

Configuración y administración de servidores de aplicaciones.

Caso práctico



En la empresa **BK programación**, **Ada**, junto con sus empleados **Juan** y **María** se ha reunido para evaluar la posibilidad de configurar uno o dos servidores de aplicaciones para instalar en ellos demos, o **versiones beta**, de las aplicaciones que desarrollan, de esta manera los clientes, o potenciales clientes, podrían probar los productos de **BK programación** antes de adquirirlos.

Como resultado de dicha reunión han concluido que, previo paso a la instalación y puesta en funcionamiento de los servidores de aplicaciones, sería muy importante evaluar el software a instalar y los parámetros que afectarían al correcto funcionamiento de los servidores. Entre las cosas a evaluar caben destacar las siguientes:

- ✓ Servidores de aplicaciones disponibles en el mercado y características específicas de los mismos (Tomcat, WildFly, etc.).
- ✓ Necesidades de instalación, configuración y mantenimiento del servidor de aplicaciones elegido.
- ✓ Proceso de despliegue de una aplicación web o empresarial en el servidor y herramientas se habría que utilizar.
- ✓ Flexibilidad para la administración remota, lo cual implica la configuración de la conexión remota a los mismos.
- ✓ Herramientas que pueden utilizarse para la automatización de tareas en el servidor (como por ejemplo Ant).
- ✓ Medidas de seguridad a implementar en el servidor de aplicaciones para evitar posibles ataques o intrusiones.

Debido a la cantidad de ítems que hay que considerar para poner en funcionamiento un servidor de aplicaciones, **Ada** ha decidido que sus empleados se documenten apropiadamente. Además, está evaluando la posibilidad de que estos realicen algún curso de formación sobre la administración de servidores de aplicaciones.

1.- Introducción a las aplicaciones Web y servidores de aplicaciones.

Caso práctico

Frente al encargo de **Ada**, **Juan** se ha puesto manos a la obra a investigar. Está muy motivado con este nuevo encargo, y es porque todo esto de las aplicaciones web y los servidores de aplicaciones web le suena, pero no termina de tener claro los conceptos.

Por eso, en la wiki que está construyendo, **Juan** ha decidido dedicar dos apartados a dos conceptos de vital importancia:

- ✓ **Aplicaciones web** y
- ✓ **Servidor de Aplicaciones**.

Juan sabe que ambos conceptos están estrechamente relacionados.

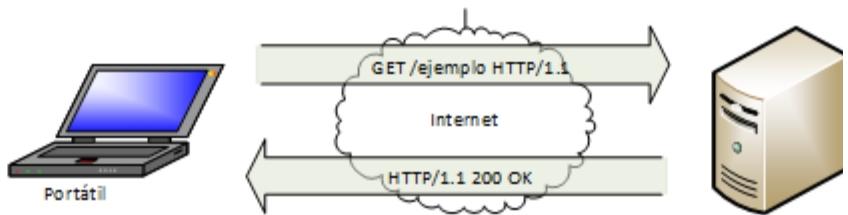


Y tú, ¿cómo definirías una aplicación web? **Una aplicación web es una aplicación informática que se ejecuta en un entorno web.** Se trata de una aplicación cliente-servidor. Esto quiere decir que parte de la aplicación se ejecuta en un servidor, otra parte de la aplicación se ejecuta en el cliente, y la comunicación entre el cliente y el servidor se realiza a través de Internet usando un protocolo de comunicaciones ya conocido.

Una aplicación web puede considerarse como una extensión de un servidor de aplicaciones o de un servidor web.

Analicemos las partes que intervienen en una aplicación web, son conceptos que seguro que te suenan de la unidad anterior:

- ✓ **Cliente:** corresponde con el **navegador web** que se ejecuta en el ordenador de un usuario (Firefox, Chrome o Edge, son ejemplos de navegadores web). La función del navegador web es mostrar las páginas web que recibe del servidor web. Recuerda que una página web está generalmente formada por HTML, CSS, imágenes, etc., y opcionalmente por código que se ejecuta en el lado del cliente para hacer la página dinámica (escrito normalmente en JavaScript).



- ✓ **Protocolo de comunicaciones:** es el conjunto de normas de comunicación que deben seguir tanto el cliente (navegador web), como servidor (servidor web), para comunicarse. En el entorno web los protocolos utilizados son HTTP y HTTPS.
- ✓ **Servidor:** sería un sistema cuya ubicación es, a priori, desconocida por el usuario. En el servidor se ejecuta un software conocido como **servidor web** (por ejemplo, Apache), cuya función es dar soporte a los protocolos de comunicaciones HTTP y HTTPS. En ese servidor web se ejecuta otra parte de nuestra aplicación web, que se encargará de generar la página web (HTML, CSS, imágenes, e incluso JavaScript) que el cliente web debe mostrar.

Revisemos el proceso de ejecución de la aplicación web en el entorno descrito anteriormente:

- ✓ Después de que el usuario ponga la URL en la barra de direcciones (o haga clic en un enlace), el navegador web hace una petición HTTP a nuestro servidor web.
- ✓ El servidor web recibe la petición y en consecuencia, si nuestro servidor web está apropiadamente configurado, ejecuta el programa que hemos creado (por ejemplo, en PHP). Este programa generará todo el contenido de nuestra página web (HTML, CSS, etc.), y se lo entregará al servidor web. El servidor web después le hará llegar al cliente el contenido generado por nuestra aplicación web.
- ✓ Después, el cliente web recibe el contenido y lo muestra al usuario, ejecutando, si fuese necesario el código JavaScript recibido del servidor.

En el modelo de ejecución anterior, cada vez que el cliente realiza una petición el servidor web inicia la ejecución de un programa en nuestro servidor. Significa que si nuestra aplicación web recibe 20 peticiones cada segundo, nuestro programa se ejecutará 20 veces. Esto puede asustar, sin embargo, entornos como PHP, que funcionan de esta forma, están profundamente optimizados para mejorar el rendimiento.

Un servidor web al que se le ha añadido soporte para scripting (como PHP) gestiona el ciclo de ejecución de una aplicación web de la siguiente forma: inicio de ejecución, procesado de una única petición y finalización de la ejecución.

Pero todavía falta un concepto por aclarar, ¿qué es un **servidor de aplicaciones**? Para entender que es un servidor de aplicaciones debemos pensar que el proceso anterior puede ser diferente. Veamos como podría ser:

- ✓ Ejecutamos en nuestro servidor nuestra aplicación web de "forma permanente". Esta aplicación web estará siempre cargada en memoria a la espera de que se le pida que genere el contenido web.
- ✓ Después, el cliente envía la petición HTTP a nuestro servidor web.
- ✓ El servidor web entonces, apropiadamente configurado, redirige la petición HTTP a nuestra aplicación web, que ya está cargada en memoria de antes, lista y atenta para responder a lo que se le pida.
- ✓ Después, nuestra aplicación web genera una respuesta con el contenido y se lo devuelve al servidor web, el cuál, se lo hará llegar al navegador web. Nuestra aplicación web se queda a la espera de la siguiente petición.
- ✓ El navegador web entonces recibe los datos, los muestra y ejecuta el código del lado del cliente.

Este esquema tiene sus ventajas y sus inconvenientes. Entre sus ventajas podemos decir que, generalmente, las aplicaciones web son más eficientes al seguir este modelo (aunque muchos expertos y expertas cuestionarían este aspecto). Entre sus inconveniente podemos destacar los siguientes:

- ✓ Las aplicaciones web se tienen que diseñar de una forma diferente, ateniéndose a una serie de especificaciones que permiten su ejecución, dado que en la práctica se convierten en un componente que extiende las capacidades del servidor web y que participan del proceso de generar la respuesta al cliente web.
- ✓ El software de servidor web se transforma en algo diferente, dado que ahora es necesario mantener en memoria una aplicación, ejecutarla de forma permanente, y reenviar las peticiones HTTP para que se genere una respuesta. Este tipo de servidores se les conoce, por tanto, como **servidores de aplicaciones**.

Un servidor de aplicaciones gestiona el ciclo de ejecución de una aplicación web de la siguiente forma: inicio de la ejecución, procesado de una o más peticiones HTTP y finalización de la ejecución.

Sin embargo, aunque hoy día la mayor parte de los servidores de aplicaciones están destinados a comunicar aplicaciones del lado del servidor, con aplicaciones del lado del cliente a través de HTTP, su perspectiva es algo más amplia. Podríamos generalizarlo de la siguiente forma:

Un servidor de aplicaciones es un software que ejecuta aplicaciones en un equipo remoto con el objetivo de dar servicio a otras aplicaciones cliente situadas en el equipo del usuario. Los servidores de aplicaciones, además, proporcionan un conjunto de servicios que facilitan el desarrollo de la lógica de negocio.

Para saber más

En la siguiente web se detallan 20 de las aplicaciones web mas influyentes de los últimos tiempos y el propósito de cada una de ellas.

 [20 Most Influential Open-Source Web Applications](#)

Autoevaluación

¿Cuál de las siguientes opciones son **falsas**?

- La ejecución de la aplicación web se realiza en un servidor de aplicaciones o un servidor web con soporte de scripting.
- La aplicación web se puede considerar como una extensión del cliente web.
- Un servidor de aplicaciones es el gestionar la ejecución de la aplicación web, cargandola en memoria y reenviandole peticiones del cliente.
- El navegador web se comunicaría con el servidor, ya sea servidor web o de aplicaciones, a través de los protocolos HTTP o HTTPS.

[Mostrar retroalimentación](#)

2.- Introducción a las aplicaciones Web basadas en Java EE.

Caso práctico

Juan empieza a entender que un servidor de aplicaciones es diferente a un servidor web, pero no termina de ver que aporta un servidor aplicaciones frente a un servidor web. **María** le pregunta como lleva su misión:

- ¿Cómo lo llevas **Juan** con la wiki? Esto de los servidores de aplicaciones es para mí un mundo nuevo y apasionante - comenta **María**.

- ¿Sí? - le responde **Juan** - Pues yo no termino de verlo. No termino de ver que tiene de especial un servidor de aplicaciones. Solo veo un montón de terminología compleja, por ejemplo, ¿qué demonios es un servlet?



[Robert. Un servidor de aplicaciones en funcionamiento. \(CC BY\)](#)

Y tú, ¿sabrías decir que particularidades tiene un servidor de aplicaciones? En el apartado anterior se ha visto una perspectiva inicial y aquí vamos a intentar situarnos un poco mejor dentro de la filosofía de un servidor de aplicaciones, encajándola en el desarrollo de aplicaciones web.

Java EE es posiblemente la plataforma de servidor de aplicaciones más extendida hoy día, pero es solo un conjunto de especificaciones. No obstante, existen bastantes implementaciones de dicha especificación: WildFly (de Red Hat), TomcatEE (bajo el sombrero de Apache), WebSphere Application Server (de IBM), GlassFish y WebLogic Server (de Oracle); son los ejemplos más significativos.

Java EE es una arquitectura que **permite la implementación de aplicaciones que dan servicio a otras aplicaciones cliente**.

Para lograr lo anterior, Java EE proporciona un conjunto de servicios de sistema que permite que el equipo de programación no tenga que hacer programación de bajo nivel, ni código repetitivo, y que pueda centrarse en solucionar la lógica de la aplicación. Cuando hablamos de la lógica de la aplicación, podemos separar entre:

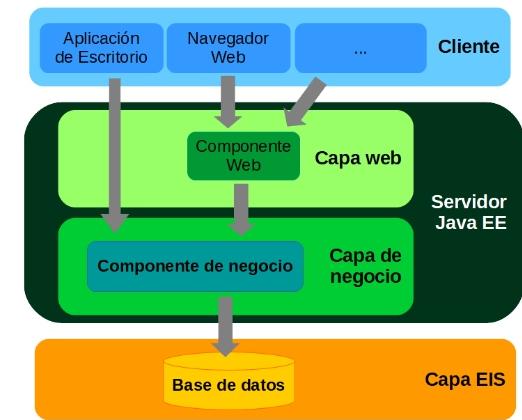
- ✓ **Lógica de negocio.** Corresponde con el cerebro de la aplicación, donde se llevan a cabo la mayor parte de los procesamientos y operaciones internas. Algunos ejemplos podrían ser: obtener la lista de clientes, dar de alta a un cliente o obtener la lista de pedidos de un cliente.
- ✓ **Lógica de presentación.** Corresponde con lo que ve el usuario, es decir, la interfaz de usuario. Los datos resultantes de la lógica de negocio son representados por la lógica de presentación de forma que el usuario puede entenderlos, proporcionando además una interfaz para comunicarse con la aplicación.

Y seguro que ahora te estarás preguntando lo siguiente, ¿cómo es posible separar la lógica de negocio de la lógica de presentación? Para ello las aplicaciones Java EE siguen un modelo basado en componentes, multicapa y distribuido:

- ✓ **Basado** en componentes por que una aplicación Java EE se compone de unidades de software llamadas componentes, que agrupan una o varias funciones.
- ✓ **Distribuido** por que cada componente de la aplicación se puede ejecutar en un sistema diferente, donde los componentes pueden comunicarse entre sí a través de la red.
- ✓ **Y multicapa** por que cada tipo de componente se aloja en una capa diferente, donde los componentes de una capa pueden usar los componentes de las capas inferiores para realizar su función.

Las capas de una aplicación Java EE son las siguientes:

- ✓ **Máquina cliente:** donde encontramos componentes como el navegador web, applets Java y aplicaciones de escritorio; aunque estos no entran dentro de la plataforma Java EE.
- ✓ **Capa Web:** donde encontramos **componentes web** encargados de producir contenido Web dinámico (HTML, CSS, etc.), aunque también puede generar otro tipo de contenido (XML o JSON por ejemplo). Los componentes de esta capa si se ejecutarían en el servidor Java EE, en lo que denominamos un **contenedor Web**. Aparte de estos componentes web, en esta capa también deberíamos ubicar el contenido estático de la aplicación web (imágenes, páginas HTML puras, etc.) y otro código que se ejecutaría en el navegador (JavaScript, por ejemplo).
- ✓ **Capa de negocios:** donde encontramos componentes encargados de llevar a cabo la lógica de negocio de nuestra aplicación y que también se ejecutarían en el servidor Java EE. Estos componentes son los **Enterprise Java Beans (EJB)** se ejecutarían en un **contenedor EJB**.
- ✓ **Capa del sistema de información de la empresa** (Enterprise Information System, EIS): es un término que de forma muy general hace referencia a los sistemas donde se almacena la información de la empresa, que aunque pueden ser de lo más variopinto, lo normal es que sean un sistema gestor de bases de datos. **Esta capa no está dentro de la plataforma Java EE.**



Elaboración propia (Salvador Romero Villegas). *Modelo en capas de los servidores Java EE.* (Dominio público)

Vale, ya va tomando forma todo esto, pero, ¿cómo se comunican estos componentes entre sí? Veamos algunos ejemplos:

- ✓ Un navegador web, situado en la máquina cliente, se comunicará solamente con la capa web del servidor Java EE, para obtener información en un formato que luego pueda mostrar al usuario final.
- ✓ Una aplicación de escritorio podría comunicarse con la capa web (obteniendo información por ejemplo en formato XML si hemos implementado un  Web Service), o directamente con la capa de negocio.
- ✓ La capa web podría comunicarse con la capa de negocio para pedirle a esa capa que realice alguna tarea.
- ✓ La capa de negocio podría comunicarse con la capa EIS para pedirle que almacene o actualice algunos datos de una base de datos.

En el modelo de aplicación Java EE **la capa web estaría destinada a implementar la lógica de presentación, y la capa de negocios estaría destinada a implementar procesos de la lógica de negocio**, aunque en algunas ocasiones, cuando la aplicación es de poca complejidad, en la capa web también se resuelve la lógica de negocio.

En el modelo de aplicación de Java EE **la capa web y la capa de negocio forman lo que denominamos capa intermedia (middletier)**, que es dónde situamos nuestras aplicaciones web.

Como veremos más adelante, los componentes siguen una estructura estándar y se empaquetan de forma que pueden ser usados en cualquier servidor de aplicaciones que cumpla con la certificación Java EE, en un proceso conocido como **despliegue**. Una vez desplegada una aplicación, la aplicación se puede empezar a ejecutar.

Debes conocer

Hay que tener en cuenta que cuando hablamos, de forma general, de un servidor de aplicaciones, puede entenderse este como la máquina donde se ejecuta una determinada aplicación que ofrece servicio a aplicaciones cliente, y no necesariamente al software de servidor de aplicaciones que usamos para servir las aplicaciones (como podría ser WildFly o WebLogic).

También hay que tener en cuenta que Java EE no es la única plataforma que sirve de servidor de aplicaciones, existen otras con una estructura interna diferente:

 [Lista de servidores de aplicaciones \(en inglés\).](#)

Por último, ten en mente que no existe una definición precisa de "servidor de aplicaciones", por lo que hay quien considera un servidor  LAMP como un servidor de aplicaciones. Aquí nos hemos centrado en la idea de servidor de aplicaciones de Java EE, que es la más extendida.

2.1.- Servlets y JSP.

Caso práctico

Ada se interesa por como llevan **Juan** y **María** la tarea encomendada. **Juan** se muestra un poco más tranquilo que al principio, lo cual tranquiliza también a **Ada**.

- **Juan**, como van esos servidores de aplicaciones - comenta **Ada** de forma chistosa.

- Más o menos bien, ya voy pillando el concepto, ahora estoy investigando sobre los componentes web - comenta **Juan** con un gesto de preocupación.



Ada nota que **Juan** está haciendo avances y que simplemente es cuestión de tiempo. Así que le da ánimo y lo deja para que siga con su tarea.

Todavía no hemos resuelto la duda que tenía Juan, ¿qué es un servlet? ¿tú lo sabes? Un servlet es un componente de la capa web, es decir, un tipo de componente web. Entre los componentes que podemos ejecutar en la capa web tenemos: **Servlets**, **JavaServer Pages** y **JavaServer Faces**. De estos, nos vamos a centrar en los siguientes:

- ✓ **Servlets**: son clases en Java que reciben peticiones y ofrecen respuestas de forma dinámica. Normalmente las peticiones y las respuestas son usando el protocolo HTTP, y como resultado producen contenido web, pero no tiene por qué ser necesariamente así.
- ✓ **JavaServer Pages (JSP)**: es una tecnología destinada a generar contenido web dinámico (aunque también podría generar otro tipo de documentos), embebiendo trocitos de código Java que modifican el contenido generado; la idea es parecida a lo que hace PHP o ASP.NET, pero con Java. Los archivos JSP realmente se compilan y se transforman en un servlet.

No vamos a abordar otras tecnologías, ni vamos a profundizar demasiado en estas, porque el objetivo de esta unidad es aprender a desenvolverse con un servidor de aplicaciones, y no crear aplicaciones web usando estas tecnologías. Pero para entender como funciona un servidor de aplicaciones, sí es conveniente entender como se produce el proceso de petición y respuesta, y como se llega a ejecutar un componente web.

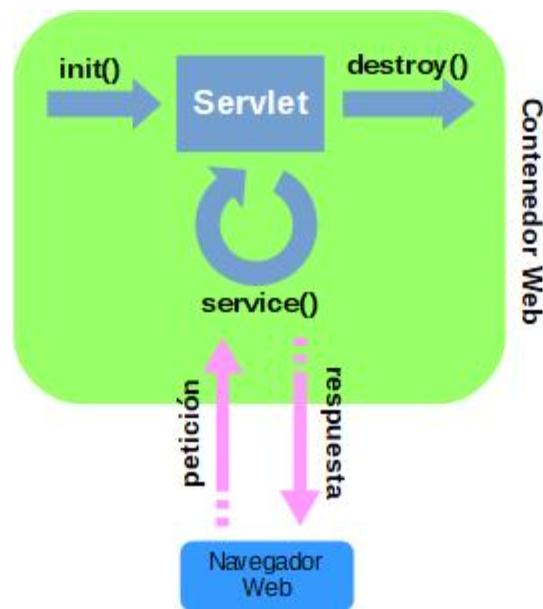
Una aplicación web está formada por uno o más componentes web (Servlets, JSP, JSF, etc.), **y también, por otros recursos estáticos opcionales**: HTML, imágenes, CSS, JavaScript, sonidos, etc.

Imaginemos que tenemos una aplicación web que ya está ejecutándose en un servidor local (localhost, por ejemplo). Y que la URL de nuestro servicio es la siguiente: <http://localhost/ejemplo/hoyes>. El objetivo de nuestra aplicación web es simple: darnos información sobre que día es hoy. Cuando un cliente web solicita un recurso web, ocurriría, más o menos, lo siguiente:

- ✓ El usuario teclea la URL <http://localhost/ejemplo/hoyes> en el navegador web o cliente web.
- ✓ **El navegador web lanza una petición HTTP al servidor web.** Se trata de un servidor web que tiene soporte para componentes web basados en servlets, es decir un contenedor web.
- ✓ **El servidor web convierte la petición en un objeto de tipo HttpServletRequest que pasará al componente web adecuado** (puede haber más de un componente web). Como puede haber varios componentes web, la aplicación web debe estar apropiadamente configurada para poder enviar la petición al componente web adecuado.
- ✓ **El componente web**, que hemos programado, **procesará la petición y generará** una respuesta en forma de un objeto tipo `HttpServletResponse`.
- ✓ **El contenedor web transforma ese objeto en una respuesta HTTP** al cliente.

La función de un contenedor web es:

- ✓ **Enviar la petición web al componente web adecuado** para que sea procesada. Para esto es necesario que la aplicación web esté **configurada** de forma adecuada.
- ✓ **Servir el contenido estático** (páginas HTML, imágenes, código JavaScript, etc.).
- ✓ **Permitir la integración de la aplicación web en diferentes contextos.** Por ejemplo, en el contexto de desarrollo se suele utilizar una base de datos de prueba, para no dañar los datos originales, pero al cambiar al contexto de producción, la base de datos a utilizar es otra. Es una función de administración el establecer la fuente de datos usada en producción.
- ✓ **Ejecución segura** de la aplicación web.
- ✓ **Concurrencia.** Un componente web debe poder atender múltiples peticiones simultáneamente.
- ✓ **Gestión del ciclo de vida del componente web.**
- ✓ **Dar acceso a la aplicación web a un conjunto de servicios** para realizar funciones diversas: acceder a la base de datos (JDBC), acceso a servicios de directorio como LDAP (a través de JNDI), procesar archivo XML (a través de JAXP), etc. Esos servicios pueden ser usados por el equipo de desarrollo en el momento de desarrollar la aplicación web.



