LRU (Least Recently Used[[1]](#footnote-1)), etc.), **redireccionan una petición http del usuario a la máquina que convenga según el algoritmo**.   + Son mucho más rápidos que los de softare, puesto que **se basan en conmutación de circuitos y no examinan ni interpretan el paquete http**.   + Al no examinar ni interpretar el paquete http, no garantiza el mantenimiento de la misma sesión de usuario en la misma maquina.   + **Condiciona seriamente el diseño**, dado que **fuerza a que la información relativa a la sesión del usuario sea almacenada**, bien en cookies o bien en base de datos por el implementador del mismo. * **Balanceador hardware http**: se trata de dispositivos hardware, pero que examinan el paquete http y mantiene la relación usuario-máquina servidora.   + Son más rápidos que los balanceadores software, pero un poco más lentos que los hardware.   + Es una de las soluciones más aceptadas en el mercado. |
| **Cluster** | Consiste en crear **agrupaciones de servidores**. |
| Con la aparición de los **servidores de aplicaciones en cluster**, se abre una nueva capacidad de escalabilidad que, dependiendo de cómo se aplique, podría clasificarse como vertical u horizontal. |
| Un cluster de aplicaciones **permite el despliegue de una aplicación web** corriente, **de forma que su carga de trabajo va a ser distribuida entre la granja de servidores** que forman el clúster. |
| El cluster, mediante el mecanismo de **replicación de sesión**, **garantiza**  **que sea cual sea la máquina que sirva la petición http, tendrá acceso a la sesión del usuario** (objeto HttpSession en java).  Sin embargo, esta replicación hace que se suelan presentar **problemas de rendimiento**. |

**Debes conocer**: Como realizar la instalación del paquete XAMPP en un equipo con Ubuntu:

**Existen paquetes software que incluyen en una única instalación una plataforma AMP completa**. Algunos ni siquiera es necesario instalarlos, e incluso disponen de versiones para distintos sistemas operativos como Linux, Windows o Mac. **Uno de los más conocidos es XAMPP**.

**En primer lugar, se procede a la búsqueda del paquete**, para ello en la página del buscador **Google** introducimos XAMPP y ya en el primer resultado obtenemos "apachefriends" a donde accedemos, una vez dentro seleccionamos el sistema Linux y vamos a la sección "downloads" en donde podemos descargar diversas versiones de XAMPP, nos quedaremos con la última versión.

**Nos descargamos el paquete “xampp-linux-x64-5.5.19-0-installer.run”**. Vamos al directorio de descarga con el comando “cd”. Por ejemplo, si ha sido en “Descargas”, sería: cd Descargas.

**Damos los permisos pertinentes para poder ejecutarlo con**: sudo chmod 755 xampp-linux-x64-5.5.19-0-installer.run

**Lo instalamos con**: sudo ./xampp-linux-x64-5.5.19-0-installer.run

**Y aparecerá un instalador al más puro estilo de Windows** (siguiente, siguiente, …). Durante la instalación, **se puede elegir la opción de instalar** “Core files o archivos del núcleo” y “Developer Files o archivos de desarrollo” (**yo aconsejo dejar los 2 marcados para instalar**) e información sobre “[BitNami](http://bitnami.com/stack/xampp" \t "_blank)” (módulos para WordPress.org, Joomla!, Drupal, phpBB, Moodle, …). **El paquete queda instalado en /opt/lampp**.

Al finalizar la instalación y **en la última ventana nos da la opción de lanzar Xampp** al finalizar y se debería de abrir directamente la página de bienvenida de Xampp. **Accedemos a nuestro servidor web a través de: http://localhost**

**Para activar todos los servicios ejecutamos en una terminal**: sudo /opt/lampp/lampp start

**Desinstalación:**  
Lo podemos desinstalar, **borrando el directorio /opt/lampp**directamente con:

sudo rm -rf /opt/lampp

**Comandos para utilizar Xampp:**

* **Iniciar xampp:**sudo /opt/lampp/lampp start
* **Reiniciar xampp:**sudo /opt/lampp/lampp restart
* **Detener xampp:**sudo /opt/lampp/lampp stop

**Panel de control de Xampp**

Pero para un usuario común en un entorno de escritorio grafico **han creado una interfaz grafica que nos permite Iniciar / Detener de una forma fácil los servicios más importantes (Apache, MySQL y ProFTPD) o uno por uno**. Es el llamado **Panel de Control de Xampp** y se abre con el siguiente comando:

sudo /opt/lampp/share/xampp-control-panel/xampp-control-panel

**Pero antes, es necesario instalar el siguiente paquete**: sudo apt-get install python-glade2

**Ahora queda poner en funcionamiento los servidores**, comprobamos la ruta en la que se ha instalado XAMPP, en este caso en una subcarpeta de la carpeta**/opt/**, ahora arrancamos el servicio que nos indica la página:

**lampp** start

y arrancarán los **servidores**.