Elaboración propia (Salvador Romero Villegas). *Ciclo de vida de un servlet.* (Dominio público)

Las aplicación web, aparte de programarse, deben configurarse. Las aplicaciones web se configuran a nivel de desarrollo, donde el equipo de desarrollo define el comportamiento interno de la aplicación, y a nivel de administración, donde se integra la aplicación ya desarrollada en un ambiente de ejecución concreto.

A nivel de desarrollo, la configuración permite al equipo de desarrollo describir como se comportará internamente la aplicación web. Esta configuración puede hacerse de diversas formas:

- ✓ A un nivel más local, **a través de anotaciones** directamente realizadas en el código del componente web (por ejemplo en las clase Java del servlet).
- ✓ A un nivel más global, **a través del descriptor de despliegue**. Se trata de un archivo **XML**, generalmente llamado web.xml, cuyo objetivo es precisamente describir como se tiene que hacer el despliegue de la aplicación web.
- ✓ Usando **descriptores de despliegue dependientes de la implementación**. Y es que, dependiendo del servidor Java **EE** concreto en el que vayamos a desplegar nuestra aplicación, dispondremos también de descriptores de despliegue específicos para esa plataforma (por ejemplo, Oracle WebLogic Server permite el uso de un archivo llamado weblogic.xml).

Configurar nuestras aplicaciones web a través de anotaciones es más sencillo, sin embargo, las opciones que podemos configurar a través de anotaciones son menos que las que podemos hacer a través del descriptor de despliegue. Por lo que se suele usar una estrategia mixta, parte se configura a través de anotaciones y otra parte a través del descriptor de despliegue.

Gracias al uso de anotaciones, el uso de un descriptor de despliegue se convierte en algo opcional en muchos casos.

A nivel de **administración**, la configuración está destinada a integrar una aplicación web en un servidor o ambiente concreto, información que llamaríamos **contexto**. En Tomcat, por ejemplo, el archivo destinado a esta parte de la configuración es context.xml.

Autoevaluación

Una aplicación web que se está ejecutando en un servidor de aplicaciones está usando una base de datos MySQL, y por una decisión estratégica, ahora se va a usar un servidor MariaDB, ¿qué podemos hacer?

- Configurar con anotaciones la base de datos a utilizar.
- Configurar en el descriptor de despliegue el servidor de bases de datos a utilizar.
- Configurar en el servidor de aplicaciones la base de datos que debe usar la aplicación web.
- Esto no se puede hacer, si la aplicación está diseñada para funcionar con un motor de base de datos concreto, no puede funcionar con otro.

2.2.- Estructura de una aplicación WEB.



Chris P. Jobling (CC BY-SA)

Una aplicación web Java EE puede desplegarse en un contenedor web usando el formato expandido o empaquetado como WAR.

Los archivos situados en estas carpetas no son servidos directamente al cliente como los recursos situados directamente en el directorio raíz, dado que su propósito es otro.

En la carpeta WEB-INF podríamos encontrar lo siguiente:

- ✓ **Descriptor de despliegue.** En el ejemplo anterior el descriptor de despliegue podríamos encontrarlo en la ruta ejemplo/WEB-INF/web.xml.
- ✓ **Clases necesarias para nuestra aplicación web (archivos .class).** Esto incluye a todo el conjunto de clases que componen nuestra aplicación web y a los servlets (que son una clase más). Estos archivos se sitúan dentro de la carpeta WEB-INF/classes. Por ejemplo:
 - ◆ ejemplo/WEB-INF/classes/agenda/Agenda.class
 - ◆ ejemplo/WEB-INF/classes/servlets/ExportarLista.class
- ✓ **Librerías.** Se trata de archivos JAR que contienen librerías que pueden ser utilizadas por nuestra aplicación. Ejemplos de estas librerías podrían ser drivers JDBC para conectar a bases de datos o librerías para generar archivos PDF. Estas librerías se sitúan en la carpeta WEB-INF/lib/, por ejemplo:
 - ◆ ejemplo/WEB-INF/lib/libreria.jar
- ✓ Otros archivos y recursos internos de nuestra aplicación.

Todos los archivos privados, al igual que los ficheros .class de los servlets, deberían almacenarse dentro del directorio WEB-INF.

En la carpeta META-INF podemos encontrar lo siguiente:

- ✓ Archivos de configuración adicionales, dependientes del servidor Java EE usado. Por ejemplo, Tomcat permite incluir aquí el archivo context.xml, el cuál permite al equipo de administración configurar como se integra la aplicación web en un ambiente concreto.
- ✓ MANIFEST.MF: archivo típico de los paquetes JAR que contiene información sobre el paquete en si mismo, y que también se utiliza cuando se empaqueta una aplicación web en un archivo WAR.

Veamos un ejemplo completo de una estructura de directorios de una aplicación web:

src	agenda
	agenda.jsp
	META-INF
	MANIFEST.MF
	WEB-INF
	classes
	agenda
	agenda/Agenda.class
	agenda/Clases.class
	agenda/servlets
	agenda/servlets/ExportarLista.class
	WEB-INF

Durante la etapa de desarrollo de una aplicación web se emplea esta estructura de directorios (directorio expandido o desempaquetado). Luego, en la etapa de producción, la aplicación se suele empaquetar en un archivo .war.

Para saber más

En el siguiente enlace encontrarás el ejemplo anterior, que contiene un ejemplo muy simple de aplicación de gestión de citas:

 [Aplicación web de ejemplo \(citas\)](#) (zip - 10.89 KB) .

Y en el siguiente enlace puedes descargar el proyecto NetBeans usado para generar la aplicación web anterior:

 [Proyecto NetBeans de la aplicación web de ejemplo \(citas\)](#) (zip - 24.53 KB) .

Ten en cuenta que para poder abrir el proyecto anterior es necesario que tengas la edición de NetBeans completa o la que incluye soporte para Java EE.

Autoevaluación

Si necesitaras usar una plantilla de correo electrónico para tu aplicación web, de tal manera que la aplicación web usara dicha plantilla para el envío de correos, ¿en qué directorio la pondrías?

- En la carpeta WEB-INF.
- En la carpeta raíz.
- En la carpeta META-INF.
- Fuera de la aplicación web.

2.3.- Empaquetado WAR de aplicaciones web.

Caso práctico

En la empresa **BK programación** continúan planificando la puesta en marcha de uno o varios servidores de aplicaciones. **Ada** considera necesario formar al personal acerca de cómo desplegar aplicaciones, especificar la estructura a seguir, etc. Un punto importante a detallar es la distribución de aplicaciones web mediante los archivos WAR, y **Juan** comienza a detallarlo en su wiki.

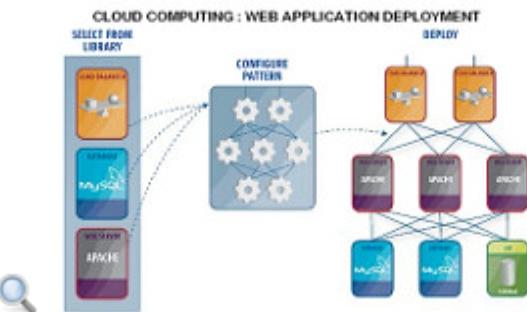


¿Sabes qué significan las siglas WAR? Su nombre procede de Web Application Archive (Archivo de Aplicación Web); y la idea de los archivos WAR es permitir empaquetar en una sola unidad aplicaciones web Java EE completas. Como se explicó en el apartado anterior, una aplicación web basada en Java EE puede desplegarse en formato de directorio expandido (o desempaquetado), pero también como un archivo WAR.

Los archivos WAR son en realidad archivos JAR, pero cuyo contenido es una aplicación web. Generalmente los archivo JAR están destinados a contener una aplicación o una librería Java, pero en este caso, su objetivo es almacenar una aplicación web, y por eso tienen la extensión .war.

El uso de archivos WAR tiene muchas ventajas sobre el formato de directorio expandido. La primera ventaja directa es la simplificación del proceso de despliegue, dado que todo va incluido en un único archivo. Dentro de dicho archivo WAR podremos tener todo lo que nuestra aplicación web requiera: servlets, archivos JSP, contenido estático (HTML, imágenes, archivos CSS, etc.), y otros recursos.

Para generar un archivo .war se parte de la estructura de directorios de una aplicación en formato expandido, estudiada en el apartado anterior, la cual se empaqueta con el comando jar. A dicho comando se le pasaría el nombre del archivo WAR deseado (generalmente con extensión .war) y la ruta completa a la aplicación web en formato expandido:



[louisvolant CC BY-NC-SA](#)

Autoevaluación

Dada la definición de archivo .war<code lang="en">, ¿cuál de las siguientes opciones son correctas?

- Es un archivo comprimido que se puede generar con cualquier tipo de compresor, por ejemplo winzip, winrar, tar, etc.
- Es una aplicación web formada únicamente por archivos .html
- Es un archivo en el cual se empaqueta en una sola unidad, aplicaciones web completas.
- Es un archivo que engloba el protocolo de comunicación de las aplicaciones web generadas con Java.

[Mostrar retroalimentación](#)

2.4.- El descriptor de despliegue.

¿Qué es exactamente un descriptor de despliegue? Un **descriptor de despliegue** es un documento XML que describe las características de despliegue de una aplicación, un módulo o un componente. Por esto, la información del descriptor de despliegue es declarativa, y ésta puede ser cambiada sin la necesidad de modificar el código fuente, aunque hay que tener en cuenta que hoy en día, parte de la configuración que se indica en el descriptor de despliegue se suele poner como anotaciones dentro del mismo código fuente.

Hoy en día, no es obligatorio que una aplicación web aporte un descriptor de despliegue, pero si se incluye este tiene que estar en la carpeta WEB-INF y debe llamarse web.xml (**WEB-INF/web.xml**). Dependiendo de la plataforma donde vayamos a desplegar nuestra aplicación web, podremos tener descriptores de despliegue adicionales con información específica para la plataforma.

Generalmente todos los contenedores web incluyen un descriptor de despliegue por defecto, que contiene una configuración aplicable a todas las aplicaciones web bajo su mando. Por ejemplo, en el caso concreto de Tomcat, existe un archivo web.xml en la carpeta de configuración (conf/web.xml) con la configuración general para todas las aplicaciones web. Este descriptor por defecto generalmente no se modifica nunca, dado que cada aplicación web puede disponer de su propio descriptor para complementar a este descriptor por defecto.



Elaboración Propia (Salvador Romero Villegas). *Función del descriptor de despliegue.* (Dominio público)

El descriptor de despliegue por defecto del contenedor web no debe incluir una configuración específica para una aplicación concreta.

Un ejemplo de descriptor de despliegue de una aplicación web concreta puede ser el siguiente archivo web.xml:



Con la configuración anterior, si la aplicación web es accesible a través de la URL `http://localhost:8080/ejemplo/` y el usuario la teclea directamente en la barra de direcciones, el servidor de aplicaciones enviará al cliente web la página `http://localhost:8080/ejemplo/index.jsp` aunque no se indique `index.jsp` en la URL.

El objetivo de este apartado es que tengas una visión global de lo que es un descriptor de despliegue. No profundizaremos más en este aspecto, porque el objetivo de esta unidad no es hacer aplicaciones web, sino saber desplegar una aplicación web ya realizada.

Para saber más

En el siguiente enlace podrás encontrar información adicional sobre como crear y que incluir en un descriptor de despliegue estándar:

 [Escribir un descriptor de despliegue \(en inglés\).](#)

3.- El servidor de aplicaciones Tomcat.

Caso práctico

Mientras **Juan** sigue investigando e investigando, a la empresa **BK programación** le ha surgido un nuevo proyecto: una aplicación web para gestionar las citas de una clínica médica; y **Ada**, sabiendo que **María** es muy capaz, decide encargarle que evalúe las necesidades técnicas del nuevo proyecto. María enseguida ve que es necesario contar con un servidor de aplicaciones y decide tratarlo con **Juan**. Después de hablarlo detenidamente con él, y después de haber estudiado las posibles opciones, ambos han concluido que lo mejor es **instalar un servidor Tomcat**.

Sin embargo, **María** no para de darle vueltas. Aunque parece claro que el servidor Tomcat será suficiente para el proyecto, decide instalarlo y hacer pruebas con el mismo antes de informar a **Ada**.



¿Crees que Tomcat cumplirá su cometido para este proyecto? Quizás te falta información para poder llegar a una conclusión.

Apache Tomcat es, a la vez, un servidor HTTP y un contenedor web, desarrollado bajo el paraguas de la fundación Apache, y por supuesto, es software libre. Como contenedor web, está centrado en permitir ejecutar aplicaciones web que están formadas por servlets y JavaServer Pages, por lo que normalmente se le dice de él que es solamente un contenedor de servlets. Aparte de esas tecnologías, implementa otras como Java EL o Java WebSockets.



shane cururu |CC BY-NC

¿Recuerdas el apartado anterior? ¿Cuál era la función de un contenedor web? ¿Y cuál es la función de un servidor HTTP? Veamos:

Tomcat, como servidor HTTP recibe las peticiones http y las redirige al contenedor web, el cual ejecutará los componentes web necesarios, que en este caso serán servlets o archivos JSP. Una vez que procesada la petición por el componente web adecuado, el resultado vuelve a ser enviado al cliente web (navegador) por el servidor HTTP.

Esa idea, a priori sencilla, requiere que Tomcat tenga una arquitectura jerárquica compuesta de diversos componentes, cada uno encargado de una función. Vamos a aproximarnos a los elementos que componen su estructura interna de la forma más sencilla posible:

- ✓ **Servidor.** Representa al contenedor de servlets en si mismo, es decir, a Tomcat en todo su conjunto. Dentro de él se ejecutan el resto de componentes.
- ✓ Dentro del **servidor** podemos tener uno o más **servicios** ejecutándose. La función de un **servicio** es enlazar el componente encargado de comunicarse con el cliente, con la parte que procesa la petición y genera una respuesta. Veamos, dentro de un servicio podemos tener:
 - ◆ Uno o más **conectores** (connectors). Es el componente encargado de recibir peticiones del cliente remoto y de pasárselas al motor (engine), una vez que el motor procesa la petición, el conector hará llegar la respuesta al cliente remoto.
 - ◆ **Motor** (engine). Será el encargado de procesar la petición y de devolver el contenido generado al conector para que lo haga llegar al cliente remoto. Para lograr su cometido tendrá que ejecutar, si se requiere, la aplicación web que hemos diseñado.
 - Dentro de un motor, podremos tener uno o varios **Hosts**. El propósito del **Host** es asociar un nombre de equipo al servidor Tomcat (www.juntadeandalucia.es, por ejemplo), de tal manera que nuestro servidor Tomcat pueda responder peticiones asociadas a un nombre de equipo concreto.

Dentro de Tomcat una aplicación web equivaldrá a un contexto (context), que se definirá y ejecutará dentro de un Host.

Aunque no vamos a profundizar demasiado en la estructura interna de Tomcat, entenderla facilita su posterior configuración, dado que cada componente tiene su parte en el proceso de configuración.

Aparte de los componentes anteriores hay otros elementos no nombrados. Por ejemplo, el motor Jasper, que es el encargado de compilar los archivos JSP convirtiéndolos en servlets.

Debes conocer

Dependiendo de la versión de Apache Tomcat que vayamos a instalar dispondremos de una serie de tecnología u otras, y necesitaremos una versión de Java u otra. Por ejemplo:

- ✓ Apache Tomcat 9.x.x necesita Java 8 o superior, y soporta la especificación de Servlets versión 4.0.
- ✓ Sin embargo, las versiones de Tomcat 8.x.x necesitan Java 7 o superior, y soportan la especificación de Servlets versión 3.1.

Esto es importante, porque no puedes ejecutar ciertas versiones si no tienes instalada la versión adecuada de Java, y además, dependiendo de la especificación de Servlets que use la aplicación web desarrollada necesitarás una versión de Tomcat específica. Puedes ver más información al respecto aquí:

 [Versiones de Tomcat](#)

Autoevaluación

Las funciones del servidor web Apache y las funciones del servidor Tomcat, ¿son equivalentes?

- Sí.
- No.

Reflexiona

Seguramente te habrás preguntado porqué Tomcat implementa solo una parte de la especificación Java EE, en lugar de implementarla entera. ¿Por qué se quedan a medias?

Realmente Tomcat es una parte de la implementación de Java EE que en asociación con otros componentes pueden crear un servidor Java EE completo. Es el caso de TomEE, también bajo el paraguas de la fundación Apache:

 [Web oficial de TomEE](#)

3.1.- Instalación.

Caso práctico

María está en proceso de hacer una instalación para hacer pruebas con Tomcat. Una consulta rápida por Internet le ha proporcionado un montón de enlaces sobre diferentes métodos y formas de instalación. ¿Cuál será el mejor método de instalación? No quiere perder demasiado el tiempo.

Después de investigar un poco más, concluye que, entre las innumerables configuraciones que podría haber, existen tres configuraciones que se repiten bastante, pero todavía no sabe cuál es la mejor para llevar a cabo sus pruebas.



¿Qué forma de instalar Tomcat le aconsejarías a **María**? Instalar Tomcat no es algo excesivamente complejo. Cuando se instala un servidor Tomcat en un entorno de desarrollo (donde se prueban aplicaciones web) no es normal hacer instalaciones complicadas, pero Tomcat permite instalaciones de diferentes tipos, sobre todo en producción (donde se ejecutan aplicaciones web que son usadas por usuarios reales). Veamos algunas formas de instalarlo y configurarlo:



[William Warby Trabajando en instalar Tomcat. CC BY](#)

1.- Instalar Tomcat como un servidor independiente.

En esta situación solo existe un servidor Tomcat que escucha las peticiones de los clientes y ejecuta las aplicaciones web. Es una de las formas más habituales de instalarlo, tanto para desarrollo, como para producción.

2.- Integración de Tomcat con un servidor web existente. Es normal que cuando una organización evoluciona, decida desarrollar ciertas aplicaciones web con unas tecnologías y otras con otras tecnologías completamente diferentes. En este tipo de situaciones pueden darse escenarios complejos, por ejemplo:

2.1.- Puede existir la necesidad de ejecutar aplicaciones web basadas en PHP, y por otro lado, aplicaciones web basadas en la especificación servlets usando Tomcat. En este escenario podemos configurar un servidor web Apache para que las peticiones asociadas al servlet se redirijan a un servidor Tomcat de forma transparente para el usuario.

2.2.- En un momento dado es posible que tengamos dos aplicaciones web que, por su naturaleza y demanda, estén saturando el servidor donde se ejecutan. En esta situación se pueden mover ambas aplicaciones web a dos máquinas diferentes, cada una con su servidor Tomcat independiente. En este caso, podríamos poner una tercera máquina con un servidor web Apache que redirigiera la petición web al servidor Tomcat correspondiente de forma transparente para el usuario.

3.- Instalación de un clúster de servidores Tomcat. Cuando una aplicación web tiene muchísima demanda, una de las soluciones más habituales para dar un servicio decente al usuario, es tener varias máquinas o nodos con su propio servidor Tomcat ejecutando la misma aplicación web, en lo que se conoce como un clúster. Este proceso requiere de una estructura compleja en la que un servidor web Apache balancea transparentemente la carga de trabajo a diferentes servidores Tomcat, y donde los servidores Tomcat intercambian información entre sí para dar continuidad al servicio (es decir, para que el usuario no perciba si en un momento dado, sus peticiones están siendo atendidas por un nodo o por otro).

En los casos 2 y 3 antes descritos, la comunicación entre Tomcat y el servidor web Apache se realiza a través de un protocolo llamado AJP (Apache JServ Protocol), como se verá más adelante.

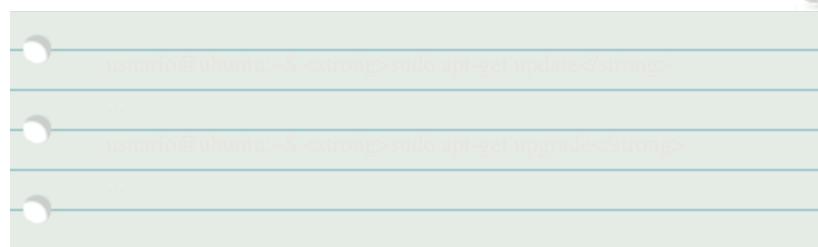
Autoevaluación

A la hora de instalar Tomcat en un entorno de desarrollo, ¿cuál sería la mejor opción?

- Instalarlo en clúster, dado a que es un entorno muy cercano al de producción.
- Instalarlo como un servidor independiente.
- Integrarlo con otro servidor web.

3.1.1.- Instalación desde repositorio.

La instalación desde repositorio es algo muy habitual en sistemas operativos basados en Linux. En este caso vamos a describir como se podría hacer la instalación de Tomcat en Ubuntu 17.10 desde el repositorio, es decir, descargando los paquetes desde el repositorio de aplicaciones de la propia distribución Ubuntu, pero el proceso es extensible a casi todas las distribuciones actuales basadas en Debian (como es el caso de Ubuntu). Lo primero que deberíamos pensar es en actualizar el repositorio de paquetes y instalar las posibles actualizaciones:



nchenga CC BY-NC

Actualización del repositorio con las últimas actualizaciones.

Ubuntu/Ubuntu 17.10: Actualización del repositorio de paquetes.

...

Después tenemos que comprobar es si existen los paquetes de Tomcat en el repositorio y que versión hay disponible. Para ello podemos comprobar los paquetes existentes ejecutando el comando apt-cache search tomcat en un terminal:

...
tomcat8-admin - Apache Tomcat 8.0.30 Admin JSP Engine - common web applications
tomcat8-common - Apache Tomcat 8.0.30 Admin JSP Engine - common files
tomcat8-docs - Apache Tomcat 8.0.30 Admin JSP Engine - Documentation
tomcat8-examples - Apache Tomcat 8.0.30 Admin JSP Engine - example web applications

La opción -n limita la búsqueda, buscando paquetes cuyo nombre contenga la cadena tomcat, lo cual reduce los resultados. Como se puede ver en el ejemplo anterior, en el repositorio de aplicaciones está disponible Tomcat 8, y además, su instalación estaría dividida en varios paquetes. Esto puede variar de una distribución de linux a otra, en este caso, los paquetes serían:

- ✓ Paquetes de la instalación principal, que son tomcat8 y **tomcat8-common**.
- ✓ Gestor de aplicaciones de Tomcat, que corresponde con el paquete tomcat8-admin.
- ✓ Documentación de Tomcat, que corresponde con el paquete tomcat8-docs.
- ✓ Y un conjunto de aplicaciones web de ejemplo, que corresponde con el paquete tomcat8-examples.

Como ya sabes, Tomcat 8 depende de Java, por lo que tendrá que instalarse Java sin más remedio. No te preocupes por esto, si lo instalas desde repositorio se instalarán las dependencias de forma automática. Para saber de que paquetes depende Tomcat puedes ejecutar el comando apt-cache depends tomcat8 tomcat8-common :

Con esto, ya tenemos instalado Tomcat y listo para ejecutarse. No hay que hacer nada más, no obstante, hay varias cosas que debes saber después de instalarlo. La primera es, **¿dónde se ha instalado el JDK?** Para ello podemos recurrir al comando `dpkg -L` al que le pasamos como parámetro el paquete del que queremos saber sus archivos (en este caso `openjdk-8-jdk-headless`):

```
root@Ubuntu-VM:~# dpkg -L openjdk-8-jdk-headless
```

En este caso, se mostrará la lista de archivos que contiene el paquete anterior, y podemos ver que el JDK se ha instalado en la carpeta base `/usr/lib/jvm/java-8-openjdk-amd64`. Si fuese necesario, la variable de entorno `JAVA_HOME`, que debe contener la carpeta de instalación del JDK, debería apuntar a dicha carpeta. En este caso no es necesario.

Otra cosa importante es saber, **¿dónde está instalado Tomcat?** Para ello podemos también usar el comando anterior y estudiar la estructura de carpetas de los paquetes relacionados con Tomcat. En este caso, lo más normal es que tengamos las siguientes carpetas importantes:

- ✓ `/var/lib/tomcat8`: carpeta principal de Tomcat.
- ✓ `/etc/tomcat8`: carpeta está la configuración de Tomcat. Esta carpeta también está enlazada en la carpeta principal anterior y será usada para configurar muchos aspectos de este servidor.
- ✓ `/var/log/tomcat8`: carpeta que contiene los logs del servidor Tomcat.
- ✓ `/var/lib/tomcat8/webapps`: carpeta que contiene las aplicaciones web en formato expandido, aquí será donde instalaremos nuestras aplicaciones web.
- ✓ `/usr/share/tomcat8/bin`: carpeta que contiene los scripts para iniciar, parar, comprobar la configuración, etc. de Tomcat.

En otro tipo de instalaciones, como la instalación manual, es necesario configurar las variable de entorno `CATALINA_BASE` y `CATALINA_HOME`, pero en este caso, de momento, no es necesario.

Y la tercera cosa a saber es, **¿cómo se sabe si se está ejecutando Tomcat? ¿Cómo puedo arrancarlo o detenerlo?** Esta pregunta puedes responderla de varias formas. Una de ellas es con el comando `service`:

```
root@Ubuntu-VM:~# service tomcat8 status
```

```
* tomcat8.service - LSB: Start Tomcat
   Loaded: loaded (/etc/init.d/tomcat8; generated, vendor preset: enabled)
   Active: active (running) since Wed, 2018-03-07 17:21:23 UTC; 1 min ago
     Docs: man:systemctl(1)
```

El comando anterior te permitirá saber el estado del servicio (activo en este caso). Asociado al comando anterior, tenemos una lista de comandos para operar con el servicio (iniciarlo, detenerlo, etc.). Como aquí la lista de comandos puede ser bastante larga, vamos a verlos de forma resumida:

- ✓ Comandos para comprobar el estado de Tomcat:
 - ◆ service tomcat8 status: nos dirá si se está ejecutando o no.
 - ◆ sudo /etc/init.d/tomcat8 status: es equivalente al comando anterior.
- ✓ Comandos para arrancar Tomcat (se puede usar cualquiera de ellos):
 - ◆ service tomcat8 start
 - ◆ sudo /etc/init.d/tomcat8 start
- ✓ Comandos para detener Tomcat (se puede usar cualquiera de ellos):
 - ◆ service tomcat8 stop
 - ◆ sudo /etc/init.d/tomcat8 stop
- ✓ Comandos que permiten comprobar el puerto que está escuchando Tomcat para aceptar peticiones HTTP:
 - ◆ sudo ss -tpl
 - ◆ sudo netstat -tpl

Veamos, a modo de ejemplo, el comando ss en acción:

user@centminme:~\$ sudo ss -ltp	
LISTEN 0 5 127.0.0.1:443	user@centminme:~\$ sudo ss -ltp
LISTEN 0 128 0.0.0.0:80	user@centminme:~\$ sudo ss -ltp
LISTEN 1024 0 0.0.0.0:8080	user@centminme:~\$ sudo ss -ltp
LISTEN 0 3 0.0.0.0:8080	user@centminme:~\$ sudo ss -ltp
LISTEN 1024 0 0.0.0.0:8080	user@centminme:~\$ sudo ss -ltp
LISTEN 0 128 0.0.0.0:8080	user@centminme:~\$ sudo ss -ltp

Como se puede ver en el esquema anterior, el servidor Tomcat está escuchando en los puertos 8005 y 8080 (http-alt significa puerto HTTP alternativo).

En este apartado se ha descrito la instalación desde repositorio en Ubuntu 17.10. **La instalación en otros sistemas derivados de la distribución Debian (como es el caso de Ubuntu) es similar, aunque pueden variar ligeramente las carpetas de instalación y los nombres de los servicios en función de la versión (en vez de ser tomcat8 puede ser tomcat9, por ejemplo).**

Después de arrancar el servicio Tomcat, puedes acceder a la página web por defecto usando un simple navegador. Para ello, puedes poner la URL `http://localhost:8080` en la barra de direcciones del navegador (si estás accediendo desde el mismo equipo donde has instalado Tomcat), o bien la URL `http:// DIRECCION_IP:8080` donde DIRECCION_IP es la dirección IP del equipo donde has hecho la instalación.

Autoevaluación

¿Con cuál de los siguientes comandos podemos instalar Tomcat?

- service install tomcat8
- apt-get install tomcat8
- ss -tpl install tomcat8
- apt-cache install tomcat8

3.1.2.- Instalación manual.

En este apartado vamos a proceder a instalar todos los archivos necesarios manualmente. Esto nos permitirá un grado mayor de control, pero puede ser ligeramente más complicado.

En primer lugar, como en el caso anterior, tendremos que instalar el JDK. En esta ocasión, en vez de usar OpenJDK, recurriremos al JDK de Oracle que podemos descargar de la siguiente página:

 [Descargar Java SE Development Kit de Oracle](#)



Ginny Instalando Tomcat de forma manual. CC BY-SA

Aquí tendremos que tener cuidado, porque deberemos elegir la versión del JDK que necesita la versión de Tomcat que deseamos instalar. En este caso instalaremos JDK 8, para luego instalar Tomcat 9. Para instalarlo procedemos a descargar la versión correspondiente a nuestra plataforma (en este caso escogeremos la versión Linux x64 empaquetada como archivo .tar.gz).

Aquí vamos a mostrar como se realizaría la instalación en Ubuntu 17.10, aunque el procedimiento de instalación es similar en otros entornos Linux. Para empezar, usaremos la carpeta /opt para realizar toda la instalación, lo cual es una opción muy habitual, dado que esa carpeta se suele usar para instalar software que no se incluye en la instalación por defecto. Veamos como sería el proceso:

```
1. Descargando el JDK: sudo apt update  
2. Cambiando contraseña del usuario: sudo passwd user  
3. Actualizando VirtualBox: sudo apt update  
4. root@user:~# tar -xvf jdk-8u171-linux-x64.tar.gz -C /opt
```

En el proceso anterior se da por sentado que el JDK se habrá descargado comprimido como .tar.gz se habrá descargado en la carpeta de sistema /opt (si no está en dicha carpeta, habrá que copiarlo). Si dicha carpeta no existe, podemos crearla manualmente. Una vez realizada la descompresión, tendremos el JDK descomprimido en una carpeta como opt/jdk1.8.0_171 (esta carpeta puede cambiar dependiendo de la versión del JDK instalada).

Después de descomprimir el JDK debemos configurar una serie de variables de entorno. De cara a esta configuración tenemos varias opciones, entre las que podemos destacar:

- ✓ Añadir las variables de entorno en el archivo /etc/environment. Este archivo contiene las variables de entorno globales para todo el sistema y quizás es un buen lugar para configurarlas.
- ✓ Añadir las variables de entorno al final del archivo /etc/profile. Esta es una alternativa muy utilizada.
- ✓ Y la opción más elegante, es crear un script en la carpeta /etc/profile.d/ que se ejecutará al iniciar sesión y cargará las variables de entorno. El script podría llamarse, por ejemplo, jdk.sh y podría contener el siguiente código:

```
#!/bin/bash

#Establece la variable JAVA_HOME a la raíz de la instalación del JDK
JAVA_HOME=/usr/lib/jvm/java-8-oracle

#Modifica el path para poder ejecutar los comandos de Java
PATH=$PATH:$JAVA_HOME

#Exporta las variables para hacerlas variables de entorno
export JAVA_HOME
export PATH
```

Estos archivos y carpetas pueden cambiar dependiendo de la distribución de Linux que usemos, estamos hablando del caso concreto de Ubuntu. El código anterior podríamos ejecutarlo de la siguiente forma, aunque al reiniciar la sesión ya estaría disponible:

```
root@ubuntu:~$ source /etc/profile.d/java.sh
```

Una vez instalado el JDK correspondiente, debemos instalar Tomcat y configurar sus respectivas variables de entorno de forma similar a como hemos hecho con el JDK. Para descargar Tomcat siempre recurrimos a la página oficial, en este caso, instalaremos Tomcat 9:

 [Descargar Tomcat 9.0.8 de la página oficial.](#)

De todos los archivos disponibles descargaremos la distribución binaria (binary distribution) del núcleo (core), empaquetada como un archivo .tar.gz. Procedemos a copiarla a la carpeta /opt y la descomprimimos con el mismo comando usado para descomprimir el JDK anterior:

```
root@ubuntu:~$ cd /opt
root@ubuntu:~$ wget https://tomcat.apache.org/tomcat-9.0.8/bin/apache-tomcat-9.0.8.tar.gz
root@ubuntu:~$ tar -xvzf apache-tomcat-9.0.8.tar.gz
```

Una vez descomprimido, se creará una carpeta con nuestro Tomcat (en este caso /opt/apache-tomcat-9.0.8, aunque el nombre de esta carpeta variará en función de la versión descargada). Dentro de esa carpeta, que será la carpeta principal de Tomcat, encontraremos varias carpetas importantes (X.Y.Z hace referencia a que puede ser cualquier versión del servidor):

- ✓ /opt/apache-tomcat-X.Y.Z/bin: carpeta donde encontrarás los ejecutables de Tomcat (como el script catalina.sh usado para arrancar y detener el servidor).
- ✓ /opt/apache-tomcat-X.Y.Z/conf: carpeta donde encontrarás los archivos de configuración de Tomcat.
- ✓ /opt/apache-tomcat-X.Y.Z/webapps: carpeta que contiene las aplicaciones web en formato expandido, aquí será donde instalaremos nuestras aplicaciones web.
- ✓ /opt/apache-tomcat-X.Y.Z/logs: carpeta donde se guardarán los logs del servidor.

Lo más normal ahora es configurar la variable de entorno CATALINA_HOME para que apunte a la carpeta base de Tomcat y la variable PATH para que pueda accederse fácilmente a los scripts de arranque de Tomcat (situated en la carpeta bin anterior). Para ello optaremos por crear un archivo llamado tomcat.sh en la carpeta /etc/profile.d/:

```
#!/bin/bash

#Establece la variable CATALINA_HOME a la raiz de la instalación de Tomcat
CATALINA_HOME=
#Modifica el path para poder ejecutar los scripts de arranque de Tomcat
PATH=$PATH $CATALINA_HOME
#Exporta las variables para hacerlas variables de entorno
export CATALINA_HOME
export PATH
```

Como ocurría con el JDK, el código anterior se ejecutará en el próximo inicio de sesión, aunque podemos activarlo en este momento ejecutando el siguiente comando:

```
root@ubuntu-04:~/source/010/prome# ./catalina.sh
```

Una vez que hemos realizado todo esto, podemos iniciar y detener Tomcat a través del script catalina.sh que ya viene incluido en nuestra instalación de Tomcat. El parámetro start iniciará Tomcat:

```
root@ubuntu-04:~/source/010/prome# sudo ./catalina.sh start
Tomcat initialized successfully using port(s) 8080.
Tomcat is running on port(s) 8080. To stop Tomcat, run:
  /usr/local/tomcat/bin/shutdown.sh
  or
  /usr/local/tomcat/bin/catalina.sh stop
  or
  /usr/local/tomcat/bin/catalina.sh -stop
  or
  /usr/local/tomcat/bin/catalina.sh -stopwait
  or
  /usr/local/tomcat/bin/catalina.sh -stopnow
Tomcat started.
```

Aquí es importante ejecutar sudo con "-i" para que cargue las variables de entorno.

Ahora podremos acceder al servidor Tomcat usando un simple navegador web. Para ello, podemos poner la URL <http://localhost:8080> en la barra de direcciones del navegador (si estás accediendo desde el mismo equipo donde has instalado Tomcat), o bien la URL http://DIRECCION_IP:8080 donde DIRECCION_IP es la dirección IP del equipo donde has hecho la instalación.

Esta URL te mostrará la característica página de inicio de Tomcat.

Para detenerlo simplemente ejecutamos el comando catalina.sh, pasándole como argumento stop:

```
root@ubuntu-04:~/source/010/prome# sudo ./catalina.sh stop
```

Para saber más

Instalar Tomcat en otros sistemas operativos, como Windows, también es posible. Échale un vistazo a la siguiente página web para saber como:

 [Instalar Tomcat en Windows.](#)

3.2.- Arquitectura de Tomcat.

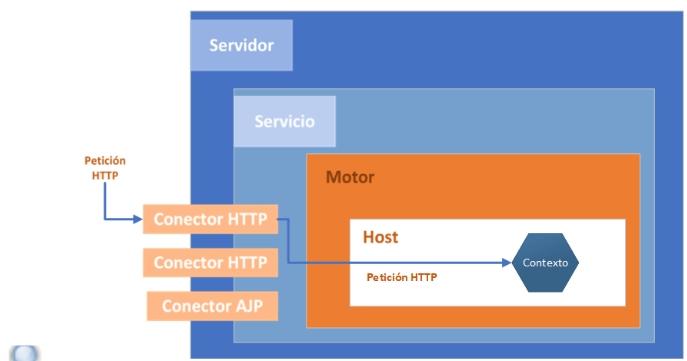
Caso práctico

No fue fácil, pero **María** consiguió instalar Tomcat antes de lo que pensaba. Después de instalarlo se ha puesto a indagar en los directorios y esta intentando ver que archivos hay por aquí y por allí. Localiza el directorio donde van las aplicaciones web, el directorio donde van los archivos de configuración, etc. Después de verlo, piensa, ¿cómo funciona todo esto? Habla con **Juan** sobre el tema y entre ambos, investigando un poco, logran entender como funciona a un nivel general.

¿Por qué crees que **María** esta tan interesada en entender como funciona internamente Tomcat? Antes se hablo de forma introductoria de la arquitectura de Tomcat, es decir, de como Tomcat está compuesto internamente. Pero, ¿por qué es necesario saber como está compuesto Tomcat internamente? Pues porque a la hora de configurarlo es necesario saber exactamente que parte de Tomcat se ve afectada por una configuración concreta.

Tomcat está formado por una serie de componentes anidados (uno dentro de otro) **formando una jerarquía**.

Como ya se comentó antes, Tomcat es una combinación de un servidor HTTP y un contenedor de servlets. Antes de ahondar en como funciona, vamos a ver un pequeño diagrama que explica como funciona Tomcat internamente:



Elaboración propia (Salvador Romero Villegas). *Estructura interna de un servidor Tomcat.*
(Dominio público)

En esta estructura, el servidor (**server**) es el elemento central de Tomcat, y representa al servidor en si mismo. Dentro del servidor hay uno o más servicios ejecutándose. La misión del servicio (**service**) es simplemente enlazar los conectores (**connector**), que recibirán las peticiones web, con el motor (**engine**), que será el encargado de llevar la petición al componente web adecuado.

Veamos como sería el proceso que seguiría Tomcat para atender una petición HTTP de forma abreviada:

- 1.- El conector HTTP recibirá la petición HTTP del cliente web. Como cada conector HTTP está configurado dentro de un servicio (service), y dentro del servicio habrá obligatoriamente un único motor (engine), la petición HTTP se derivará siempre al motor asociado al servicio.
- 2.- El motor (engine), al recibir la petición HTTP del conector, la analiza. Averigua entonces el nombre de equipo al que fue realizada la petición, información que extraerá generalmente de las cabeceras de la petición HTTP, donde el cliente web suele incluir la cabecera Host con el nombre de la equipo destino (www.juntadeandalucia.es, por ejemplo).
- 3.- Dentro del motor habrá configurados uno o más hosts, cada uno configurado para un nombre de equipo diferente. El motor, gracias a la información extraída en el paso anterior, sabrá entonces a que host debe trasladar la petición HTTP. El hecho de poder tener más de un host dentro de un motor permite tener hosts virtuales (como ocurría con el servidor web Apache).
- 4.- Un host ahora recibe la petición HTTP desde el motor. En el host puede haber uno o más contextos, que serán básicamente las aplicaciones webs y las rutas a dichas aplicaciones web. Al recibir la petición web, el host la derivará a un contexto concreto, en base a la ruta indicada en la misma petición HTTP. Por ejemplo, si la petición fue GET /app1 HTTP/1.1, esta se derivaría a una aplicación web llamada app1, mientras que si fuera GET /app2 HTTP/1.1, se derivaría a la aplicación web app2.
- 5.- Una vez reenviada la petición a un contexto concreto, pueden ocurrir dos cosas:
 - 5.1.- Que la petición HTTP requiera servir solo un recurso estático (como una imagen), lo cual se resuelve rápidamente enviando el contenido del archivo correspondiente.
 - 5.2.- Que la petición HTTP requiera que un componente web procese la petición y genere un resultado. En este caso, la petición se reenviará al componente web correspondiente (un servlet, por ejemplo), y este se ejecutará, generando una respuesta que se devolverá al cliente web.

Todo el proceso anterior, que puede parecer relativamente complejo, permite tener mucha flexibilidad a la hora de configurar el servidor Tomcat, permitiendo intercalar otros elementos que interceptarán las peticiones HTTP, tal y como se verá más adelante.

Debes conocer

¿Sabes qué diferencia hay entre contexto y aplicación web? ¿Es lo mismo? Pues aunque a ojos de Tomcat son lo mismo, en realidad, no son lo mismo:

- ✓ Una aplicación web es algo exportable y con un formato común a todos los servidores de aplicaciones. **Una aplicación web es la aplicación tal cual es desarrollada por el equipo de desarrollo**, la cual, **se puede desplegar en un servidor de aplicaciones que soporte Java EE**. Esta puede incluir el archivo <code>WEB-INF/web.xml, llamado descriptor de despliegue que indica como se comporta la aplicación internamente.
- ✓ En cambio, un contexto es algo propio de Tomcat. **El contexto corresponde con la configuración necesaria para ejecutar una aplicación en un servidor Tomcat concreto**, integrándola en un ambiente de ejecución concreto. Por ejemplo, podemos tener la misma aplicación usando una base de datos de pruebas, y por otro lado usando una base de datos real, sería la misma aplicación pero en ambientes diferentes.

La configuración del contexto se realiza en el momento de instalar la aplicación en un servidor de aplicaciones.

3.3.- Configuración básica de Tomcat.

La siguiente pregunta a responder ahora, es, **¿dónde se configura Tomcat?** Tomcat dispone de varios archivos de configuración que podemos modificar para adaptar el funcionamiento de Tomcat a nuestras necesidades. Estos archivos podemos encontrarlos en la carpeta de configuración de Tomcat, dependiente del mecanismo de instalación que hayamos elegido. Todos ellos, tienen formato XML, por lo que es necesario que respes la estructura y sintaxis de este tipo de documentos (con una configuración errónea, Tomcat no arrancará). Los archivos de configuración disponibles son los siguientes (conf/ hace referencia a la carpeta de configuración):

- ✓ conf/server.xml: es el archivo de configuración principal, donde encontraremos configurados los diferentes componentes de Tomcat (servidor, servicios, conectores, motor, hosts, etc.). Aquí es importante tener en cuenta como se anidan unos componentes dentro de otros (por ejemplo, un conector solo puede configurarse dentro de un servicio).
- ✓ conf/context.xml: es el archivo de configuración del contexto base de todas las aplicaciones web que se ejecutan en Tomcat. Este archivo no debe modificarse. Cada aplicación web puede tener su propio contexto de ejecución que complementan al principal, pero para ello hay que usar otros archivos (como el archivo META-INF/context.xml dentro de la aplicación web desplegada).
- ✓ conf/web.xml: este archivo contiene el descriptor de despliegue usado por defecto para las aplicaciones web desplegadas en Tomcat. Dentro de una aplicación web, los valores de esta configuración se pueden complementar a través del archivo WEB-INF/web.xml (el cuál no es obligatorio y se crea en tiempo de desarrollo), pero no debes eliminar ni modificar el archivo conf/web.xml.
- ✓ conf/tomcat-users.xml: este archivo contiene, entre otras cosas, la configuración de los usuarios que podrán usar el gestor de aplicaciones web de Tomcat. Este gestor, descrito más adelante, permite desplegar aplicaciones cómodamente desde un entorno web.

De los cuatro archivos anteriores, el más importante es el archivo server.xml y es el que vamos a describir a continuación de forma general. La idea es que conozcas su estructura para luego poder añadir o quitar elementos de configuración. Este archivo comienza con un conjunto de etiquetas XML que definen el servidor en su conjunto:

```
port="8005" shutdown="SHUTDOWN">>
```

```
...
```

Dentro de esas etiquetas encontraremos definidos uno o más servicios (el atributo name define el nombre del servicio, donde no puede haber dos servicios con el mismo nombre):

```
port="8005" shutdown="SHUTDOWN">>
```

```
...
```

```
name="Catalina">>
```

```
...
```

```
...
```

Dentro de un servicio podrá haber uno o más conectores. Como se puede observar a continuación cada conector define entre otras cosas el puerto donde se escucharán las peticiones y el protocolo aceptado en dicho puerto:

```
port="8005" shutdown="SHUTDOWN">
...
  name="Catalina">
    port="8080" protocol="HTTP/1.1" connectionTimeout="20000" redirectPort="8443" />
...
...