Para comprobar el correcto funcionamiento de los servidores Apache, PHP, etc. **abriríamos una página en el navegador en la que escribimos la siguiente URL "http://localhost"** aparecerá la ventana de XAMPP y desde aquí ya podríamos configurar y acceder a diversas funcionalidades de los servidores.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

# 2.- Servidor web Apache.

Un servidor web es un programa que **se ejecuta de forma continua en un ordenador**, **manteniéndose a la espera de peticiones** por parte de un cliente, **para contestarlas de forma adecuada**.

Uno de los servidores web más populares del mercado y el más utilizado actualmente es **Apache**. **Características de su arquitectura**:

* **Estructurado en módulos**.
  + Cada módulo contiene un conjunto de funciones relativas a un aspecto concreto del servidor.
* El archivo binario **httpd** contiene un conjunto de módulos que han sido compilados.
* La funcionalidad de estos módulos **puede ser activada o desactivada al arrancar el servidor**.
* Los módulos de Apache **se pueden clasificar en tres categorías**:
  + **Módulos base**: se encargan de funciones básicas.
  + **Módulos multiproceso**: encargados de la unión de los puertos de la máquina, aceptando las peticiones y atendiéndolas.
  + **Módulos adicionales**: se encargan de añadir funcionalidad al servidor.

El servidor Apache se desarrolla dentro del proyecto HTTP Server (httpd). La **Licencia Apache** permite la distribución de derivados de código abierto y cerrado **a partir de su código fuente original**.

**Para saber más**: web manual de referencia, guía de usuario, tutoriales prácticos, etc. Sobre el servidor web Apache. [Documentación del Servidor HTTP Apache 2.0 - Servidor HTTP Apache](https://httpd.apache.org/docs/2.0/es/)

## 2.1.- Instalación y configuración.

Instalación de Apache

1. **Actualizar repositorios**: sudo apt-get update
2. **Instalar paquete Apache**: sudo apt install apache2
3. **Probamos el servidor**: entrando en la dirección <http://localhost> o <http://127.0.0.1>

**En el directorio /var/www/html podremos ver el fichero index.html** que nos muestra al acceder a nuestro servidor. Veremos cómo modificar cuál es el fichero que nos lanza por defecto

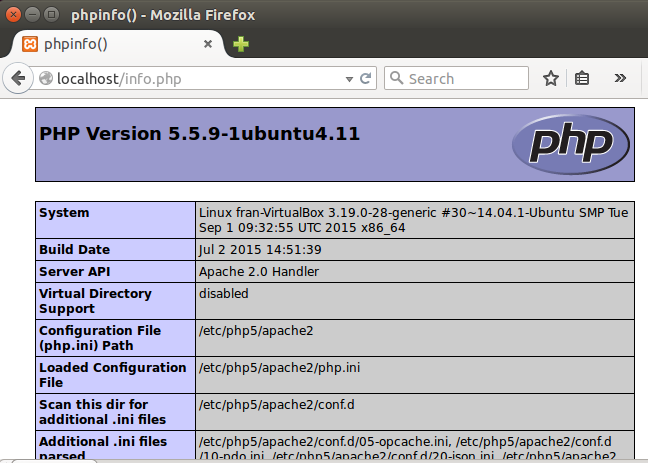
Debido a que pretendemos **montar una plataforma LAMP**, instalaremos también los siguientes componentes: MySQL y PHP.

Instalación de PHP

* **Instalación**: sudo apt install php5 libapache2-mod-php5
* **Reiniciamos** **el servicio** para que los cambios surtan efecto: sudo /etc/init.d/apache2 restart

|  |
| --- |
| “Notice” o advertencia de Apache |
| Podemos encontrarnos un “Notice” (advertencia) de Apache referente a la no existencia de la directiva Servername en el fichero de configuración. Para solucionarlo, añadimos “Servername localhost” en el fichero /etc/apache2/apache2.conf |

* **Probar PHP**: sudo nano /var/www/html/info.php
  + **Introducimos el siguiente código**: <?php phpinfo(); ?>
  + **Tecleamos en el navegador**: <http://localhost/info.php>

****

Instalación de MySQL

* **Instalación**: sudo apt install mysql-server mysql-client.
  + **Importante**: recordar la contraseña del root del gestor de bases de datos que pongamos durante la instalación.
* **Reiniciamos los servicios**: sudo /etc/init.d/apache2 restart

Configuración

Si la única pretensión es servir una página web, podemos integrar su contenido aquí. En caso de que se pretenda **servir más páginas web**, es más recomendable la utilización de los **hosts Virtuales**.