```

Además, dentro del servicio habrá un único motor, al que se reenviarán las peticiones recibidas por los conectores:

```
port="8005" shutdown="SHUTDOWN">
...
  name="Catalina">
    port="8080" protocol="HTTP/1.1" connectionTimeout="20000" redirectPort="8443" />
...
  name="Catalina" defaultHost="localhost">
...
...

```

En un motor (engine) podemos especificar bastantes atributos de configuración, entre ellos tenemos: name que será el nombre del motor (el cual no se puede repetir en otro motor) y defaultHost, que corresponderá al host por defecto que atenderá las peticiones cuando no se haya configurado ningún host específico para un nombre de equipo concreto. Como aquí se ha especificado que el defaultHost será localhost, deberá existir un host configurado con dicho nombre. Véamos:

```
port="8005" shutdown="SHUTDOWN">
...
  name="Catalina">
    port="8080" protocol="HTTP/1.1" connectionTimeout="20000" redirectPort="8443" />
...
  name="Catalina" defaultHost="localhost">
...
    name="localhost" appBase="webapps" unpackWARs="true" autoDeploy="true">
...

```

La etiqueta <Host> permitirá configurar un host dentro del motor. Esta etiqueta tiene muchos atributos de configuración, pero el atributo name aquí es importante, dado que permite indicar cual será el nombre DNS del equipo. La posibilidad de tener varios <Host> dentro de un <Engine>, permite que nuestro servidor Tomcat pueda ejecutar diferentes aplicaciones webs asociadas a diferentes nombres DNS. Es la misma idea que los Virtual Hosts del servidor web Apache. De los otros atributos, cabe destacar los siguientes: appBase indica cual será el directorio que contendrá las aplicaciones web para un <Host> concreto, y autoDeploy indica que las aplicaciones bajo el directorio appBase se desplegarán y pondrán en marcha automáticamente.

Ahora es un buen momento para que le eches un vistazo al archivo server.xml incluido en tu instalación de Tomcat, y que localices los elementos explicados antes. Verás que hay otros elementos de configuración no explicados (cuidado, las etiquetas <!-- --> son comentarios), pero con lo visto aquí es suficiente por ahora.

Autoevaluación

¿Dónde encontraríamos un conector?

- Justo dentro de un <Service>.
- Dentro de un <Host>.
- Dentro de un <VirtualHost>.
- Justo dentro de un <Engine>.

3.4.- Servir una aplicación web con Tomcat.

Caso práctico

Finalmente **María**, ha montado una máquina Ubuntu con el servidor de aplicaciones Tomcat para que los miembros de **BK programación** puedan desplegar, en dicho servidor, las aplicaciones web que consideren necesarias. **Juan** ha realizado una primera prueba de despliegue y ha documentado todos y cada uno de los pasos que es preciso realizar para que la aplicación web quede totalmente operativa en el servidor, y así cualquier cliente de la empresa pueda disfrutar de la funcionalidad de la aplicación.



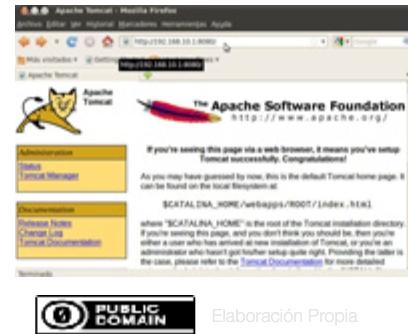
Y tú, ¿sabrías como desplegar una aplicación web en Tomcat? Como ya habrás deducido de apartados anteriores, desplegar una aplicación web consiste en instalar dicha aplicación web en el servidor de aplicaciones, que en este caso concreto hace referencia a Tomcat. Una vez instalada la aplicación, el servidor de aplicaciones podrá ejecutar los componentes web y servir el contenido estático bajo demanda (cuando el cliente web realice una petición).

Como estamos tratando aplicaciones web que siguen las especificaciones Java EE, cuya estructura ya se explico en apartados anteriores, podremos desplegar nuestra aplicación web en cualquier servidor que implemente dichas especificaciones (o algunas de ellas, como es el caso de Tomcat).

No obstante, hay que tener cuidado, **si nuestra aplicación web se basa en unas versiones concretas de las tecnologías asociadas**, por ejemplo servlets 4.0 o JSP 2.3, **es altamente probable que no funcione del todo bien en un servidor que implementa versiones anteriores de dichas especificaciones**.

En el proceso de despliegue, el servidor de aplicaciones debe enlazar el contenedor de servlets con los diferentes componentes web que tenga la aplicación web, dado que los componentes webs se consideran como una extensión del mismo servidor. Este proceso, se puede realizar de diferentes formas. Lo más habitual es posiblemente desplegar la aplicación de forma dinámica (con el servidor Tomcat en funcionamiento), para lo cual se disponen de dos mecanismos fundamentales:

- ✓ **Copiar y configurar manualmente las aplicaciones web.** En este caso copiaríamos la aplicación web en el directorio base de despliegue. Este directorio base aparece configurado en el archivo server.xml, y en la configuración por defecto de Tomcat es el directorio webapps.
- ✓ **Realizar un despliegue remoto.** Tomcat permite hacer despliegue remoto de diversas formas:
 - ➡ Usando el gestor de aplicaciones web de Tomcat, que facilita la gestión de las aplicaciones web desplegadas a través de una interfaz web amigable y sencilla.
 - ➡ Usando un software específico para realizar el despliegue (como Ant o Maven).



Elaboración Propia

Para saber más

En el siguiente enlace podrás encontrar más información sobre el despliegue de aplicaciones web en Tomcat:

 [Como desplegar aplicaciones web en Tomcat.](#)

3.4.1.- Despliegue de una aplicación web.

Caso práctico

Juan ha completado con éxito su primer despliegue, y se ha puesto a documentar el proceso. Después de consultar la documentación y diversas fuentes de Internet ha visto que existen diferentes alternativas de despliegue, y no está seguro de cual será la mejor.

Lo habla con **Ada**, y después de la conversación **Ada** le da un poco más de tiempo para investigar, y documentar el proceso.



Desplegar una aplicación web en Tomcat es muy sencillo. Para despegarla necesitamos uno de los dos siguientes elementos:

- ✓ El archivo WAR (Web application ARchive) de la aplicación web.
- ✓ O bien, la aplicación en formato de directorio expandido (también llamado desempaquetado).

Para proceder a la instalación primero debemos saber en que Host vamos a instalar la aplicación, dado que podremos tener más de uno. Esta información estará en el archivo server.xml del directorio de configuración:

A photograph of a peacock with its tail feathers fully spread, displaying a vibrant array of blue, green, and gold feathers. The peacock is standing on the ground, and its feathers are fanned out in a circular pattern.

```
name="localhost" appBase="webapps" unpackWARS="true" autoD  
...  
...
```

Por defecto, Tomcat viene configurado con un host llamado localhost, que hace referencia al propio equipo. Esta es una configuración ideal para desarrollo. Si te fijas en el atributo appBase verás que el directorio donde se espera que estén las aplicaciones web de este host se llama webapps. Como el atributo appBase no tiene una ruta absoluta, sino una ruta relativa, **dicho directorio se espera que esté en la carpeta base de instalación de Tomcat** (que dependerá del mecanismo de instalación elegido).

Tania.Paz CC BY-NC-SA

Para instalar ahora la aplicación simplemente la copiamos la aplicación en el directorio anterior. Si por ejemplo, hemos hecho una instalación desde repositorio de Tomcat 8 en una distribución Ubuntu 17.10, el comando podría ser el siguiente:

```
root@192.168.1.100:~/ejemplo# cp exemplo.war /var/lib/tomcat8/webapps/
```

En el comando anterior, la carpeta ./ejemplo contendría la aplicación web. Recuerda que si hemos hecho una instalación manual de Tomcat, el directorio destino sería otro, por ejemplo /opt/apache-tomcat-X.Y.Z/webapps.

Como en la configuración del Host por defecto (localhost) lleva la opción autoDeploy="true", la aplicación se desplegará automáticamente (si fuera false podría no ocurrir así). En este caso, no habría que hacer nada más, y la aplicación quedaría desplegada con la ruta / ejemplo, que corresponde con el nombre de directorio de la aplicación web dentro de webapps. Abriendo un navegador, podríamos acceder a nuestra aplicación web poniendo:

- ✓ <http://localhost:8080/ejemplo> (recuerda que el puerto por defecto es el 8080)

En esta configuración por defecto, si deseásemos que la aplicación fuera accedida a través de otra ruta (por ejemplo, <http://localhost:8080/agenda>), deberíamos renombrar el directorio de la aplicación web (cambiando /ejemplo por /agenda). En una configuración más avanzada se puede crear un archivo de contexto específico para la aplicación web (en XML), donde se indica la ruta a la aplicación web, pero no es normal en un servidor de desarrollo.

Como se comentó antes, la aplicación web también se puede desplegar como un archivo WAR, para ello, hacemos lo mismo, copiamos el archivo WAR en la carpeta correspondiente. Veamos un ejemplo:

```
root@192.168.1.100:~/ejemplo# cp ejemplo2.war /var/lib/tomcat8/webapps/
```

Funcionará de forma similar a la anteriormente descrita. Tomcat detectará el archivo WAR y lo desplegará automáticamente (gracias a la opción autoDeploy="true"). Como en la configuración del Host aparece la opción unpackWARs="true", Tomcat desempaquetará de forma automática el archivo WAR para proceder a ejecutar la aplicación web (si fuera false, lo ejecutaría sin desempaquetar, aunque no es lo más habitual). El nombre por tanto del archivo WAR (ejemplo2.war en este caso), sería también la ruta usada para el despliegue (/ejemplo2):

- ✓ <http://localhost:8080/ejemplo2>

Aunque parece lógico y evidente, es necesario saber que no podemos tener dos aplicaciones web con el mismo nombre en nuestro servidor web, dado que tendrían la misma ruta y Tomcat no sabría identificar cual es cual.

Por último, es posible desplegar una aplicación web que no necesite una ruta para ser accedida (como puede ser /ejemplo o /ejemplo2), es decir, es posible poner una aplicación web que responda directamente con el nombre DNS del equipo (escribiendo solamente <http://localhost:8080>), tal y como ocurre en las páginas web de verdad (como por ejemplo, <http://www.juntadeandalucia.es>). Obviamente, en el servidor Tomcat solo puede haber una aplicación por Host que de este tipo.

Para conseguir esto simplemente debemos desplegar nuestra aplicación con el nombre de directorio ROOT o el nombre del archivo WAR ROOT.war (respetando las mayúsculas y minúsculas), dentro de la carpeta appBase del Host correspondiente (que correspondería con la carpeta webapps en la instalación por defecto de Tomcat). Veamos un ejemplo:

También se podría copiar directamente la aplicación web de forma desempaquetada en el directorio anterior, y cambiarle el nombre a la carpeta por ROOT.

Cuando desarrollamos una aplicación web en un IDE, como puede ser NetBeans o Eclipse, en el mismo directorio tendremos el código fuente y la aplicación para su distribución (y despliegue obviamente). Recuerda que es muy importante no incluir en la aplicación desplegada el código fuente, porque podría ser accesible por un tercero cuyas intenciones desconocemos.

Para saber más

Esta web muestra, de forma amplia, el funcionamiento, configuración, instalación, administración, etc. del servidor de aplicaciones Tomcat, donde también podemos encontrar cómo desplegar aplicaciones.

 [Administración Apache Tomcat.](#)

Reflexiona

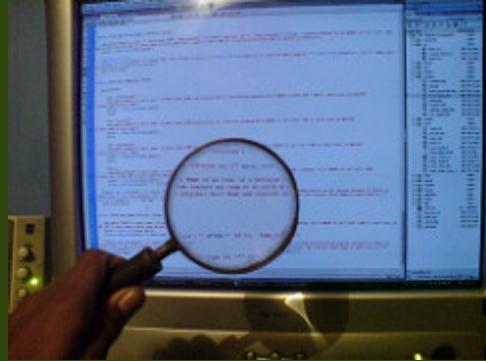
Si el equipo en el que hemos desplegado la aplicación web, pertenece a una red de computadores más amplia y tiene la dirección IP 192.168.1.10 y la máscara 255.255.255.0. ¿Podríamos acceder desde otros computadores a la aplicación web? En caso afirmativo, ¿cuál sería la URL que deberíamos teclear?

[Mostrar retroalimentación](#)

3.4.2.- Implementar el registro de acceso.

Caso práctico

Las aplicaciones web que están siendo desarrolladas por la empresa **BK programación** comienzan a ser probadas por los clientes en el servidor Tomcat de prueba. **Ada** quiere, de algún modo, realizar de algún seguimiento sobre estas aplicaciones, de manera que se puedan comprobar los accesos que han tenido, en qué momento han ocurrido, y qué recursos son los más demandados. Para ello **Ada** ha encargado a **Juan** y a **María**, configurar el servidor Tomcat para poder adaptar los logs, de manera que puedan facilitar más información sobre los accesos a las aplicaciones web.



Dumi Jay. Analizando los registros de acceso. (CC BY)

¿Sabes qué es un registro o log de acceso? Los servidores web y los servidores de aplicaciones permiten recopilar los accesos a una determinada aplicación web o a todo el servidor, incluyendo información como la dirección IP desde la que se produjo el acceso, la fecha y la hora. La información recopilada es almacenada generalmente en un archivo de disco.

Para configurar los registros de acceso en un servidor de aplicaciones Tomcat, tendremos que utilizar un componente propio de Tomcat llamado Valve (válvula en español). Las válvulas son elementos que se insertan en medio del flujo de procesado de una petición web, capturando la petición en un momento determinado y permitiendo realizar alguna acción o preprocesado antes de que pase al siguiente componente de Tomcat.

Por ejemplo, podríamos poner una válvula en un <Engine>, para capturar todas las peticiones web antes de ser trasladadas al siguiente componente anidado (<Host> en este caso):

```

name="Catalina" defaultHost="localhost">

    className="org.apache.catalina.valves.AccessLogValve" directory="logs"
        prefix="catalina_access_log" suffix=".txt"
        pattern="%h %l %u %t "%r" %s %b" />

    ...

```

```

name="localhost" appBase="webapps" unpackWARs="true" autoDeploy="false">
    ...

```

Una válvula puede ponerse en tres sitios:

- ✓ Dentro de un Engine.
- ✓ Dentro de un Host.
- ✓ Dentro del contexto de una aplicación web, llevándose a cabo el procesamiento requerido antes de acceder al contenido o al componente web correspondiente.

Revisando el ejemplo anterior, podemos ver que Valve tiene varios atributos que lo configuran, vamos a echarles un vistazo:

- ✓ El atributo className es obligatorio e indica la válvula a utilizar. Para implementar un registro de acceso necesitamos obligatoriamente la válvula org.apache.catalina.valves.AccessLogValve.
- ✓ El atributo directory indica la carpeta en la que se guardarán los registros de acceso. Como es una ruta relativa ("logs"), Tomcat entenderá que se está haciendo referencia al directorio logs situado en la carpeta base de la instalación de Tomcat.
- ✓ El atributo prefix indica como comenzará el nombre del archivo donde se guardarán los logs (en este caso "catalina_access_log"), después de este texto, Tomcat añadirá automáticamente la fecha de creación del archivo. Este archivo no hay que crearlo manualmente, será Tomcat quien se encargará de ello.
- ✓ El atributo suffix indica la extensión que Tomcat debe poner al archivo de log. En este caso, será .txt, si no se indica este atributo, el archivo no tendrá extensión. Sea cual sea la extensión, su contenido siempre será texto sin formato.
- ✓ El atributo pattern indica como será el formato de cada evento registrado. Tomcat sustituirá los identificadores que comienzan por '%', por el valor correspondiente, por ejemplo:
 - ⇒ %h será sustituido por el nombre del equipo remoto o su dirección IP.
 - ⇒ %u será sustituido por el usuario remoto, si hubo un proceso de autenticación previa.
 - ⇒ %t será sustituido por la fecha y la hora.
 - ⇒ %r será sustituido por la primera línea de la petición HTTP.
 - ⇒ %s será sustituido por el código de estado HTTP de la respuesta generada (200 por ejemplo)
 - ⇒ %b será sustituido por la cantidad de bytes enviados al cliente, sin incluir la información del protocolo HTTP en si mismo.

Un ejemplo de log generado por Tomcat puede ser el siguiente:

...@elUbuntu-OptiPlex-5090:~/Documentos\$ java -Djava.util.logging.config.file=logging.properties -jar tomcat-examples-9.0.43.jar

127.0.0.1 - - [07/Nov/2018:18:27:26 +0000] "GET /examples/ HTTP/1.1" 200 1010

127.0.0.1 - - [07/Nov/2018:18:27:26 +0000] "GET /examples/ HTTP/1.1" 200 1010

127.0.0.1 - - [07/Nov/2018:18:27:26 +0000] "GET /examples/ HTTP/1.1" 200 1010

127.0.0.1 - - [07/Nov/2018:18:27:26 +0000] "GET /examples/ HTTP/1.1" 200 1010

Debes conocer

En el siguiente enlace podrás encontrar el resto de atributos disponibles a la hora de configurar un nuevo Valve de registro de acceso, incluidos los diferentes identificadores que puedes usar en el patrón (pattern) con el que se registrará cada evento en el log. Además, encontrarás diferentes ejemplos de uso:

 [Atributos de una válvula tipo AccessLogValve.](#)

Es normal que el registro de acceso se configure de forma exclusiva para una aplicación web, es decir, que registre los accesos solo de una aplicación web. Para realizar esto debemos configurar el contexto (Context) de la aplicación.

Para ello podemos añadir la información de contexto de diferentes formas, pero la forma más conveniente es creando un archivo context.xml dentro de la carpeta META-INF de la aplicación web (META-INF/context.xml). La idea sería meter una válvula dentro de las etiquetas `<Context>/</Context>`

```
...@elUbuntu-OptiPlex-5090:~/Documentos$ cat META-INF/context.xml
<Context>
    <Valve
        className="org.apache.catalina.valves.AccessLogValve"
        directory="logs"
        prefix="ejemplo_access_log"
        suffix=".txt"
        pattern="%h %l %u %t \"%r\" %s %b" />
</Context>
```

Recuerda no confundir la carpeta META-INF con la carpeta WEB-INF.

Autoevaluación

¿Dónde no se podría poner un Valve?

- Directamente dentro de un <Host>.
- Directamente dentro de un <Engine>.
- Directamente dentro de un <Connector>.
- En el contexto de una aplicación web concreta.

3.4.3.- Sesiones persistentes.

Caso práctico

Después de analizar los registros de acceso, **Ada** está preocupada sobre el rendimiento de algunas aplicaciones web que se van a ejecutar en el servidor de aplicaciones, en especial aquellas que hacen un uso intensivo de sesiones, y que tienen mucha demanda, es decir, que tienen muchos usuarios. En este tipo de aplicaciones, el servidor de aplicaciones debe gestionar las sesiones de forma eficiente, o el rendimiento caerá empicado al aumentar el número de usuarios. Como sabe que **Juan** y **María** están enfrascados en la instalación y configuración de los servidores Tomcat, les habla sobre el tema:

- Necesito que busquéis la forma de mejorar la gestión de las sesiones en Tomcat, vamos a tener una aplicación web que es posible que sea usada por en torno a 100 usuarios simultáneamente, con gran cantidad de información de sesión - comenta **Ada**.
- Vale **Ada**, nos ponemos con ello -comenta **María**-, justamente estaba leyendo sobre ello.
- Perfecto -dice **Ada**-, lo dejo en vuestras manos.

Juan y **María** no saben muy bien que es una sesión, y **María** simplemente ha empezado a leer del tema, así que, a pesar de haberle dicho a **Ada** que se harían cargo, estaban un poco preocupados. ¿Qué van a hacer?



(CC) BY-NC-SA boltron

Y tú, ¿sabes que es una sesión? Una sesión es un conjunto de información que la aplicación web almacena para cada cliente. Por ejemplo, la información de sesión puede incluir:

- ✓ Productos incluidos en el carrito de la compra por un cliente en una aplicación web de venta on-line.
- ✓ Datos introducidos por última vez por un usuario en un formulario de una aplicación web.
- ✓ Nivel de privilegios en la aplicación web de un usuario previamente autenticado.

Las sesiones se asocian a cada cliente web (navegador) durante un periodo de tiempo concreto o durante un proceso concreto. Para ello se hace uso de las famosas  cookies. De forma abreviada podemos decir que el proceso es el siguiente:

Si quisieramos configurar las sesiones usando un gestor de sesiones más avanzado, como PersistenManager, podríamos usar una configuración como la siguiente en el archivo context.xml:

```
classNamed="org.apache.catalina.session.PersistentManager" saveOnRestart="true"
    maxActiveSession="3" minIdleSwap="10" maxIdleSwap="60" >
        className="org.apache.catalina.session.FileStore"/>
```

Analicemos lo que encontramos en el ejemplo anterior. Dentro de la etiqueta <Manager> tendremos:

- ✓ El atributo classname, que especifica el gestor de sesiones a utilizar, en este caso org.apache.catalina.session.PersistentManager.
- ✓ El atributo saveOnRestart, que indica al gestor de sesiones que se guarden las sesiones activas en disco al apagar el servidor, por defecto será true (si fuera false las sesiones no se guardarían). Para almacenar las sesiones es necesario configurar un <Store> dentro del <Manager>.
- ✓ El atributo maxActiveSession indica el número de sesiones máximas que se crearán por el gestor (-1 es ilimitado). Si el número máximo se alcanza, la aplicación web no podrá crear más sesiones y tendrá que tratar esa circunstancia. Este atributo también se puede especificar para el gestor StandardManager.
- ✓ Los atributos minIdleSwap y maxIdleSwap establecen el intervalo de tiempo mínimo y máximo que hay que esperar, cuando una sesión se convierte en inactiva, para almacenarla de forma persistente. Por defecto, estas opciones están deshabilitadas.
- ✓ Por último, el componente anidado <Store> permite indicar dónde y cómo almacenar las sesión activas pero que no están en uso. Para este componente tenemos dos implementaciones posibles. La implementación a utilizar se indicaría con el atributo className: org.apache.catalina.session.FileStore o org.apache.catalina.session.JDBCStore. La implementación FileStore está destinada a almacenar las sesiones en disco, mientras que la implementación JDBCStore en una base de datos. Esta última implementación requiere configuración extra para indicar la conexión a la base de datos a utilizar.

Aunque no es recomendable, es posible configurar las sesiones persistentes de forma global. Para ello tendremos que manipular el archivo conf/context.xml, de la carpeta de configuración de Tomcat.

Para saber más

En la siguiente página podrás encontrar todo el conjunto de opciones disponibles para configurar los dos tipos de gestores de sesiones que incluye Tomcat.

 [El componente Manager de Tomcat \(en inglés\)](#)

Citas para pensar

“

Los sistemas nuevos generan problemas nuevos.

Ley de Murphy

3.4.4.- Autenticación de usuarios.

Es normal hoy día que muchas aplicaciones web necesiten de un control de acceso, para evitar accesos no autorizados al servicio. El equipo de desarrollo, en base a unas especificaciones previamente dadas, establecerá las necesidades de autenticación de la aplicación web. Para ello, la especificación Servlet establece un mecanismo estándar que permite, a través del descriptor de despliegue (WEB-INF/web.xml), indicar las necesidades de autenticación de su aplicación. Este mecanismo permite autenticar diferentes componentes de la aplicación web en lo que se denominan diferentes reinos (Realm).

Veamos un ejemplo de la configuración incluida en el descriptor de despliegue para la autenticación. En primer lugar, debemos indicar los roles que nuestra aplicación soporta, esto permitirá hacer que una parte de la aplicación sea accesible por unos roles y otras partes por otros roles diferentes:

```
<security-role>
    <description>Usuario normal de la aplicación</description>
    <role-name>usuario</role-name>
</security-role>
```

Recuerda que esta información es propia de la aplicación web (WEB-INF/web.xml). Además de la información anterior, en la que se define un único rol llamado usuario, hay que indicar como se negociará la autenticación con el navegador web. Aquí tenemos muchos métodos, vamos a escoger el método BASIC, que seguro que te suena del tema anterior:

```
<login-config>
    <auth-method>BASIC</auth-method>
    <realm-name>EjemploWebApp</realm-name>
</login-config>
```

En el caso anterior, se ha especificado cuál es el nombre del reino (<realm-name>EjemploWebApp</realm-name>), éste solamente se usa para preguntar al usuario por el usuario y la contraseña:



Salvador Romero Villegas (elaboración propia). Cuadro de dialogo donde se solicita usuario y contraseña. (Dominio público)

Por último, en el descriptor de despliegue de la aplicación web, habría que incluir que recursos están limitados por el control de acceso. Se pueden especificar solo aquellos recursos y componentes web que necesitemos, pero en este caso, vamos a limitar el acceso a toda la aplicación web:


```

<!-- Recursos globales -->
<GlobalNamingResources>
    <!-- Datos de los usuarios y sus contraseñas -->
    <Resource name="UserDatabase" auth="Container"
        type="org.apache.catalina.UserDatabase"
        description="User database that can be updated and saved"
        factory="org.apache.catalina.users.MemoryUserDatabaseFactory"
        pathname="conf/tomcat-users.xml" />
</GlobalNamingResources>

```

En el fragmento de configuración anterior se define un archivo que contendrá la información de los usuarios, que concretamente será el archivo conf/tomcat-users.xml (ubicado en la carpeta de configuración de Tomcat). Si quisieramos usar otro archivo, el mecanismo es fácil, simplemente se duplica el <Resource> anterior, cambiando la ruta (pathname) y el nombre (name), aunque también tendríamos que cambiar el nombre del recurso (resourceName) en el <Realm> anterior (declarado en el contexto de la aplicación). Ahora podemos insertar nuestros usuarios y contraseñas en el archivo conf/tomcat-users.xml de forma relativamente sencilla, teniendo en cuenta que el rol de los usuarios que tendrá nuestra aplicación es usuario:

```

<?xml version="1.0" encoding="UTF-8"?>
<tomcat-users xmlns="http://tomcat.apache.org/xml"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://tomcat.apache.org/xml tomcat-users.xsd"
    version="1.0">
    <!-- Declaración del rol -->
    <role rolename="usuario"/>
    <!-- Declaración del usuario, su contraseña y los roles a los que pertenece (separados por comas si fuese necesario) -->
    <user username="ejemplo" password="ejemplo" roles="usuario"/>
</tomcat-users>

```

Cuando cambiamos este archivo de usuarios, o la configuración de Tomcat, es necesario reiniciar el servidor para que cargue los cambios.

Este es uno de los mecanismos que incluye Tomcat para almacenar la información de autenticación de los usuarios, el más sencillo con diferencia. Existen otros mecanismo más complejos y eficientes, como por ejemplo, usar un servidor de bases de datos.

Normalmente, los servidores de aplicaciones basados en Java EE permiten la declaración y configuración de **dominios de seguridad**. Dominio de seguridad se entiende como una configuración de seguridad que es usada por una o más aplicaciones para autenticación (identificar y validar a quien usa el servicio) y autorización (determinar que permisos tiene y que acciones puede realizar). Las aplicaciones web usan por tanto este servicio del servidor de aplicaciones para manejar su información de seguridad. Es la especificación [JAAS](#) (Java Authentication and Authorization Service) la que está detrás de todo esto.

Para saber más

En el siguiente enlace encontrarás mucha más información sobre como configurar diferentes tipos de reinos (<Realm>) para autenticación de usuarios en Tomcat:

 [Realm Configuration How-To \(en inglés\)](#)

Autoevaluación

La siguiente afirmación es, ¿verdadera o falsa?

El elemento de <security-constraint> se indica al integrar la aplicación web en Tomcat.

- Verdadero Falso

3.5.- Integración de Tomcat con Apache.

Caso práctico

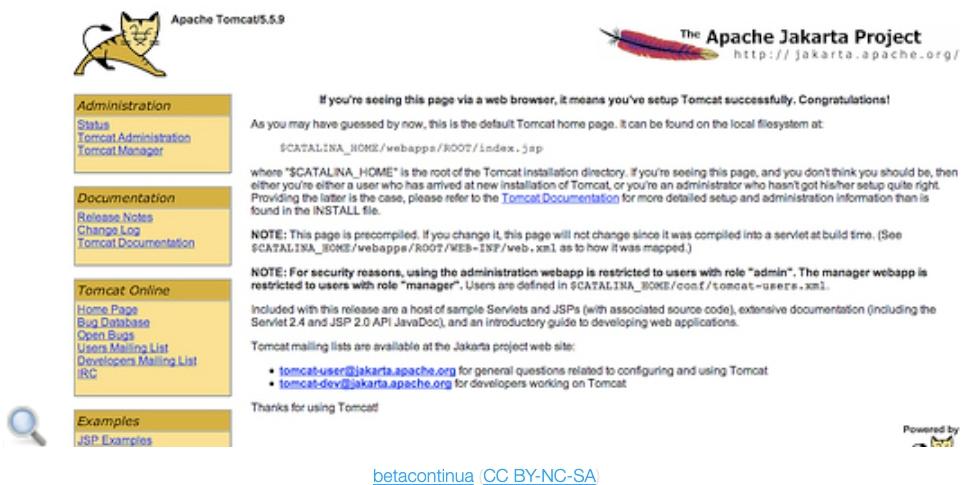
Ada acaba de recibir una buena noticia por parte de uno de sus equipos de trabajo. ¡¡Han terminado un proyecto antes de tiempo!! Se trata de una aplicación web basada en Java EE que se tiene que integrar con un servidor web Apache ya existente. En ese servidor web Apache se están ejecutando otras aplicaciones web basadas en PHP de la misma empresa, y la empresa quieren que todas sean accesibles desde el mismo servidor web, no quieren tener que acceder a través de un nombre de equipo DNS diferente o a través de un puerto diferente.

Para resolver este problema **Ada** pregunta a **Maria** y **Juan**, y ambos se ponen a investigar como realizar la integración, aunque no saben si será posible.



 br1dotcom

Y tú, ¿crees que será posible? Para realizar esta difícil tarea, el servidor web Apache deberá reenviar las peticiones web asociadas a una determinada ruta (por ejemplo, la aplicación), al servidor Tomcat que ejecuta la aplicación web. Esto deberá hacerse de forma transparente al usuario, que tendrá la sensación de que está accediendo a un único servidor.



If you're seeing this page via a web browser, it means you've setup Tomcat successfully. Congratulations!

As you may have guessed by now, this is the default Tomcat home page. It can be found on the local filesystem at: \${CATALINA_HOME}/webapps/ROOT/index.jsp

where "CATALINA_HOME" is the root of the Tomcat installation directory. If you're seeing this page, and you don't think you should be, then either you're either a user who has arrived at new installation of Tomcat, or you're an administrator who hasn't got his/her setup quite right. Providing the latter is the case, please refer to the [Tomcat Documentation](#) for more detailed setup and administration information than is found in the INSTALL file.

NOTE: This page is precompiled. If you change it, this page will not change since it was compiled into a servlet at build time. (See \${CATALINA_HOME}/webapps/ROOT/META-INF/web.xml as to how it was mapped.)

NOTE: For security reasons, using the administration webapp is restricted to users with role "admin". The manager webapp is restricted to users with role "manager". Users are defined in \${CATALINA_HOME}/conf/tomcat-users.xml.

Included with this release are a host of sample Servlets and JSPs (with associated source code), extensive documentation (including the Servlet 2.4 and JSP 2.0 API JavaDoc), and introductory guide to developing web applications.

Tomcat mailing lists are available at the Jakarta project web site:

- tomcat-user@jakarta.apache.org for general questions related to configuring and using Tomcat
- tomcat-dev@jakarta.apache.org for developers working on Tomcat

Thanks for using Tomcat!

Powered by 

Para realizar esta comunicación entre el servidor web Apache y Tomcat de forma transparente, podemos emplear el protocolo AJP (Apache JServ Protocol), cuyo objetivo es ese precisamente.

En primer lugar, vamos a describir el escenario:

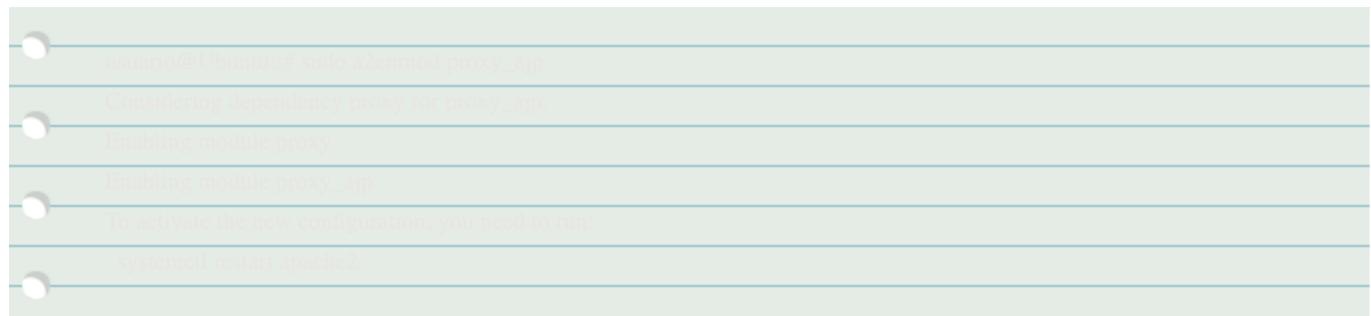
- ✓ Contamos con un servidor web Apache escuchando en el puerto 80.
- ✓ En la misma máquina, o en otra diferentes, contamos con un servidor Tomcat escuchando en otro puerto, por ejemplo, el puerto 8080.
- ✓ Ambas máquinas tienen que comunicarse usando el protocolo AJP.

El primer paso será configurar el servidor Tomcat para aceptar conexiones a través del protocolo AJP. Para ello, debemos habilitar un conector de tipo AJP en el servicio (<Service>) del que dependa nuestra aplicación web:

```
...
<Service name="Catalina">
...
<Connector port="8009" protocol="AJP/1.3" redirectPort="8443" />
...
</Service>
```

Después de realizar esta modificación en la configuración, **no olvides reiniciar el servidor Tomcat**. En la configuración anterior, la configuración port="8009" establecerá que el puerto donde se escucharán peticiones usando el protocolo AJP (protocol="AJP/1.3") será el 8009.

Con esto ya tenemos completada la primera etapa. La segunda etapa es configurar el servidor web Apache. En este apartado obviaremos los detalles de instalación del servidor web Apache, dado que ese aspecto se trata en la unidad anterior. Suponiendo que ya tenemos un servidor web Apache funcionando, lo primero que debemos hacer es habilitar el módulo proxy_ajp, que será el que dará soporte a la funcionalidad necesaria:



Una vez habilitado el módulo anterior, el siguiente paso es añadir la configuración necesaria para hacer la redirección de peticiones. Para ello deberemos modificar la configuración de un <VirtualHost> del servidor web Apache, añadiendo la siguiente configuración:

```
<VirtualHost *:80>

  ServerAdmin webmaster@localhost
  DocumentRoot /var/www/html

  ErrorLog ${APACHE_LOG_DIR}/error.log
  CustomLog ${APACHE_LOG_DIR}/access.log combined

  ProxyPass "/ejemplo" "ajp://localhost:8009/ejemplo"

</VirtualHost>
```

En la línea 9 de la configuración anterior, se ha añadido la directiva ProxyPass. Con esta directiva las peticiones destinadas a la ruta /ejemplo se redireccionarán al servidor Tomcat, usando la URL `ajp://localhost:8009/ejemplo`. En esa URL se indica el protocolo (ajp), el nombre del equipo donde estará ejecutándose el servidor Tomcat (localhost), el puerto donde se atienden peticiones del protocolo AJP en Tomcat (8009) y la ruta a la aplicación web enlazada (/ejemplo). Las rutas a las aplicaciones web en Tomcat y el servidor web Apache no tienen porqué coincidir.

Entre las muchas consideraciones de seguridad aplicables a este tipo de configuraciones, debes considerar tres acciones especialmente importantes:

- ✓ No mezclar en un mismo servidor Tomcat, o por lo menos no el mismo <Host>, aplicaciones web que son directamente accesibles desde Tomcat y aplicaciones web que son accedidas a través de un servidor web vía AJP.
- ✓ Limitar las direcciones IP que pueden acceder a las aplicaciones web que están en Tomcat, de tal manera que solo pueda acceder a dicha aplicación web el servidor web Apache.
- ✓ Proteger adecuadamente el servidor web Tomcat con un cortafuegos.

Esta configuración es una configuración típica de producción, con usuarios reales, por lo que hay que tener especial cuidado.

Para saber más

Esta web documenta los pasos a seguir para montar un clúster horizontal formado por dos servidores, con una instancia de Tomcat corriendo en cada uno de ellos.

 [Clúster mediante Tomcat.](#)

Autoevaluación

¿A través de que protocolo se comunicarían el servidor web Apache con el servidor Tomcat?

- A través del protocolo HTTP pero usando el puerto 8009.
- A través del protocolo HTTPS pero usando el puerto 8443.
- A través del protocolo AJP.
- A través de la red de área local.

3.6.- Seguridad.

Caso práctico

En **BK Programación** van a tener visita. Durante unos días, aproximadamente un mes, un pequeño equipo de trabajo de una empresa colaboradora va a trabajar de forma conjunta en un proyecto con un equipo de **BK Programación**.

Ada, aunque está siguiendo de cerca dicha colaboración, no quiere que el equipo de trabajo visitante tenga acceso a los otros proyectos que la empresa tiene en marcha. Para ello, necesita que **Juan** y **María** estudien qué medidas de seguridad se pueden implementar en el servidor Tomcat para limitar el acceso a esos otros proyectos. **Ada** les explica la situación y ambos se ponen manos a la obra.



¿Y cómo podríamos añadir más seguridad a una aplicación web? ¿Y al servidor Tomcat? Cuando hablamos de hacer una aplicación web más segura, se suelen utilizar diferentes técnicas. Veamos algunos ejemplos:

- ✓ Limitar el conjunto de usuarios que pueden conectarse a la aplicación. Esto ya se vio con anterioridad en la autenticación de usuarios.
- ✓ Limitar el conjunto de equipos que pueden conectarse a la aplicación. Por ejemplo, podríamos configurar una aplicación web para que solo fuese usada por un conjunto limitado de equipos con un rango de direcciones IP concretas.
- ✓ Usar una conexión segura ([HTTPS](#)) para acceder al servicio web. Esto implica instalar y configurar [SSL/TLS](#) en Tomcat.

Para limitar el conjunto de equipos que pueden conectarse a una aplicación web, podemos usar las válvulas. Las válvulas ya se explicaron con anterioridad, y como ya se comentó, pueden ubicarse en diferentes partes de la configuración, dependiendo del ámbito donde quieran aplicarse, aunque lo recomendado es usar el archivo META-INF/context.xml propio de la aplicación web.

Veamos que válvulas podemos usar para este caso:

- ✓ **Remote Address Filter:** permite limitar o permitir el acceso solo a un conjunto de direcciones IP. Para ello compara la dirección IP del cliente con una o más expresiones regulares y, como resultado de ello, denegar o bien permitir la solicitud presentada por el cliente. Un ejemplo de uso podría ser el siguiente:

```
<Valve className="org.apache.catalina.valves.RemoteAddrValve" deny="127.*">
```

En el caso anterior a los clientes que tengan una IP que comienza por 127 se les va a denegar la solicitud. Si quisieramos que solo se permitieran ciertas direcciones IP, deberíamos usar una configuración como la siguiente:

```
<Valve className="org.apache.catalina.valves.RemoteAddrValve" allow="192\168.*">
```

En este caso, solo se permitiría iniciar una comunicación con el servidor de aplicaciones a aquellos equipos cuya dirección IP empiece por 192.168.

- ✓ **Remote Host Filter:** es muy parecido al anterior pero con la diferencia que permite comparar por nombre de equipo en lugar de IP.

```
<Valve className="org.apache.catalina.valves.RemoteHostValve" deny="pc_fp.*">
```

En este caso solo los equipos cuyo nombre DNS empiece por pc_fp serán permitidos.

Para saber más

En el siguiente enlace puedes encontrar de forma detallada como configurar Tomcat para usar el protocolo HTTPS. Usar el protocolo HTTPS es, hoy en día, casi obligatorio para todas las aplicaciones web cuando están en producción (cuando están siendo usadas por usuarios reales):

 [Como configurar SSL/TLS en Tomcat \(en inglés\).](#)

Para saber más

A la hora de poner un servidor Tomcat en producción, hay una serie de medidas de seguridad que debes revisar. Estas medidas no se suelen tener tan en cuenta cuando se trata de un servidor de pruebas o de desarrollo. En el siguiente enlace puedes encontrar más información.

 [Consideraciones de seguridad para Tomcat.](#)

4.- El gestor de aplicaciones Web de Tomcat.

Caso práctico

En la empresa **BK programación** han puesto en marcha un servidor de aplicaciones secundario basado en Tomcat. La idea es que se use en el proyecto conjunto que tienen con el equipo de trabajo visitante.

Por ello, **Ada** ha pedido a **Juan** y **María** que expliquen al equipo de trabajo visitante como usar el gestor de aplicaciones de Tomcat para subir la aplicación web que están desarrollando.

María se encargará de configurar un usuario para poder acceder y **Juan** va a preparar una minipresentación para explicarles como funciona.



Elaboración Propia

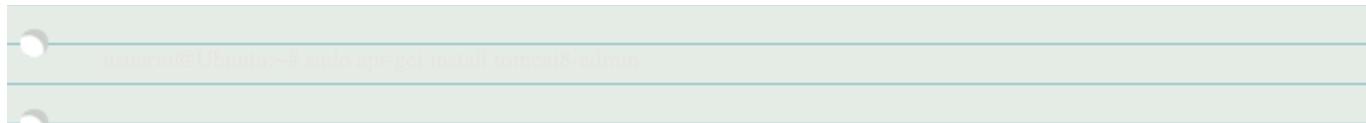
¿Qué ventajas crees que aporta usar un gestor de aplicaciones? Tomcat incorpora una aplicación web que facilita la gestión de aplicaciones de forma remota. El gestor de aplicaciones de Tomcat tiene una interfaz de usuario Web, lo cual permite su utilización desde cualquier lugar. Este gestor permite:

- ✓ Desplegar una aplicación web de diferentes formas: por ejemplo, subiendo un archivo WAR directamente.
- ✓ Replegar una aplicación web, que es básicamente, desinstalar la aplicación web y borrar su contenido del servidor.
- ✓ Ver la lista de aplicaciones desplegadas, detenerlas y reanudarlas si es necesario.
- ✓ Recargar una aplicación web para reflejar cambios en la misma (por ejemplo, si se ha modificado la versión de una librería que utiliza).



¿Sabes cómo instalar el gestor de aplicaciones de Tomcat? Si instalaste Tomcat en Ubuntu u otra distribución de Linux de forma manual, el gestor de aplicaciones ya va incluido en la instalación y no hay que hacer nada adicional para instalarlo.

Por otro lado, si instalaste Tomcat en Ubuntu desde el repositorio de aplicaciones, solo tienes que instalar un paquete adicional, y si has seguido el procedimiento descrito en el apartado 3.1.1, dicho paquete ya se instaló. No obstante, si necesitaras instalarlo porque en la etapa anterior no lo instalaste, basta con que ejecutes el siguiente comando:



No olvides reiniciar el servidor Tomcat después de instalar el administrador de aplicaciones.

¿Sabes cómo se accede al gestor de aplicaciones? Una vez arrancado **Tomcat** en el equipo servidor, podremos acceder al gestor de aplicaciones mediante la URL siguiente:

- ✓ <http://localhost:8080/manager> si accedemos desde la propia máquina en la que está corriendo Tomcat.
- ✓ http://ip_servidor:8080/manager si accedemos desde cualquier otra máquina de la red, donde ip_servidor es una dirección IP del servidor.

Si intentas acceder ahora al gestor de aplicaciones, seguramente encontrarás que te pregunta por un usuario y una contraseña. ¿Cuál es el usuario y la contraseña? En el siguiente apartado se responderá esta pregunta.