1. En la carpeta "**/etc/apache2/sites-enabled**", hay un fichero llamado "**000-default**":
   1. Nos va a servir **de ejemplo para la creación de hosts virtuales**, los cuales van a **permitir servir varias webs desde una sola dirección IP** **utilizando para cada una un puerto distinto**.

El **archivo principal de configuración** se llama **apache2.conf**.

* Se pueden añadir otros archivos de configuración mediante la directiva “Include”, y se pueden usar comodines para incluir muchos archivos de configuración. Todas las directivas deben colocarse en alguno de esos archivos de configuración.
* Apache solo reconocerá los cambios realizados cuando se inicie o reinicie.

**Configuraciones del archivo etc/apache2/apache2.conf**:

* Número de puerto.
* **DocumentRoot**: especifica la raíz de documentos de Apache.
* Módulos.
* Archivos de registro.
* Hosts virtuales…
* **ServerTokens**: para configurar la cantidad de información que Apache aporta sobre sí mismo.
* **ServerSignature**: para indicar datos sobre Apache en el pie de los mensajes de error.
* **Alias**: permite direccionar a una carpeta que pueda estar fuera del árbol de directorios especificado en DocumentRoot.
* **userDir**: permite redireccionar al directorio personal del usuario si se recibe una solicitud del tipo ~usuario.
* **RedirectMatchen**: las peticiones **se redirigirán a /var/www/apache2-default**, que es donde reside el sitio predeterminado de Apache2

**Configuraciones individuales para cada servidor virtual**:

**Para modificar el servidor virtual predeterminado**: fichero /etc/apache2/sites-available/default

En el caso de querer configurar un **nuevo servidor o sitio virtual**, **copiaríamos ese archivo** dentro del mismo directorio con el nombre que se haya elegido, y **editaríamos el nuevo archivo para configurar el nuevo sitio usando algunas de las directivas** **que se describen a continuación**:

* **ServerName**: en el caso de no tener un dominio registrado, emplearíamos localhost.
* **CustomLog**: define el archivo .log donde se guardan los logs de acceso.
* **ServerAdmin**: especifica la dirección de correo del administrador del servidor. El valor por defecto es **webmaster@localhost**.
* **Listen**: especifica el puerto (y, opcionalmente, la dirección IP) por el que escuchará Apache2.
  + Se puede cambiar en **su propio archivo de configuración**: /etc/apache2/ports.conf
* **DocumentRoot**: especifica dónde Apache debe buscar los archivos que forman el sitio. El valor predeterminado es /var/www.
* **RedirectMatchen**: en el archivo de host virtual: cambiar este valor en este archivo, implica crear ese directorio si fuese necesario.

**Debes conocer**: cómo instalar la herramienta de gestión web de un sistema, Webmin, en Ubuntu.

Se parte de un S.O Ubuntu con la herramienta Webmin previamente instalada. **Webmin es una herramienta de configuración de sistemas accesible vía web** para OpenSolaris, GNU/Linux y otros sistemas Unix. **Con él se pueden configurar aspectos internos de muchos sistemas operativos**, como usuarios, cuotas de espacio, servicios, archivos de configuración, apagado del equipo, etcétera, así como modificar **y controlar muchas aplicaciones libres, como el servidor web Apache, PHP, MySQL**, entre otros.

En primer lugar, se **busca el paquete de Apache en el buscador de la herramienta Webmin**, se selecciona el paquete correspondiente y, a continuación, la opción de instalar.

Se **comprueba que el servidor web Apache ha sido correctamente instalado**, posteriormente se **comprueba que es capaz de mostrar una página web** que nosotros deseemos que el Apache sirva. **Para ello debemos copiar la página a la carpeta /var/www**, **estableciendo**, en primer lugar, **los permisos correspondientes para trabajar con dicha carpeta**.