Reflexiona

Si estás trabajando como administrador de sistemas en una empresa (supongamos que es **BK programación**), en la que eres el encargado de administrar, entre otras, un máquina en la que hay un servidor de aplicaciones web Tomcat.

¿Cómo solicitarías a los desarrolladores de aplicaciones que te enviasen las aplicaciones a desplegar en dicho servidor?

4.1.- Configuración del gestor de aplicaciones.

Caso práctico

Lo primero que tiene que hacer **María** es crear un nuevo usuario para que el jefe de equipo del equipo visitante pueda acceder al gestor de aplicaciones. Pero antes de hacerlo, quiere revisar toda la documentación para ver que medidas de seguridad tiene que tomar, ¡**Ada** es muy exigente! Si no lo hace ahora, sabe que **Ada** se lo va a pedir después.



[Christiaan Colen, Acceso restringido. \(CC BY-SA\)](#)

¿Cómo se puede acceder al gestor de aplicaciones?

El gestor de aplicaciones de Tomcat va protegido por usuario y contraseña, pero no incluye ningún usuario por defecto. Esto quiere decir que debemos configurar nuestro propio usuario que pueda acceder al gestor de Tomcat.

El gestor de aplicaciones va configurado para soportar varios tipos de roles de usuarios:

- ✓ manager-gui: los usuarios con el rol manager-gui pueden acceder a la interfaz web del gestor de aplicaciones de forma completa y pueden realizar todas las tareas administrativas.
- ✓ manager-script: los usuarios con el rol manager-script podrán acceder a una interfaz modo texto pensada para tareas de despliegue automatizadas.
- ✓ manager-status: los usuarios con el rol manager-status solo podrán acceder a la página de estado del gestor de aplicaciones.

El gestor de aplicaciones web está configurado para usar el archivo conf/tomcat-users.xml de la carpeta de configuración de Tomcat, de forma similar a como se describió en el apartado anterior 3.4.4. Esto quiere decir que para configurar los usuarios que pueden usar el gestor de aplicaciones simplemente debemos añadirlos en dicho archivo:



Elaboración Propia

```

<?xml version="1.0" encoding="UTF-8"?>
<tomcat-users xmlns="http://tomcat.apache.org/xml"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://tomcat.apache.org/xml tomcat-users.xsd"
  version="1.0">

<user username="admin" password="admin" roles="manager-gui" />

</tomcat-users>

```

En el ejemplo anterior, se ha configurado un usuario con el rol manager-gui (`roles="manager-gui"`) llamado admin (`username="admin"`), cuya contraseña es admin (`password="admin"`). Después de configurarlo, necesitamos reiniciar el servidor Tomcat, y después, ya podemos acceder al administrador de aplicaciones con dicho usuario.

The screenshot shows the Tomcat Application Manager interface. At the top, there's a logo of a yellow cat and the Apache Software Foundation logo. Below that is the title "Gestor de Aplicaciones Web de Tomcat". A message box says "Mensaje: OK". The main area has tabs: "Listar Aplicaciones", "Ayuda HTML de Gestor", "Ayuda de Gestor", and "Estado de Servidor". Under "Aplicaciones", there's a table with columns: Trayectoria, Versión, Nombre a Mostrar, Ejecutándose, Sesiones, and Comandos. The table contains three rows:

Trayectoria	Versión	Nombre a Mostrar	Ejecutándose	Sesiones	Comandos
/	Ninguno especificado		true	2	<input type="button" value="Arrancar"/> <input type="button" value="Parar"/> <input type="button" value="Recargar"/> <input type="button" value="Replegar"/> <input type="button" value="Expirar sesiones sin trabajar ≥ 30 minutos"/>
/docs	Ninguno especificado	Tomcat Documentation	true	0	<input type="button" value="Arrancar"/> <input type="button" value="Parar"/> <input type="button" value="Recargar"/> <input type="button" value="Replegar"/> <input type="button" value="Expirar sesiones sin trabajar ≥ 30 minutos"/>
/ejemplo	Ninguno especificado		true	0	<input type="button" value="Arrancar"/> <input type="button" value="Parar"/> <input type="button" value="Recargar"/> <input type="button" value="Replegar"/> <input type="button" value="Expirar sesiones sin trabajar ≥ 30 minutos"/>

At the bottom left is a magnifying glass icon, and at the bottom right is a "PUBLIC DOMAIN" logo with the text "Elaboración Propia".

Tomcat también dispone de un gestor de hosts virtuales (host-manager). Para acceder a él podemos poner la siguiente URL:

- ✓ `http://localhost:8080/host-manager`: si accedemos desde el mismo equipo donde está ejecutándose el servidor Tomcat.
- ✓ `http://ip_servidor:8080/host-manager/`: si accedemos desde otro equipo de la red, donde ip_servidor es la dirección IP del servidor Tomcat.

Para tener acceso al host-manager tendríamos que crear también un usuario, pero estableciendo el rol admin-gui. Podemos usar un usuario diferente al usuario creado para el gestor de aplicaciones, o usar el mismo, si usamos el mismo simplemente añadiremos otro rol a la lista de roles:

```
<user username="admin" password="admin" roles="manager-gui,admin-gui" />
```

¿Qué precauciones adicionales crees que se deben tener con el gestor de aplicaciones? Disponer de una herramienta tan potente es fantástico y agiliza el trabajo, pero tiene sus riesgos. Es necesario proteger el acceso a la misma para que solo pueda ser usada por las personas adecuadas y desde determinados equipos.

Para limitar el acceso a dicha herramienta es conveniente configurar, en el contexto del gestor de aplicaciones, las direcciones IP que pueden acceder al gestor de aplicaciones.

La ubicación del archivo de contexto del gestor de aplicaciones varía en función de la forma de instalación. Si se instaló de forma manual, estará en la ruta webapps/ manager/META-INF/context.xml (partiendo de la carpeta base de la instalación de Tomcat), pero si se instaló desde repositorio, el archivo de contexto del gestor de aplicaciones será conf/Catalina/localhost/manager.xml (vista esta ruta desde la carpeta base de la configuración). Un ejemplo de configuración que limita el acceso por direcciones IP podría ser la siguiente:

```
<Context path="/manager"
    docBase="/usr/share/tomcat8-admin/manager"
    antiResourceLocking="false" privileged="true">
    <Valve className="org.apache.catalina.valves.RemoteAddrValve" allow="127\0\0\1|192\168\1\130" />
</Context>
```

Fíjate que para hacer esta limitación usamos un Valve explicado anteriormente. En el ejemplo anterior, se permitiría el acceso desde dos direcciones IP: 127.0.0.1 y 192.168.1.130.

Autoevaluación

De las siguientes opciones, ¿qué roles podría tener un usuario para usar el gestor de aplicaciones?

- manager-gui
- manager-script
- manager-status
- admin-gui

[Mostrar retroalimentación](#)

4.2.- Uso del gestor de aplicaciones de Tomcat.

El gestor de aplicaciones de Tomcat (Tomcat Manager) dispone de una interfaz muy sencilla de utilizar. Después de entrar en la misma, usando por ejemplo la URL <http://localhost:8080/manager>, lo primero que podemos ver es un listado de las aplicaciones desplegadas en nuestro servidor, en una tabla fácil de identificar. Veamos un ejemplo:



Aplicaciones					
Trayectoria	Versión	Nombre a Mostrar	Ejecutándose	Sesiones	Comandos
/	Ninguno especificado		true	0	<input type="button" value="Arrancar"/> <input type="button" value="Parar"/> <input type="button" value="Recargar"/> <input type="button" value="Replegar"/> <input type="button" value="Expirar sesiones"/> sin trabajar a 30 minutos
/docs	Ninguno especificado	Tomcat Documentation	true	0	<input type="button" value="Arrancar"/> <input type="button" value="Parar"/> <input type="button" value="Recargar"/> <input type="button" value="Replegar"/> <input type="button" value="Expirar sesiones"/> sin trabajar a 30 minutos
/host-manager	Ninguno especificado	Tomcat Host Manager Application	true	0	<input type="button" value="Arrancar"/> <input type="button" value="Parar"/> <input type="button" value="Recargar"/> <input type="button" value="Replegar"/> <input type="button" value="Expirar sesiones"/> sin trabajar a 30 minutos
/manager	Ninguno especificado	Tomcat Manager Application	true	1	<input type="button" value="Arrancar"/> <input type="button" value="Parar"/> <input type="button" value="Recargar"/> <input type="button" value="Replegar"/> <input type="button" value="Expirar sesiones"/> sin trabajar a 30 minutos

Lista de aplicaciones ejecutándose en Tomcat. (Dominio público)
Elaboración propia (Salvador Romero Villegas).

Cada aplicación web se muestra en una fila de la tabla, y para cada aplicación web es posible hacer las siguientes operaciones:

- ✓ Botones **Parar** y **Arrancar**: haciendo clic en el botón **Parar** la aplicación web dejará de ejecutarse en el servidor Tomcat y no será accesible. Como resultado de esta operación la columna etiquetada como "**Ejecutándose**" se pondrá a false para la aplicación en cuestión, y se activará el botón **Arrancar**. La aplicación podrá reanudarse nuevamente haciendo clic en el botón **Arrancar**.
- ✓ Botón **Recargar**: este botón elimina de memoria la aplicación web y la vuelve a cargar desde disco, activando posibles cambios que se hayan podido producir en los archivos que componen la aplicación Web.
- ✓ Botón **Replegar**: este botón detiene y desinstala la aplicación web. Si queremos volver a ejecutarla tendremos que volver a despegarla nuevamente.
- ✓ Botón **Expirar Sesiones**: este botón borra las sesiones que no hayan expirado de una determinada aplicación web. Como se puede ver en la imagen, el mismo gestor de aplicaciones tiene una sesión iniciada, que fue la sesión utilizada para realizar la captura de pantalla anterior. Al lado de este botón, hay una pequeña caja de texto que permite indicar la antigüedad, en minutos, de las sesiones a expirar.

En la misma página, bajando un poco más abajo, encontraremos una pequeña interfaz para desplegar una nueva aplicación web:



Desplegar	
Desplegar directorio o archivo WAR localizado en servidor	
Trayectoria de Contexto (opcional):	<input type="text"/>
URL de archivo de Configuración XML:	<input type="text"/>
URL de WAR o Directorio:	<input type="text"/>
<input type="button" value="Desplegar"/>	
Archivo WAR a desplegar	
Seleccione archivo WAR a cargar:	<input type="button" value="Examinar..."/> ejemplo.war
<input type="button" value="Desplegar"/>	

Elaboración propia (Salvador Romero Villegas).
Despliegue de aplicaciones web usando el gestor de aplicaciones de Tomcat (Dominio público)

El gestor de aplicaciones de Tomcat facilita dos métodos para desplegar una aplicación web:

- ✓ Desplegar una aplicación web desde un directorio del servidor. Para esto debemos subir previamente la aplicación web al servidor como archivo WAR, o en formato expandido o desempaquetado. Para realizar la subida podemos usar el protocolo FTP.
- ✓ Desplegar una aplicación web subiendo un archivo WAR a través del mismo gestor de aplicaciones.

El primer método requiere que seamos capaces de conectarnos al servidor web usando un protocolo de transferencia de archivos, y que subamos la aplicación web a un directorio del servidor concreto (por ejemplo, al directorio /var/local/webapps/). Este proceso requiere:

- ✓ Subir la aplicación al directorio designado (el directorio elegido puede ser cualquiera, pero el usuario bajo el que se ejecute el servidor Tomcat deberá tener acceso). Imaginemos para que el directorio donde hemos subido la aplicación web sea /var/local/ejemplo.
- ✓ Indicar en el gestor de aplicaciones **la trayectoria del contexto**, que permite indicar que ruta hay que poner en la URL para acceder a la aplicación web. Si deseamos, por ejemplo, que la aplicación anterior sea accesible a través de http://localhost:8080/nuevoejemplo la trayectoria de contexto debería ser /nuevoejemplo.
- ✓ Indicar en el gestor de aplicaciones la **URL del WAR o directorio** dentro del servidor que contiene la aplicación web. Imaginemos que se trata de la aplicación web en formato extendido, tendríamos que indicar el directorio de la siguiente forma: file:/var/local/ejemplo.
- ✓ Indicar, si es necesario, la **URL del archivo de configuración XML**. Este archivo contiene la información de contexto, pero no tiene que llamarse obligatoriamente context.xml. Imagina que dicho archivo lo has puesto en la ruta /var/local/ejemplo.xml, pues en ese caso habría que indicar la ruta al archivo de la siguiente forma: file:/var/local/ejemplo.xml.

The screenshot shows a deployment configuration form. It has three input fields: 'Trayectoria de Contexto (opcional)' containing '/nuevoejemplo', 'URL de archivo de Configuración XML' containing 'file:/var/local/ejemplo.xml', and 'URL de WAR o Directorio:' containing 'file:/var/local/ejemplo'. Below the fields is a 'Desplegar' button. A magnifying glass icon is positioned next to the configuration XML URL field.

Elaboración propia (Salvador Romero Villegas). *Despliegue desde el servidor.* (Dominio público)

Una vez realizado este proceso, podremos desplegar la aplicación web que se ha subido de forma manual al servidor.

Realizar el despliegue siguiendo el segundo método, subiendo una archivo .war directamente, es mucho más sencillo, dado que solo requiere elegir el archivo WAR a desplegar y hacer clic en el botón desplegar. En este caso, la ruta a la aplicación web dependerá del nombre del archivo .war. Si por ejemplo, el archivo WAR es otroejemplo.war, la ruta a la aplicación será http://localhost:8080/otroejemplo (si accedemos desde el mismo equipo donde se instaló el servidor Tomcat).

El gestor de aplicaciones web también proporciona una página para ver el estado del servidor web, esta página es accesible a través de las rutas /manager/status y /manager/status/all (en esta última se puede ver una mayor cantidad de información). Por ejemplo, accediendo desde la misma máquina en la que hemos instalado el servidor Tomcat, podríamos ver el estado del servidor a través de la URL: http://localhost:8080/manager/status.

Autoevaluación

Dada la siguiente afirmación, ¿es verdadera o falsa?

"A través del gestor de aplicaciones un usuario puede desplegar una aplicación, pero previamente hay que subirla siempre al servidor."

- Verdadero
- Falso

5.- El servidor de aplicaciones WildFly.

Caso práctico

Durante el desayuno, los empleados de **BK programación** están hablando de algo que **María** desconoce, así que se lanza a preguntarles: "¿Qué es WildFly?"

Sus compañeros y compañeras le explican que se trata de un servidor gratuito y de código abierto orientado a aplicaciones empresariales. También le comentan que es uno de los servidores de aplicaciones principales de hoy día, y que tiene, desde hace tiempo, una gran importancia en el mercado, tanto para particulares como para grandes empresas.

A **María** le despierta la curiosidad y decide que es buen momento para estudiarlo.



[Francesco Crippa](#). Conferencia sobre software libre. [\(CC BY\)](#)

Y tú, ¿has oido hablar de WildFly? El servidor WildFly es un proyecto gratuito, de código abierto y escrito en Java, que implementa un servidor de aplicaciones Java EE.

Mientras que Tomcat es un solo un servidor web y un contenedor de servlets, WildFly es un servidor de aplicaciones que implementa todas las funcionalidades de la especificación Java EE. Por ejemplo, WildFly es un servidor web, un contenedor de servlets, y también un contenedor de Enterprise JavaBeans (EJB), entre otras cosas, mientras que Tomcat no.

La historia de WildFly empezó cuando la empresa Red Hat compró la empresa JBoss, Inc., responsable del proyecto JBoss, en 2006. JBoss era por aquellos entonces un servidor de aplicaciones muy relevante en el mercado. Después de esta compra, el proyecto JBoss evolucionó hasta convertirse en JBoss AS. A finales de 2012, Red Hat decidió cambiarle el nombre a JBoss AS, realizando para ello una votación pública. De entre todos los nombres alternativos que se propusieron, WildFly fue el más votado y el nombre que tomaría este servidor de aplicaciones.

Y te preguntarás, ¿qué beneficio saca de todo esto la empresa Red Hat si WildFly es software libre? Pues existe una versión comercial, conocida como JBoss EAP, la cual se basa en WildFly y que incluye un conjunto de características mayor. Red Hat ofrece soporte para esta versión comercial.

¿Qué hace a WildFly interesante?

- ✓ Realmente arranca bastante rápido para ser un servidor de aplicaciones. Esto es aplicable tanto del servidor en sí, como de las aplicaciones que se despliegan en el mismo, en comparación con otros servidores de aplicaciones.
- ✓ Es capaz de atender gran cantidad de peticiones web simultáneamente gracias a sus capacidades de escalabilidad y su servidor web de alto rendimiento. Además, está preparado para ser integrado como un servicio en la nube, para trabajar en modo clúster y para trabajar integrado con otro servidor web.
- ✓ Optimiza el uso de la memoria y de los recursos.
- ✓ Es altamente configurable y dispone de una potente utilidad de administración (conocida como consola de administración), que permite hacer casi todo tipo de tareas.
- ✓ Implementa Java EE al completo, por lo que las aplicaciones web no necesitan configurar nada fuera de estas especificaciones. Esto hace que las aplicaciones sean más portables entre diferentes servidores de aplicaciones.
- ✓ Internamente se basa en otros proyectos de software libre muy conocidos, tales como Hibernate.

Autoevaluación

¿Cuáles de las siguientes son características del servidor de aplicaciones WildFly?

- Es de código abierto.
- Está implementado en su totalidad en Java.
- Es únicamente un contenedor de EJBs.
- Funciona únicamente en servidores Microsoft Windows.

[Mostrar retroalimentación](#)

5.1.- Instalación y configuración básica.

La variedad de plataformas donde se puede instalar WildFly es enorme. En este caso vamos a describir como se realizaría la instalación en la distribución de Linux Ubuntu (concretamente la versión 17.04), aunque el proceso es similar en otras distribuciones de Linux basadas en Debian.

Para instalar WildFly, al igual que ocurre en Tomcat, es imprescindible instalar el JDK. Sea cual sea el JDK que instalemos, es importante que la versión del JDK sea la adecuada para la versión de WildFly que vamos a instalar. Por ejemplo, WildFly 13 necesita la versión 8 del JDK. Para descargar WildFly puedes utilizar la página oficial, que es la siguiente:

 [Descargar WildFly de la página oficial.](#)

El primer paso es instalar el JDK. El proceso de instalación del JDK es el mismo que el que se explicó en apartados anteriores, puedes seguir el procedimiento del apartado 3.1.1 o el del 3.1.2, así que vamos a obviar esa parte. Sea cual sea el proceso de instalación que hayas seguido (manual o desde repositorio), no olvides tener en cuenta la versión instalada del mismo.

En segundo lugar, vamos a instalar WildFly. Para realizar esta instalación vamos a optar por hacer una instalación típica de un servidor de producción, donde no es conveniente que el software de servidor se ejecute con un usuario con permisos de administración, como puede ser root. Si existe algún problema en el software, este puede causar verdaderos estragos en todo el sistema, mientras que si se ejecuta con un usuario diferente, el problema queda aislado.

Para proceder a la instalación vamos a copiar el archivo comprimido con la distribución de WildFly en la carpeta donde queramos instalarlo (en el ejemplo será la carpeta / opt), y después, lo descomprimiremos. El proceso de descompresión podría ser como el siguiente:

root@Ubuntu:~\$ tar -xvf wildfly-13.0.0.Final.tar.gz

En el caso del ejemplo anterior, la carpeta con WildFly será /opt/wildfly-13.0.0.Final, aunque esto variará obviamente en función de la versión elegida. Normalmente, lo que hace a continuación (no es obligatorio), es crear un enlace simbólico de la carpeta anterior a la carpeta /opt/wildfly, de tal manera que se pueda acceder más cómodamente a WildFly:

root@Ubuntu:~\$ ln -s /opt/wildfly-13.0.0.Final /opt/wildfly

En tercer lugar, vamos a crear un usuario que será el que ejecutará WildFly. Para crear el usuario (lo llamaremos wildfly en este caso), podemos ejecutar el comando adduser. Veamos un ejemplo:

root@Ubuntu:~\$ adduser --disabled-login --create-home --password 'XXXXXXXXXX' wildfly



 clarkbw

Las opciones aplicadas al comando adduser anterior, crean un usuario que básicamente estará destinado a ejecutar WildFly y no podrá hacer ninguna otra cosa (no se podrá iniciar sesión con dicho usuario, y por lo tanto, no tiene sentido que tenga un directorio con sus archivos). Después, habría que cambiar el propietario de la carpeta que contiene WildFly, para que dicha carpeta pertenezca al usuario antes creado (wildfly en este caso). Esto se puede hacer con el comando chmod:

```
usuario@Ubuntu:~$ sudo chown wildfly:wildfly /opt/wildfly-10.0.0.Final -recursivo -R
```

En cuarto lugar, tendremos que configurar las variables de entorno. Para realizar esto podemos seguir el procedimiento descrito en el punto 3.1.2, siguiendo un modelo similar al que se explicó para Tomcat. Podemos modificar el archivo /etc/environment, o bien el archivo, /etc/profile, aunque lo recomendado es crear nuestro propio script (llamado por ejemplo wildfly.sh) en la carpeta /etc/profile.d/. Recuerda que este proceso es específico de Ubuntu y de otras distribuciones basadas en Debian. Un ejemplo de como podría ser el script wildfly.sh sería el siguiente:

```
#!/bin/sh
export WILDFLY_HOME=/opt/wildfly-10.0.0.Final
export PATH=$PATH:$WILDFLY_HOME/bin
```

En el código anterior crearía una variable de entorno llamada WILDFLY_HOME (que contiene la ruta al servidor WildFly) y modificaría la ruta la variable PATH.

En quinto lugar, podríamos ejecutar nuestro servidor WildFly. Ahora ya podemos ejecutar de forma manual nuestro servidor WildFly de la siguiente forma:

```
root@Ubuntu:~$ . /etc/profile.d/wildfly.sh
root@Ubuntu:~$ sudo -u wildfly standalone.sh
```

Con el comando anterior estamos ejecutando el comando standalone.sh (que inicia WildFly como servidor independiente) con el usuario wildfly (-u wildfly). La opción -i, usada con anterioridad, simplemente hace que se mantengan las variables de sistema al ejecutar el comando con un nivel de privilegios mayor (comando sudo).