**Una vez copiada la página a la carpeta anterior, se para el servicio correspondiente a Apache y se vuelve a arrancar**, se comprueba el funcionamiento correcto del servidor introduciendo en el navegador la URL: http://127.0.0.1

Otra parte del vídeo demuestra cómo crear un nuevo Virtual Host y que escuche por el puerto 81, para ello es necesario **configurar el archivo /etc/apache2/ports.conf** para indicar al servidor que escuche por dicho puerto.

## 2.2.- Iniciar Apache.

Si hemos instalado Apache en la ruta **/usr/local/apache**, podemos **probar su configuración** por defecto e intentar iniciar el servicio de la siguiente forma:

**/usr/local/apache/bin/apachectl configtest**

**Si todo está correcto, debería devolver el mensaje “Syntax Ok”**.

**Para iniciar el servicio**, hacemos: **/usr/local/apache/bin/apachectl start**.

Si en un navegador introducimos la URL <http://localhost>, veríamos la página de bienvenida a Apache.

**Si el puerto especificado** en la directiva Listen del fichero de configuración es el que viene **por defecto (puerto 80), o cualquier otro puerto por debajo del 1024** (puertos privilegiados), entonces **es necesario tener privilegios de usuario root para iniciar Apache**.

**Una vez Apache se ha iniciado**, lanzará varios procesos, procesos hijo, que hacen el trabajo de escuchar y atender las peticiones de los clientes. **El proceso principal, httpd**, **continúa ejecutándose como root**, pero los procesos hijo se ejecutan con menores privilegios de usuario.

Como hemos visto antes, **se puede invocar el demonio httpd empleando el script** de control **apachectl**, que es el que se encarga de fijar variables de entorno, y le pasa al httpd cualquier opción que se le pase como arugmento por línea de comandos.

El **script apachectl** es **capaz de interpretar los argumentos** **start**, **restart** y **stop**, **y traducirlos** en señales apropiadas **para** el demonio **httpd**.

**Para parar, reiniciar o arrancar el servidor**:

* # /etc/init.d/apache2 stop
* # /etc/init.d/apache2 restart
* # /etc/init.d/apache2 start

**Reflexiona**: ¿Será necesario reiniciar el servicio Apache si, mediante la creación de un host virtual, hemos cambiado el puerto por el que escucha? Definitivamente, sí.

# 3.- Aplicaciones web y servidores de aplicaciones.

**Aplicación web**: es una aplicación informática que se ejecuta en un entorno web, de forma que se trata de una aplicación cliente-servidor junto con un protocolo de comunicación previamente establecido:

* Cliente: navegador.
* Servidor: servidor web.
* Comunicación: protocolo HTTP.

Un **servidor de aplicaciones**:

* Es un software que **proporciona aplicaciones a los equipos cliente**, por lo general, a través de Internet y utilizando el protocolo HTTP.
* **Hacen uso extensivo del contenido dinámico y se integran con las bases de datos**.
* También es una **máquina en una red de computadores** que ejecuta determinadas aplicaciones.

Las **principales ventajas** de la tecnología de los servidores de aplicaciones:

* La **centralización** y **disminución** **de la complejidad** **en el desarrollo de las aplicaciones**, ya que **no necesitan ser programadas**, sino que son ensambladas desde bloques provistos por el servidor de aplicación.
* **Integridad de datos y código**, ya que, al estar centralizada en una o un pequeño nº de máquinas servidoras, **las actualizaciones están garantizadas para todos los usuarios**.

El **término** servidor de aplicaciones **se aplica a todas las plataformas**. Se refiere a los servidores de aplicaciones basadas en web.

**Ejemplos de plataformas para servidores de aplicaciones**: la plataforma J2EE (Plataforma JavaTM 2 Enterprise Edition). Incluye:

* **Un nivel Cliente**: este servidor puede contener una o más aplicaciones o navegadores.
* **Un nivel medio**: la plataforma J2EE es de este nivel, y consiste en un servidor web y un servidor EJB (Enterprise JavaBeans). **Estos servidores son también llamados contenedores**.
  + También podría haber subniveles adicionales en el nivel intermedio.
* **Un EIS** (Enterprise Information System, o “Sistema de Información Empresarial”): contiene las aplicaciones existentes, archivos y bases de datos.

**Para saber más**: sobre las [mejores apps webs](https://hipertextual.com/2015/01/mejores-aplicaciones-web) y la [categorización de las apps webs](https://es.wikipedia.org/wiki/Categor%C3%ADa:Aplicaciones_web).