Ahora, si consultamos la URL <http://localhost:8080/> desde el equipo donde hemos hecho la instalación podremos acceder a nuestro WildFly.

En sexto lugar, de forma opcional, podemos instalar WildFly como un servicio. Aunque la instalación anterior es perfectamente válida, instalar WildFly como un servicio es muy conveniente, puesto que permite que WildFly se arranque de forma automática cuando se inicie el servidor.

La idea es la siguiente, WildFly proporciona unos scripts para este objetivo. Suelen tener el nombre de wildfly-init-<distribucion>.sh donde <distribucion> sería el nombre de la distribución de Linux donde se podrían usar. Además, estos scripts van acompañados de un archivo de configuración, que también tendremos que poner en nuestro sistema (llamado wildfly.conf). Veamos como podríamos localizarlos:

```
root@Ubuntu-5: sudo cp ./init/wildfly-13.0.0.Final/docs/contrib/scripts/init.d/wildfly-init-debian.sh /etc/init.d/wildfly
```

En el caso de ejemplo, los archivos que nos convendrían son el `./docs/contrib/scripts/init.d/wildfly-init-debian.sh`, dado que Ubuntu se basa en Debian, y el archivo `./docs/contrib/scripts/init.d/wildfly.conf`. El proceso ahora sería copiar el archivo `wildfly-init-debian.sh` a la carpeta `/etc/profile.d/init.d` con el nombre `wildfly`, y el archivo `wildfly.conf` a la carpeta `/etc/default` con el nombre `wildfly`. Un ejemplo podría ser el siguiente:

```
root@Ubuntu-5: sudo cp ./init/wildfly-13.0.0.Final/docs/contrib/scripts/init.d/wildfly-init-debian.sh /etc/init.d/wildfly
```

Después, habría que configurar el archivo `/etc/default/wildfly` indicando la información necesaria para ejecutar WildFly. Un ejemplo de esta configuración podría ser el siguiente:

```
## Location of JDK
JAVA_HOME=/usr/lib/jvm/java-1.7.0

## Location of Wildfly
JBoss_Home=/opt/wildfly-13.0.0.Final

## The username who should own the process
JBoss_User=wildfly

## The mode Wildfly should start, standalone or domain
JBoss_Mode=standalone

## Configuration for standalone mode
JBoss_Config=standalone.xml
```

En la configuración anterior se establece:

- ✓ La carpeta base de la instalación del JDK (`JAVA_HOME`).
- ✓ La carpeta base de la instalación de WildFly (`JBoss_Home`).
- ✓ El nombre de usuario con el que se ejecutará WildFly (`JBoss_User`).
- ✓ El modo de ejecución (`JBoss_Mode`) y el archivo de configuración para dicho modo de ejecución (`JBoss_Config`).

Una vez realizado esto, ya solamente queda decirle a Ubuntu que instale el script de arranque de WildFly:

```
root@Ubuntu-5: sudo update-rc.d wildfly defaults
```

Al estar instalado como servicio, WildFly se iniciará al arrancar el sistema, pero además:

- ✓ Podremos arrancarlo ejecutando:
 - ↳ sudo service wildfly start
- ✓ Podremos detenerlo ejecutando:
 - ↳ sudo service wildfly stop
- ✓ Podremos saber su estado ejecutando:
 - ↳ sudo service wildfly status

Recuerda que podremos acceder vía web al servidor WildFly usando la URL <http://localhost:8080/> desde el equipo donde se hizo la instalación.

Para saber más

Si quieres profundizar en la administración de WildFly, en el siguiente enlace puedes encontrar una completa documentación:

 [Documentación de administración de WildFly.](#)

5.2.- Aplicaciones empresariales.

Caso práctico

María ha realizado una instalación de prueba de WildFly, y ha visto que, a diferencia de Tomcat, en éste se pueden instalar aplicaciones empresariales.

Después de ojear un poco por Internet, no termina de tener claro que es una aplicación empresarial y que diferencia hay con una aplicación web. Decide preguntarle a **Juan** a ver si él sabe algo.



Como ya se comentó con anterioridad, las aplicaciones basadas en Java EE son aplicaciones multicapa, donde cada capa atiende una serie de necesidades u objetivos de la aplicación. Vamos a recordar rápidamente las capas con las que un servidor de aplicaciones tiene vinculación:

- ✓ **Capa web.** En esta capa están los componentes web, destinada a producir una aplicación cuya interfaz con el usuario es a través de la web (usando HTML, CSS, etc.)
- ✓ **Capa de negocios.** Los componentes de esta capa son principalmente los **Enterprise JavaBeans (EJB)** y se encargaría de implementar la lógica de negocio, es decir, como se comporta internamente nuestra aplicación.
- ✓ **Capa del sistema de información de la empresa.** Esta capa no se implementa por el servidor de aplicaciones, pero el servidor de aplicaciones debe facilitar la integración de diferentes fuentes de datos con las aplicaciones desplegadas.



ポート Fábrica. CC BY.

Hasta ahora, cuando hemos hecho referencia a una aplicación Java EE, hemos pensado en ella como una aplicación solo de la capa web, ignorando las otras capas. En el punto 2.2 se exploró como era el formato interno de una aplicación web, dando lugar al formato de aplicación web extendido o desempaquetado y al formato WAR.

Pero, ¿cómo es una aplicación que trabaja en la capa de negocio? Un equipo de desarrollo que implemente la lógica de negocio, implementaría uno o varios componentes EJB que se desplegarían también en un servidor de aplicaciones. Otros componentes, como por ejemplo un servlet, se podría comunicar con los EJBs para completar parte del proceso que realiza. Por ejemplo:

- ✓ Un EJB podría implementar el proceso de incrementar o reducir el stock en un almacén, modificando los datos de una base de datos.
- ✓ Un componente web, un servlet por ejemplo, cuando recibe la información del cliente, podría utilizar el EJB anterior para alterar el stock del almacén. De esta forma el servlet se centraría en la interacción con el usuario, y no en modificar los datos de la base de datos.

Los EJBs al final son clases Java, y estas se compilan y se empaquetan en un archivo JAR. Dependiendo de la versión de Java EE usada y del servidor de aplicaciones usado, los EJBs pueden ser incluso integrados en una aplicación web (archivo WAR). En este escenario los archivos .jar con los EJBs se situarían dentro de la carpeta WEB-INF/lib de la aplicación web.

El escenario anterior es el escenario más simple posiblemente. Para escenarios más complejos, donde el grado de reutilización es mayor (por ejemplo, es necesario reutilizar varios EJBs desde diferentes aplicaciones web), la forma de empaquetar aplicaciones suele ser diferente. En este caso se utiliza el empaquetado EAR.

Un paquete EAR encapsula una aplicación empresarial y está formado por un conjunto de módulos. Cada módulo puede ser:

- ✓ Una aplicación web, empaquetados como un archivo WAR.
- ✓ Un conjunto de EJBs, empaquetados en un archivo JAR.

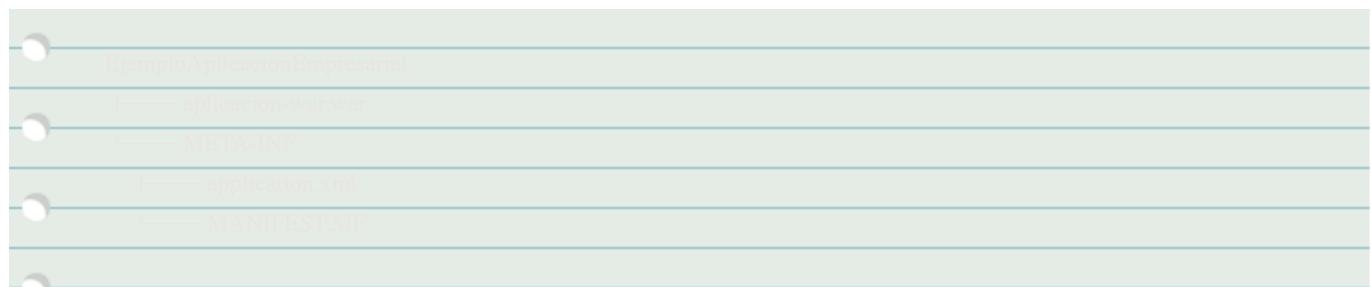
Esto quiere decir que dentro de una aplicación web puede haber uno o más módulos (incluso del mismo tipo), cada uno realizando una función diferente. Gobernando toda la aplicación empresarial, existirá un descriptor de despliegue opcional en la localización en META-INF/application.xml, con información para poner en marcha la aplicación, y los diferentes módulos que la forman.

Veamos un pequeño ejemplo de puesta en práctica de todo esto. Vamos a suponer que tenemos una aplicación web como la siguiente, realmente muy sencilla, la cual está empaquetada en el archivo aplicacion-war.war:



En la estructura de directorios anterior podemos observar una página estática llamada index.html y el descriptor de despliegue web.xml. Partiendo de esta aplicación web, vamos a construir un archivo EAR que contendrá un solo modulo, el archivo aplicacion-war.war.

Para construir el archivo EAR, vamos a crear un directorio llamado EjemploAplicacionEmpresarial y dentro pondremos el archivo .war anterior, el descriptor de despliegue application.xml en la carpeta META-INF:



Dentro del archivo opcional application.xml podemos encontrar información de despliegue de toda la aplicación empresarial, como por ejemplo, la ruta con la que se podrá acceder a la aplicación web aplicacion-war.war. Veamos, a modo ilustrativo, como podría ser este archivo application.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<application version="7" xmlns="http://xmlns.jcp.org/xml/ns/javaee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:sch
<display-name>EnterpriseApplication1</display-name>
<module>
<web>
<web-uri>aplicacion-war.war</web-uri>
<context-root>/otroejemplo</context-root>
</web>
</module>
</application>
```

En el archivo de configuración anterior se indica que la aplicación web aplicacion-war.war se accedería a través de la ruta /otroejemplo. Para crear ahora el archivo .ear podemos ejecutar el comando jar de la siguiente forma:

ejecuta el comando jar con el argumento -C para indicar el directorio de la aplicación.

En el ejemplo anterior, ruta_a_la_aplicacion tendría que ser sustituido por la ruta completa hasta el directorio que contiene la aplicación empresarial desempaquetada (fijate que hay un punto después de la ruta, no olvides ponerlo).

Autoevaluación

¿Cuáles de las siguientes afirmaciones son correctas?

- Un archivo .war puede estar formado por varios archivos .ear.
- Un archivo .ear puede estar formado por varios archivos .war.
- El comando jar permite generar archivos .war.
- El comando **jar** permite generar archivos .ear.
- Un archivo .ear puede contener archivos .jar.

[Mostrar retroalimentación](#)

5.3.- Despliegue de aplicaciones empresariales y aplicaciones Web en WildFly.

En el punto anterior se describía como instalar WildFly, en este apartado vamos a describir como usar su interfaz web de administración para desplegar una aplicación web, y si fuese necesario, una aplicación empresarial (el procedimiento es básicamente el mismo). Para poder usar la conocida consola de administración de WildFly, debemos añadir un usuario usando el script add-user.sh incluido en WildFly.

Al crear el usuario nos preguntará que tipo de usuario deseamos añadir. Deberemos elegir un usuario de gestión (Management User), que será un usuario dentro del reino (**realm**) conocido como ManagementRealm. Veamos como podría ser el proceso:

```
salvador@salvador-VirtualBox: ~ $ sudo -u wildfly ./bin/add-user.sh
What type of user do you want to add?
 1) Management User (administrator properties)
 2) Application User (application specific properties)
 3) Other
 4) None
Please enter the realm name (wildfly):
Please enter the user type (1 or 2):
Please enter the user name (admin):
Please enter the password:
Please enter the full name (Salvador Ruiz):
Please enter the roles (admin, user, etc.):
```

En el proceso anterior, aparte de preguntarnos por el tipo de usuario, el nombre de usuario (administrador en el caso anterior) y la contraseña, se realizan varias preguntas a las que podemos responder con yes (si). Una vez creado el usuario, ya podremos acceder a la consola de administración usando la URL siguiente: <http://localhost:8080/console>; accediendo desde un navegador en el mismo equipo en el que hemos instalado WildFly. Al entrar, nos pedirá el usuario y la contraseña que hemos configurado en el paso anterior, y después, veremos una interfaz sencilla de utilizar:



[Espacio CAMON](#)

Elaboración propia (Salvador Romero Villegas). *Página principal de la consola de administración de WildFly 13.* (Dominio público)

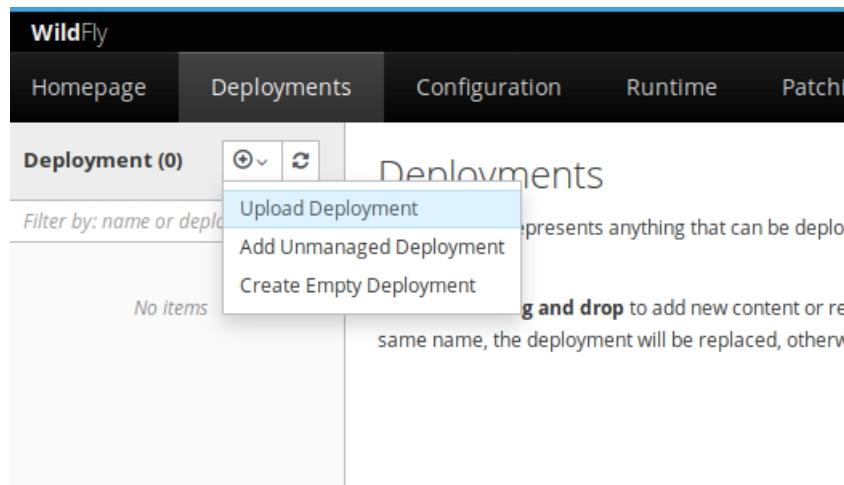
Como se puede ver en la imagen, en la consola de administración podremos llevar a cabo muchas tareas:

- ✓ Añadir y gestionar las aplicaciones desplegadas.
- ✓ Monitorizar el estado del servidor.
- ✓ Configurar los diferentes subsistemas que conforman WildFly.
- ✓ Etc.

En este caso, vamos a centrarnos en como desplegar y replegar una aplicación. Si entras en la sección "**Deployments**", podrás ver la lista de aplicaciones desplegadas:

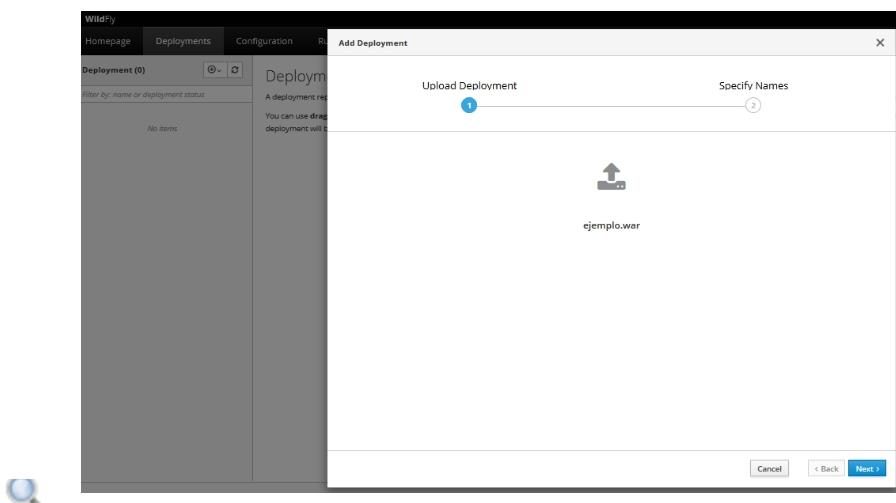
Elaboración propia (Salvador Romero Villegas). *Lista de aplicaciones desplegadas en WildFly 13.* (Dominio público)

Una vez dentro de la sección "**Deployments**", haciendo clic encima del símbolo de suma (" \pm "), podrás desplegar una aplicación:



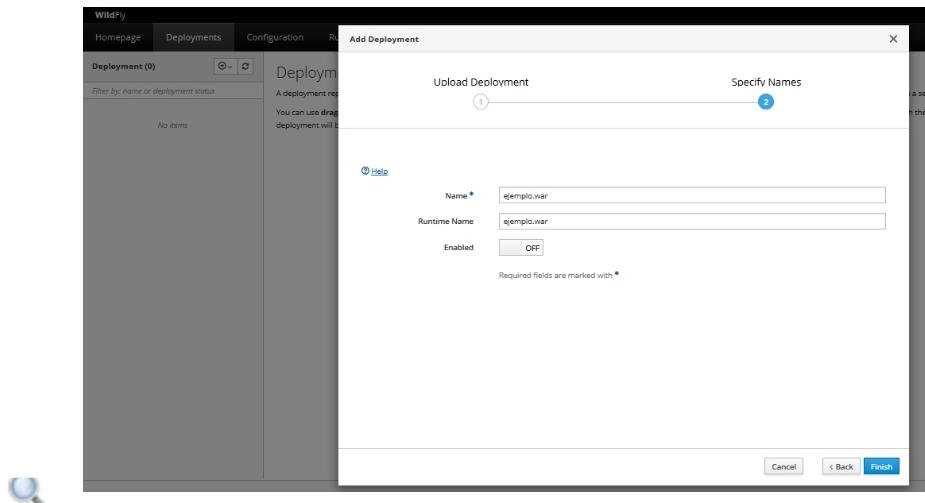
Elaboración propia (Salvador Romero Villegas). *Dónde subir una nueva aplicación para su despliegue.* (Dominio público)

Para subir una aplicación web o una aplicación empresarial (como archivo WAR o EAR), deberás seleccionar la opción "**Upload Deployment**". Después de hacer clic en esa opción, saldrá la siguiente interfaz:



Elaboración propia (Salvador Romero Villegas). *Arrastrar o seleccionar la aplicación web a desplegar.* (Dominio público)

En la interfaz anterior simplemente tendremos que arrastrar o seleccionar la aplicación web o aplicación empresarial a desplegar, y esta se subirá al servidor de aplicaciones. En este caso vamos a proceder a desplegar una aplicación web. Para continuar, haremos clic en siguiente, donde aparecerá la siguiente interfaz:



Elaboración propia (Salvador Romero Villegas). Segundo paso del proceso de despliegue en WildFly. (Dominio público)

En la interfaz anterior se puede configurar el nombre que tendrá la aplicación web en tiempo de ejecución (**Runtime Name**), lo cual permitirá indicar una ruta de acceso diferente para la aplicación web (en vez de `http://localhost:8080/ejemplo` podríamos tener `http://localhost:8080/otroejemplo` si ponemos en esta configuración `otroejemplo.war`), y también un botón que permitirá habilitar la aplicación web. Si no se selecciona la opción de habilitar aplicación, la aplicación no se ejecutará, y aparecerá en la lista como se muestra a continuación:

Main Attributes	
Name:	ejemplo.war
Runtime Name:	ejemplo.war
Enabled, Managed, Exploded:	✘ ✓ ✘
Status:	STOPPED
Last enabled at:	n/a
Last disabled at:	n/a

Elaboración propia (Salvador Romero Villegas). Aplicación web deshabilitada. (Dominio público)

Seleccionando la aplicación de la lista y haciendo clic en "**enable**", podemos habilitar la aplicación web fácilmente. Desde esta misma interfaz es relativamente sencillo volver a desplegar otra aplicación web, replegar una existente o deshabilitar temporalmente la aplicación que nos interese.

Debes conocer

WildFly es un servidor de aplicaciones mucho más complejo que Tomcat, el cuál es solo un contenedor de servlets y servidor web. Para configurar WildFly en mayor profundidad, dado el modo de instalación y ejecución aquí descrito, podemos modificar el archivo standalone/configuration/standalone.xml situado en la carpeta de instalación de WildFly. En este archivo podemos, por ejemplo, cambiar los puertos en los que WildFly escucha peticiones HTTP o las direcciones IP del servidor a través de las que atiende peticiones. Por defecto, solo atenderá peticiones desde el mismo equipo donde ha sido instalado:

```
<interfaces>
  <interface name="management">
    <inet-address value="${jboss.bind.address.management:127.0.0.1}" />
  </interface>
  <interface name="public">
    <inet-address value="${jboss.bind.address:127.0.0.1}" />
  </interface>
</interfaces>
```

Pero eso podemos modificarlo poniendo la siguiente configuración (fíjate que en vez de poner 127.0.0.1 ahora pone 0.0.0.0):