## 3.1.- El servidor de aplicaciones Tomcat.

**Tomcat** es servidor web (incluye el servidor Apache) y servidor de aplicaciones del proyecto Yakarta, con lo cual, gestiona las solicitudes y respuestas http y, además, es servidor de aplicaciones o contenedor de Servlets y JSP.

**Incluye el compilador Jasper**, que compila JSP convirtiéndolas en Servlets.

Tomcat es un contenedor de servlets con un entorno JSP. Un **contenedor de servlets** es un **Shell de ejecución** que maneja e invoca servlets por cuenta del usuario.

**Hay varios tipos de contenedores de servlets**:

|  |  |
| --- | --- |
| **Tipo de contenedor de servlets** | **Características** |
| **Stand-alone** (independientes) | Estos son **una parte integral del servidor web**. Este es el caso en el que se usa un **servidor web basado en Java**. Por ejemplo, el contenedor de servlets es parte de JavaWebServer (actualmente sustituido por iPlanet). |
| **Por defecto, Tomcat trabaja en este modo**, sin embargo, la mayoría de los servidores no están basados en Java. |
| **Dentro-de-proceso** | El contenedor servlets es una **combinación de un plugin para el servidor web y una implementación de contenedor Java**. |
| **El plugin** del servidor web **abre una JVM** (Java Virtual Machine) dentro del espacio de direcciones del servidor web **y permite que el contenedor Java se ejecute en él**. |
| **En** el **caso** **de que una petición debiera ejecutar un servlet**, **el plugin** toma el control sobre la petición y **lo pasa al contenedor Java** (usando JNI, Java Native Interface). |
| **Fuera-de-proceso** | El contenedor servlets es una combinación de un plugin para el servidor web y una implementación de contenedor Java que se ejecuta en una JVM **fuera del servidor web**. |
| El plugin del servidor web y el JVM del contenedor Java se comunican usando algún mecanismo IPC (Interprocess Comunication) (normalmente sockets TCP/IP) |
| **El tiempo de respuesta en este tipo de contenedores no es tan bueno** como el anterior, **pero es mejor en otras cosas: escalabilidad, estabilidad, etc.** |

**Tomcat puede utilizarse**:

* como un **contenedor solitario** (para desarrollo y depuración)
* o **como plugin para un servidor web existente** (actualmente soporta los servidores Apache y IIS).

Sie**mpre que despleguemos Tomcat, tendremos que decidir cómo usarlo**, si seleccionamos las opciones 2 o 3, también necesitaremos instalar un adaptador de servidor web.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente con confianza media

### 3.1.1.- Instalación y configuración básica.

**Para instalar cualquier versión de Tomcat es necesario tener instalado**.

El **objetivo** es que las **peticiones** a Apache **se redirijan a Tomcat empleando un conector** proporcionado por **Java** en este caso.

1. **Buscamos el paquete de Java** que nos pueda interesar. Opciones
   1. sudo apt-get install openjdk-7-jdk
   2. sudo aptitude search “?provides(java-runtime)”
   3. Aunque, instalando el paquete **default-jre**, sería suficiente para trabajar con Tomcat:
      1. sudo apt-get install default-jre
2. **Establecemos las variables de Java en el entorno**, editamos el fichero environment. Dos formas de hacerlo:
   1. **sudo gedit /etc/environment**

**JAVA\_HOME=/usr/lib/jvm/java-7-openjdk-amd64**

**JRE\_HOME=/usr/lib/jvm/java-7-openjdk-amd64/jre**

PATH="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:**$JAVA\_HOME:$JRE\_HOME"**

* 1. **Editar /etc/profile** y añadir (puede variar la ruta):

JAVA\_HOME=/usr/lib/jvm/java-6-openjdk/jre/

PATH=$PATH:$JAVA\_HOME/bin

export PATH JAVA\_HOME

source /etc/profile -> **Para actualizar las variables de entorno creadas**

1. **Instalamos Tomcat**. Como siempre, hay varias formas de hacerlo:
   1. sudo apt-get install tomcat7
   2. Descargar desde la URL <http://tomcat.apache.org>
   3. Descargar desde URL mediante wget en la consola:

wget [http://apache.rediiri.es/tomcat/tomcat- 6/v60.32/bin/apache-tomcat-6.0.32.tar.gz](http://apache.rediiri.es/tomcat/tomcat-%206/v60.32/bin/apache-tomcat-6.0.32.tar.gz)

Y descomprimir archivo en carpeta deseada:

mv apache-tomcat-6.0.32.tar.gz /usr/local -> para moverlo a la carpeta deseada.

tar xvzf apache-tomcat-6.0.32.tar.gz -> para descomprimirlo.

1. **Colocamos las variables de entorno** y el fichero environment quedará como sigue:

JAVA\_HOME=/usr/lib/jvm/java-7-openjdk-amd64

JRE\_HOME=/usr/lib/jvm/java-7-openjdk-amd64/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"

**Como podemos apreciar**:

* 1. **CATALINA\_HOME**: indica el directorio de instalación de Tomcat.
  2. **CATALINA\_BASE**: indica el directorio de una instancia de Tomcat. Si tenemos más de una instancia, CATALINA\_BASE será diferente para cada una.

**Para saber más**: [OpenJDK](https://openjdk.org/)

### 3.1.2.- Usuarios Tomcat.

**Cómo realizar las operaciones básicas de inicio, parada y reinicio de Tomcat**:

sudo /etc/init.d/tomcat7 start

sudo /etc/init.d/tomcat7 stop

sudo /etc/init.d/tomcat7 restart

**Cuando realicemos cualquier cambio** en la configuración de Tomcat, **tendremos que reiniciar el servicio para que surta efecto dicho cambio**.

Al abrir Tomcat y abrir el navegador **tenemos una ventana desde la que podemos acceder**, siempre que los paquetes correspondientes estén instalados, **a documentación, ejemplos y administración**.

* **Paquete de documentación tomcat7-docs**: este paquete instala una aplicación web que permite navegar por la documentación de Tomcat7 localmente.
* **Paquete de ejemplos tomcat7-examples**: este paquete instala una aplicación web que permite acceder a ejemplos de Servlets y páginas JSP de Tomcat.
* **Paquete de administración tomcat7-admin**: este paquete instala dos aplicaciones web que ayudan a administrar el servicio Tomcat. Una vez instalado, puedes obtener acceso al manager webapp y al host-manager webapp.

Para acceder a esa gestión, **tenemos que tener un usuario y contraseña definidos**, pues no viene por defecto. Para editar el fichero de usuarios de Tomcat:

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

Y **definimos nuestro usuario y contraseña**. **Al final, quedará el fichero algo parecido a esto**:

<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="fran" password="tomcat" roles="manager-gui,admin-gui,manager,admin,manager-**

**script,admin-script"/>**

</tomcat-users>

**Reiniciamos Tomcat para que los cambios** surtan efecto:

### 3.1.3.- Apache como proxy.

* **Tomcat va a estar escuchando en el puerto 8080** y va a tener **su propio directorio de trabajo**.
* **Apache2** **va a interceptar todas las peticiones en el puerto 80 y derivar las que considere necesarias a Tomcat**; de este modo, observamos la ventaja de escalabilidad, ya que Apache, **al funcionar como proxy, puede tener una batería de tomcats a los que balancear las conexiones**.

Configuración del directorio de trabajo

* **Apache por defecto busca los ficheros en /var/www**.
* Tomcat trabaja sobre la carpeta **/usr/local/tomcat/websapps/ROOT**.

**La petición de una URL se puede gestionar por parte de Apache y por parte de Tomcat. Por ello, será necesario cambiar la carpeta por defecto de trabajo para unificarlo**.

Para ello, **editamos el fichero** /usr/local/tomcat/conf/**server.xml**

sudo nano /etc/tomcat7/server.xml

En donde encontraremos una línea con “**Host name=**” y lo establecemos a:

<host name=”localhost” appBase=”/var/www”

**Cargaremos los módulos siguientes** para poder conseguir que Apache funcione como proxy:

# a2enmod proxy

# a2enmod proxy\_ajp

# a2enmod proxy\_balancer

# /etc/init.d/apache2 restart

**Ajp** (Apache JServ Protocol) es un protocolo de comunicación interno y muy rápido que usa conexiones TCP persistentes. **Lo usaremos para comunicar Apache2 con Tocmat**, aunque podría ser utilizado http, indicando que pregunte en el 8080.

**El puerto de trabajo por defecto para Tomcat es el 8009**, aunque este puede ser variado desde:

/usr/local/tomcat/conf/server.xml

**Modificamos el fichero de configuración del virtualhost** que se pretendía utilizar, empleando el establecido por defecto.

sudo nano /etc/apache2/sites-enabled/000-default

**En donde añadimos lo siguiente**:

<**Proxy balancer**://tomcat\_cluster>

Order allow,deny

Allow from all

**BalancerMember** ajp://localhost:8009

</Proxy>

**ProxyPreserveHost On**

ProxyPass /phpmyadmin/ !

**ProxyPass / balancer**://tomcat\_cluster/

ProxyPassReverse / balancer://tomcat\_cluster/

**¿Qué es cada cosa?:**

* **Proxy balancer://tomcat\_cluster**: estamos definiendo un cluster con nombre “Tomcat\_cluster”.
* **BalancerMember ajp://localhost:8009**: se define un miembro a Tomcat\_cluster, protocolo, IP y puerto.
* **ProxyPass / balancer://tomcat\_cluster/**: "/" y todo lo que cuelgue de ella, será pasado al cluster del tomcat para que lo procese él.
* **ProxyPreserveHost On**: mantiene la cabecera http host original, en vez de reescribirla.

**Lo último es cambiar un par de cosas del fichero de configuración de Apache**: /etc/apache2/sites-enabled/000-default

* el “**DocumentRoot**” y “**<Directory /var/www**” para que apunten a /var/www/ROOT.
* De esta manera, podemos decidir qué parte gestiona cada aplicación desde un solo directorio.

# 4.- Estructura y despliegue de una aplicación web.

**Una aplicación web está compuesta de una serie de servlets, páginas jsp, ficheros html**, etc.; de forma que **todos estos recursos se pueden empaquetar** **y ejecutar en varios contenedores distintos**.

Un **servlets** es una **aplicación java encargada de realizar un servicio específico** dentro de un servidor web.

La **especificación Servlet 2.2** **define la estructura de directorios para los ficheros de una aplicación web**.

El **directorio raíz** debería tener el **nombre de la aplicación** y define la raíz de documentos para la aplicación web. **Todos los ficheros debajo de esta raíz son servidos, excepto los que están bajo los directorios especiales**:

* **META-INF**.
* **WEB-INF**. Todos los ficheros privados, al igual que los ficheros class de los servlets, deberían almacenarse bajo este directorio.

**Durante la etapa de desarrollo** de una aplicación web, **se emplea la estructura de directorios**. Luego, **en la etapa de producción**, toda la estructura de la aplicación **se empaqueta en un archivo .war**.

El **código necesario** para ejecutar correctamente una aplicación web se encuentra distribuido en una estructura de directorios agrupando ficheros según su funcionalidad.

Un **ejemplo de la estructura de carpetas de una aplicación web**:

**Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

Descripción generada automáticamente**

De forma genérica, podemos decir que **una app web se estructura en tres capas**:

* **Navegador** web.
* **Tecnología web dinámica** (PHP, Java Servlets, ASP, etc.).
* **Base de datos** encargada de almacenar de forma permanente y actualizada la información que la aplicación web necesita.

## 4.1.- Archivos WAR.

Su nombre procede de Web Application Archive (Archivo de Aplicación Web): permiten **empaquetar en una sola unidad aplicaciones web de Java completas**, **conteniendo**:

* Servlets y JSP.
* Contenido estático: HTML, imágenes, etc.
* Otros recursos web.

**Su estructura es la siguiente**:

* **/**: en la carpeta raíz del proyecto se almacenan elementos empleados en los sitios web: documentos HTML, CSS, y los elementos JSP (\*.html \*.jsp \*.css).
* **/WEB-INF/**: aquí están los elementos de configuración del archivo .WAR: la página de inicio, la ubicación de los servlets, parámetros adicionales para otros componentes…
  + **El más importante de estos es el** **web.xml**.
* **/WEB-INF/classes**: contiene las clases Java empleadas en el archivo .WAR. Normalmente, **en esta carpeta se encuentran los servlets**.
* **/WEB-INF/lib**: contiene los **archivos JAR** utilizados por la aplicación.
  + Normalmente son las clases empleadas **para conectarse con la BB.DD** **o las empleadas por librerías de JSP**.

**Para generar archivos .WAR**: se pueden emplear diversas herramientas desde entornos IDE:

* **NetBeans y Eclipse**: ambos open-source.
* **JBuilder**: de Borland.
* **JDeveloper**: de Oracle.
* **Otro modo de construir archivos WAR**: es mediante Ant.
  + Se trata de una herramienta Open-Source que facilita la construcción de aplicaciones Java.
  + No es considerado un IDE, pero para los que conocen el entorno Linux**, es considerado el** **make de Java**.

Tabla

Descripción generada automáticamente

## 4.2.- Despliegue de aplicaciones con Tomcat.

**Una aplicación web puede ser desplegada empleando uno de los siguientes métodos**:

* Por medio de archivos WAR (Web Archive).
* **Editando los archivos web.xml y server.xml**. Este método es el que se describe a continuación.

Los **directorios** que forman una aplicación compilada suelen ser: www, bin, src, tomcat y gwt- cache.

* La **carpeta www** contiene, una carpeta con el nombre y ruta del proyecto que contiene los ficheros que forman la interfaz (html, js, css, etc.).
* **La carpeta bin** contiene las clases de Java de la aplicación.

**Para desplegar la aplicación en Tomcat**:

1. **Copiar la carpeta contenida en** **www** (con el nombre del proyecto) **en** el directorio **webapps** de Tomcat.
2. **Renombrar la nueva carpeta** así creada en Tomcat con un nombre más sencillo. Esa **será la carpeta de la aplicación en Tomcat**.
3. **Crear**, dentro de dicha carpeta, otra nueva, y darle el nombre **WEB-INF** (respetando las mayúsculas).
4. **Crear, dentro de WEB-INF**, otros dos subdirectorios, llamados **lib** y **classes**.
5. **Copiar en lib todas las librerías** (.jar) que necesite la aplicación para su funcionamiento.
6. **Copiar** el **contenido** **de** la carpeta **bin** de la aplicación **en** el subdirectorio **WEB-INF/classes** de Tomcat.
7. **Crear en WEB-INF** un fichero de texto llamado **web.xml**, **con las rutas de los servlets** utilizados en la aplicación.
8. **A la aplicación ya puede accederse en el servidor**, poniendo en el navegador la ruta del fichero html de entrada.

**Para saber más**: sobre los pasos implicados en el despliegue de un servlet, creación de una app web, tanto en formato expandido, como en un WAR. Ilustra cómo desplegar una app web en Apache Tomcat y en WebLogic Server 6.0, un completo servidor de aplicaciones J2EE. [Desplegar Servlets y Aplicaciones Web en Tomcat y WebLogic Server. Programación en Castellano. (programacion.net)](https://programacion.net/articulo/desplegar_servlets_y_aplicaciones_web_en_tomcat_y_weblogic_server_175)

## 4.3.- Descriptor de despliegue.

Un **descriptor de despliegue** **es un documento XML que describe las características de despliegue** de una aplicación, módulo o componente. Por esto, **la información del descriptor de despliegue es declarativa**, y puede ser cambiada sin modificar el código fuente.

**Cualquier aplicación web tiene que aportar un descriptor de despliegue**, **situado en WEB-INF/web.xml**; el descriptor <TOMCAT\_HOME>/conf/web.xml es un descriptor por defecto que se ejecuta siempre antes del descriptor de la aplicación.

Un **ejemplo de descriptor de despliegue** puede ser el siguiente archivo **web.xml**:

<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE web-app PUBLIC

"-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN"

"http://java.sun.com/j2ee/dtds/web-app\_2\_2.dtd">

<web-app>

<!-- Tus definiciones van aquí -->

</web-app>

**Entre las etiquetas <web-app> y </web-app> estarían los descriptores de despliegue de servlets**, los cuales deben tener las siguientes etiquetas en el siguiente orden:

<servlet>

<servlet-name>nombre</servlet-name>

<servlet-class>package.nombre.MiClass</servlet-class>

</servlet>

**Para probar el servlet**, una vez arrancado el servidor Tomcat, **abrimos un navegador web**, en el cual escribimos una URL con el siguiente formato:

http://{address}:{port}/{servletName}

Por ejemplo: <http://localhost:8080/Servlet_de_prueba>

**Caso práctico**:

<?xml version="1.0" encoding="UTF-8"?>

<web-app

version=

"3.0"

xmlns=

"http://java.sun.com/xml/ns/javaee"

xmlns:xsi=

"http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation=

"http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_3\_0.xsd"

>

<servlet>

<servlet-name>

HolaMundoServlet

</servlet-name>

<servlet-class>

org.prueba.servlet.HolaMundoServlet

</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>

HolaMundoServlet

</servlet-name>

<url-pattern>

/HolaMundoServlet

</url-pattern>

</servlet-mapping>

<session-config>

<session-timeout>

30

</session-timeout>

</session-config>

</web-app>

1. Least Recently Used: menos usado recientemente. [↑](#footnote-ref-1)