```
<interfaces>
  <interface name="management">
    <inet-address value="${jboss.bind.address.management:0.0.0.0}" />
  </interface>
  <interface name="public">
    <inet-address value="${jboss.bind.address:0.0.0.0}" />
  </interface>
</interfaces>
```

Esto nos permitirá usar el WildFly que hemos instalado desde otra máquina diferente de la red.

6.- Construcción y despliegue automático con Ant.

Caso práctico

En la empresa **BK Programación**, para agilizar el proceso de construcción de aplicaciones web, han pensado en la automatización del proceso con la ayuda de la herramienta **Ant** que se emplea para la realización de tareas mecánicas y repetitivas, normalmente durante la fase de compilación y construcción.

A la hora de implantar dicha herramienta se han propuesto, además, documentar el procedimiento de instalación, configuración y puesta en funcionamiento de dicha herramienta. **Juan** y **María** se están encargando de ello.



[Licencia Apache 2.0](#)

ANT (siglas de "Another Neat Tool", en español "Otra Herramienta Pura", que en inglés significan "hormiga") fue creado por James Duncan Davidson mientras realizaba la transformación del proyecto **Solar** de Sun Microsystems en código abierto (concretamente la implementación del motor JSP/Servlet de Sun, que luego se llamaría Jakarta Tomcat).

Apache Ant es una herramienta usada en programación para la realización de tareas mecánicas y repetitivas, normalmente se centra en la fase de compilación y construcción (build). Es similar al make empleado en Linux, pero desarrollado en Java; posee la ventaja de no depender de los comandos shell de cada sistema operativo, ya que se basa en archivos de configuración XML y clases Java, siendo idónea como solución multi-plataforma.

Podemos destacar los siguientes aspectos y funciones de las que **Ant** se va a ocupar:

- ✓ Compilación.
- ✓ Generación de documentación.
- ✓ Empaquetamiento.
- ✓ Ejecución, etc.

Es utilizado en muchos proyectos de desarrollo de Java y funciona a partir de un script de ensamblado, en formato XML (`build.xml`) que posteriormente se explicará con más detalle; además es fácilmente extensible e integrable con muchas herramientas empleadas por los desarrolladores, por ejemplo el editor JEdit o el IDE Netbeans.

Trabajar sin **Ant** implica una compilación manual de todos los ficheros .java (sin un control de los que han sido modificados y de los que no), incluir los  CLASSPATH relativos adecuados, tener los ficheros .class mezclados con el código fuente, etc.; sin embargo con **Ant**, en el fondo, no estás más que automatizando tareas, para que, al final, con un solo comando puedas compilar desde cero tu proyecto, ejecutar pruebas, generar la documentación, empaquetar el programa, etc.

Para trabajar con **Ant** se necesita tener instalado un JDK versión 1.4 o superior, ya que Ant no deja de ser una aplicación Java.

En la siguiente presentación se resume el concepto de la herramienta Ant.

Presentación donde se explica en qué consiste la herramienta Ant y cómo funciona.

[Descripción de la presentación de la Herramienta Ant](#)

Para saber más

En esta página podemos encontrar toda la información que nos pueda interesar para comenzar a trabajar con la herramienta Ant.

 [Página oficial de Apache Ant](#)

6.1.- Instalación y configuración de Ant.

Ant se puede instalar en cualquier sistema operativo, dado que está desarrollado en Java, y por lo tanto, es multiplataforma. No obstante, para el propósito de esta unidad, vamos instalar Ant en un ordenador con una distribución Ubuntu reciente. El proceso para instalarlo en cualquier otra distribución de Linux basada en Debian es similar.

Para poder realizar la instalación, primero debemos instalar un JDK 1.4 o superior. Para instalarlo puedes seguir cualquiera de los dos procesos explicados en los apartados 3.1.1 o 3.1.2. Para comprobar que versión de Java tienes instalada puedes usar el siguiente comando:



(cc) BY-NC-ND s.alt



Posteriormente procederemos a la descarga del paquete binario de Ant, el cuál podemos descargarlo de la siguiente página:

[Página de descarga de la distribución binaria de Ant.](http://apache.rediris.es/ant/binaries/apache-ant-1.10.3-bin.tar.gz)

También podemos descargarlo directamente desde un terminal, ejecutando el comando wget:



Una vez hemos descargado el archivo, lo podemos descomprimir empleando la instrucción siguiente:



Después, es buena idea mover la carpeta creada tras descomprimir a otro directorio, como por ejemplo /usr/local o /opt. Esta operación podríamos hacerla con el siguiente comando:



Lo único que falta ahora es crear la variable de entorno ANT_HOME y actualizar la variable PATH. Veamos que contiene cada una de estas variables:

- ✓ ANT_HOME: contiene la ruta al directorio raíz de la instalación de Ant. Para el ejemplo anterior, dicha ruta sería: <code>/opt/apache-ant-1.10.3.
- ✓ PATH: contiene las rutas de acceso a los archivos ejecutables (programas) del sistema; la modificación de esta variable permitirá acceder a los ejecutables de Ant desde cualquier otro directorio sin tener que poner la ruta completa.

Esta configuración podemos hacerla en varios archivos, tal y como se explicó con anterioridad: en el archivo /etc/environment, en el archivo /etc/profile, o en un script propio dentro de la carpeta /etc/profile.d, que podría llamarse por ejemplo /etc/profile.d/ant.sh. El contenido de dicho script podría ser el siguiente:

```
ANT_HOME=/opt/apache-ant-1.10.3  
export PATH=$ANT_HOME/bin:$PATH
```

Después, para que el sistema recoja los cambios realizados deberemos reiniciar sesión. Pero si no queremos reiniciar la sesión en ese momento podemos emplear el comando: #source /etc/profile.

Para comprobar que Ant se ha instalado correctamente, podemos ejecutar el comando ant desde un terminal. Al ejecutarlo deberíamos obtener un mensaje similar al siguiente:

```
[root@localhost ~]# ant  
Buildfile: build.xml does not exist!
```

Con esto la herramienta ant ya estaría correctamente instalada y configurada para desempeñar su función en nuestra máquina.

Debes conocer

En el siguiente vídeo puedes ver cómo se realiza la instalación de Ant en un equipo con el sistema operativo Microsoft Windows 7.

Presentación donde se explica como instalar Ant en Microsoft Windows 7.

[Descripción de la presentación de la Herramienta Ant](#)

Autoevaluación

Indica si la siguiente afirmación es verdadera o falsa:

"Ant es una herramienta que solo se puede instalar después de instalar el JDK."

- Verdadero
- Falso

6.2.- El archivo build.xml.

Como ya se ha comentado con anterioridad, **Ant** utiliza ficheros XML. Normalmente configuramos las tareas a automatizar con Ant a través de un fichero llamado build.xml, por lo que vamos a ver algunas de las etiquetas que podemos incluir en dicho archivo.

```
<?xml version="1.0" ?>
- <wxSIPUA>
  <Username>Hubert</Username>
  <Password>lala</Password>
</wxSIPUA>
```



hubert.tw

- ✓ **project:** Este es el elemento raíz del fichero XML y, como tal, solamente puede haber uno en todo el fichero, el que se corresponde a nuestra aplicación Java.
- ✓ **target:** Un target u objetivo es un conjunto de tareas que queremos aplicar a nuestra aplicación en algún momento. Se puede hacer que unos objetivos dependan de otros, de forma que eso lo trate Ant automáticamente.
- ✓ **task:** Un task o tarea es un código ejecutable que aplicaremos a nuestra aplicación, y que puede contener distintas propiedades (como por ejemplo el classpath). Ant incluye ya muchas tareas básicas, como compilación y eliminación de ficheros temporales, pero podemos extender este mecanismo si nos hace falta. Luego veremos algunas de las disponibles.
- ✓ **property:** Una propiedad o property es, simplemente, algún parámetro (en forma de par nombre-valor) que necesitamos para procesar nuestra aplicación, como el nombre del compilador, etc.

Veamos un ejemplo simple de archivo build.xml:

```
<?xml version="1.0" ?>

<project name="PracticaAnt" default="compilar" basedir=".">
  <!-- Definición de los directorios -->
  <property name="fuente" value="src"/>
  <property name="destino" value="bin"/>
  <!-- Tareas -->
  <target name="compilar">
    <javac source="1.5" target="1.5" destdir="${destino}" />
  </target>
  <target name="clean">
    <delete dir="${destino}"/>
  </target>
</project>
```

La estructura de este archivo es bastante simple. Por un lado, declaramos el proyecto (`<project>`), indicando la acción a realizar por defecto (`default="compilar"`), y el directorio base sobre el que se realizarán las tareas configuradas a continuación (`basedir="."`). Cuando el directorio base es solo un punto ("."), quiere decir que el directorio base es el directorio donde está el archivo build.xml.

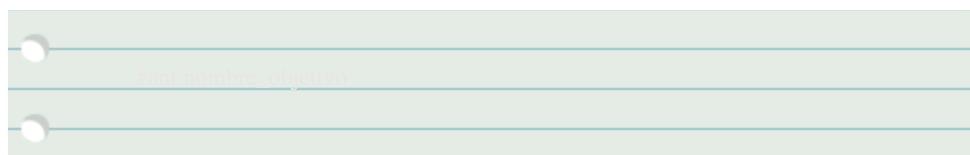
Después en el archivo de configuración aparecen etiquetas `property`, que contienen los directorios de origen y de destino (`<property name="fuente" value="."/>` y `<property name="destino" value="clases"/>`).

Por último, se puede ver un target (objetivo) llamado compilar (name="compilar"), que es el que hemos declarado como objetivo por defecto en la declaración del proyecto. En este objetivo tenemos una única tarea, la de compilación javac, a la que por medio de los atributos srodir y destdir le indicamos los directorios fuente y destino, que recogemos de las propiedades anteriormente declaradas con \${fuente} y \${destino}.

Lo único que nos queda ahora es usar este archivo build.xml. Para ello, estando situados en el directorio donde tenemos nuestro build.xml, desde la consola de Windows o el terminal GNU/Linux, podemos ejecutar lo siguiente:



Esto llevaría a cabo el objetivo declarado como objetivo por defecto en el proyecto (compilar, en este caso). Si el objetivo a ejecutar fuera otro, podríamos ejecutar Ant de la siguiente forma:



Ant se basa en ficheros XML, normalmente configuramos el trabajo a hacer con nuestra aplicación en un fichero llamado build.xml.

Autoevaluación

Para eliminar un directorio usando Ant, en el archivo build.xml deberíamos incluir:

- Una tarea dentro de un objetivo.
- Solo un objetivo.
- Una propiedad dentro de un objetivo.
- Una propiedad dentro de un proyecto.

6.3.- Empaquetar una aplicación con Ant.

Caso práctico

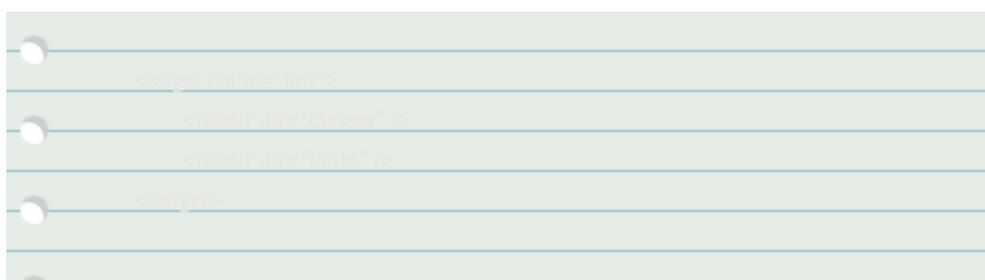
Juan y **María** están sacando mucho partido de esta herramienta. Gracias a ella están aprendiendo un montón, aunque están atascados en un punto: no saben como generar un paquete **WAR** usando Ant. **Juan** y **María** empiezan a pensar que no es posible, pero **Ada** ha visto por Internet que si que es posible, y se lo ha comentado por correo electrónico.

¿Crees que **Ada** tiene razón? Parece que sí. En este apartado vamos a ver como se puede configurar Ant para empaquetar una aplicación Java (archivo .jar) o una aplicación web (archivo .war). Para explicar el contenido de este apartado lo vamos a hacer mediante un ejemplo. En primer lugar creamos un fichero build.xml en la raíz de nuestro proyecto y definimos su nombre:



Ant, al igual que otras herramientas de construcción, se basa en el concepto de objetivo o target, explicado en el apartado anterior. Un objetivo engloba tanto las dependencias previas (objetivos que hay que hacer previamente), como los pasos a seguir para realizar el objetivo en si.

Vamos a comenzar definiendo un objetivo de preparación llamado init que será el encargado de crear un directorio classes donde guardaremos los ficheros .class resultantes de la compilación, y el directorio build para el archivo .jar final. Para ello basta incluir dentro de <project> las siguientes líneas:



Como podemos ver los objetivos se delimitan con etiquetas <target> y un nombre (name="init"). Dentro de ellos se enumeran los pasos que se han de seguir para alcanzar el objetivo, en este caso ha de crear dos directorios.

Si queremos ejecutar el objetivo **init** basta con realizar lo siguiente:

```
> ant init  
Buildfile build.xml  
  
<target name="init">  
    <echo message="Init target has been defined." />  
    <echo message="Compiling classes..."/>  
    <echo message="Cleaning up..." />
```

Es hora de compilar nuestro proyecto, para ello vamos a definir el objetivo **compile**. La compilación depende de la creación del directorio classes, que se realiza en el objetivo init (definido con anterioridad). Para indicar esta dependencia basta con poner lo siguiente (`depends="init"`):

```
<target name="compile" depends="init">  
    <javac srcdir="src" destdir="classes" />  
</target>
```

La dependencia se establece por tanto en la declaración del target, de tal manera que se garantiza su cumplimiento antes de comenzar a realizar un objetivo determinado. Nuestro código está en el directorio src, y el resultado de la compilación se lleva al directorio classes (). Aquí es importante notar que estamos usando la tarea `<javac>`, la cual es una de las muchas tareas que Ant lleva predefinidas.

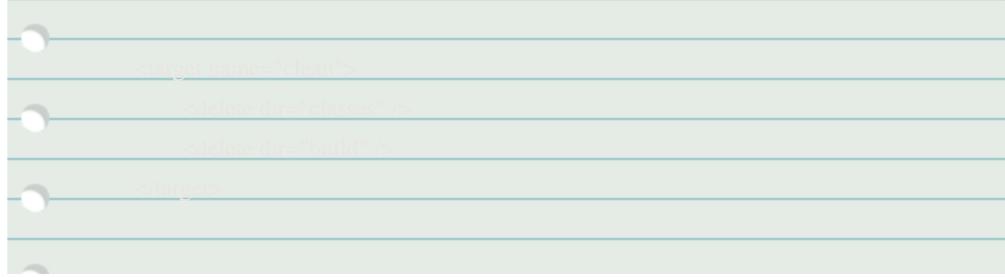
Con nuestro proyecto ya compilado, podremos generar el **.jar**. Para ello crearemos un nuevo objetivo llamado **build**. Veamos como podría ser:

```
<target name="build" depends="compile">  
    <jar destfile="build/proyecto.jar" basedir="classes" />  
</target>
```

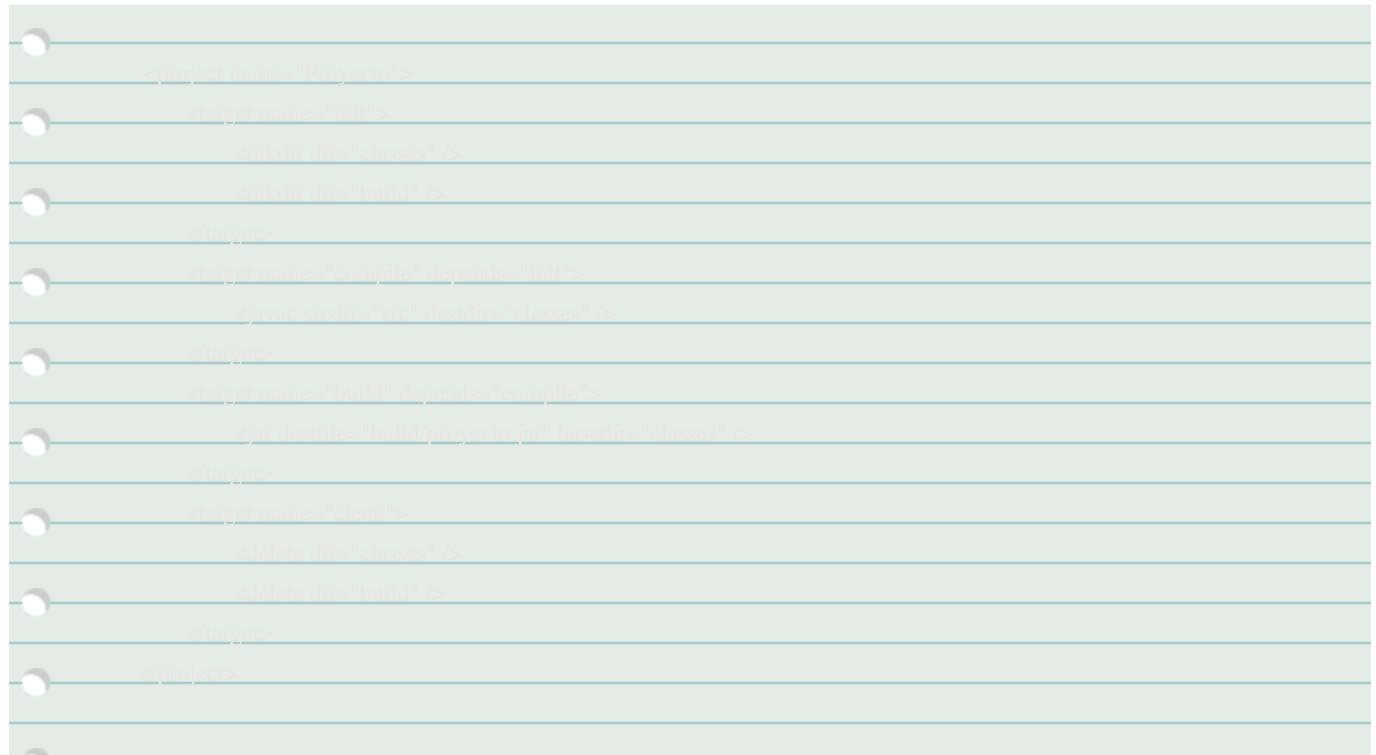
En el objetivo anterior (build), se ha indicado que hay una dependencia con el objetivo compile (`depends="compile"`), lo cual es importante, dado que es necesario compilar el código antes de poder empaquetarlo. Dentro de este objetivo, se utiliza la tarea `<jar>`, que se encarga de empaquetar todo el contenido del directorio classes en el fichero build/proyecto.jar.

Si en nuestra configuración necesitaramos generar un paquete .war, en lugar de un paquete .jar, bastaría con cambiar la extensión del archivo generado: `<jar destfile="build/proyecto.war" basedir="classes" />`.

Para completar este archivo build.xml, incluiremos un nuevo objetivo para limpiar todos los archivos generados, será el objetivo clean:



En este objetivo se vaciarían los directorios de trabajo `classes` y `build`, dejando el entorno limpio para poder repetir desde cero el proceso de empaquetado y compilación. Veamos como sería el archivo `build.xml` al completo:

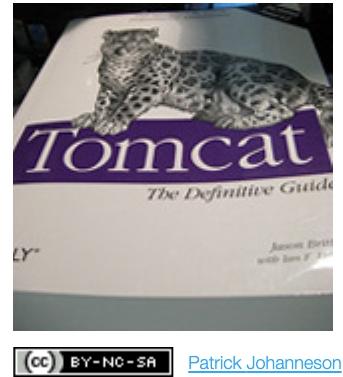


6.4.- Desplegar en Tomcat usando Ant.

Como ya se comentó en apartados anteriores, una aplicación web puede ser empaquetada en un archivo WAR para facilitar su despliegue y distribución entre clientes. Como ya sabrás, para empaquetar la aplicación web se puede emplear el comando jar, también se puede emplear la tarea <jar> de Ant, e incluso se puede utilizar un IDE como NetBeans o Eclipse.

La novedad ahora es que para desplegar un paquete WAR en un servidor de aplicaciones podemos también utilizar Ant. Existen una serie de tareas para Ant (no incluidas por defecto), que podemos utilizar para la gestión de las aplicaciones desplegadas en un servidor de aplicaciones, entre las cuales destacamos:

- ✓ <**deploy**>: despliega una aplicación web.
- ✓ <**start**>: inicia una aplicación web.
- ✓ <**stop**>: para una aplicación.
- ✓ <**undeploy**>: repliega (desinstala) una aplicación.

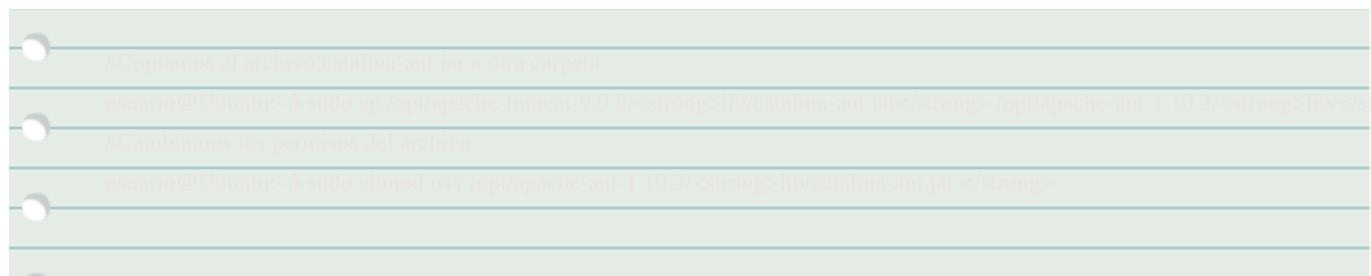


Pero, ¿cómo podemos desplegar con Ant una aplicación web en Tomcat? Ant es por fortuna una herramienta ampliable, esto quiere decir que se pueden ampliar las tareas que se pueden realizar desde un archivo build.xml programando el código adecuado. Y por suerte, Tomcat incluye una librería que permite automatizar el despliegue y repliegue desde Ant, entre otras cosas.

Veamos cual sería el primer paso. Para integrar las dos herramientas anteriores podemos seguir las siguientes operaciones:

- ✓ Instalar Ant como se describió con anterioridad.
- ✓ Copiar el fichero lib/catalina-ant.jar ubicado en la carpeta de instalación de Tomcat, en la carpeta ./lib de la instalación de Ant.
- ✓ Cambiar los permisos del archivo copiado anterior, de forma que cualquier usuario pueda leerlo.

Veamos un ejemplo de como podría ser el proceso:



El proceso anterior presupone que tanto Tomcat como Ant han sido instalados de forma manual (ten en cuenta que en función de la versión de Ant y Tomcat que hayas instalado, la carpeta de instalación sería otra). Si tienes dificultades para encontrar el archivo catalina-ant.jar, puedes ejecutar el siguiente comando:


```

<?xml version="1.0" encoding="UTF-8"?>
<project name="Ejemplo" default="default" basedir=".">>

<taskdef name="deploy" classname="org.apache.catalina.ant.DeployTask"/>

<target name="deploy">
    <deploy url="http://localhost:8080/manager/text/"
        username="sadmin"
        password="sadmin"
        path="/ejemplo"
        update="true"
        localWar="/srv/webapps/Ejemplo.war" />
</target>

</project>

```

En el ejemplo anterior podemos destacar los siguientes aspectos:

- ✓ A través del elemento `<taskdef>` se carga, desde la librería proporcionada por Tomcat, la tarea `<deploy>`.
- ✓ En la tarea `<deploy>` a través de sus diferentes atributos, se especifica:
 - ◆ URL del gestor de aplicaciones de Tomcat (fíjate que termina en `/text`).
 - ◆ Nombre del usuario con el rol manager- script que se usará para conectar (`username="sadmin"`).
 - ◆ Contraseña utilizada para conectar (`password="sadmin"`).
 - ◆ Ruta a través de la que será accesible la aplicación web (`path="/ejemplo"`).
 - ◆ En caso de existir, se actualizará la aplicación ya desplegada (`update="true"`).
 - ◆ Archivo WAR a desplegar en Tomcat (`localWar="/srv/webapps/Ejemplo.war"`).

El quinto paso sería desplegar a aplicación usando Ant con el objetivo antes configurado:



La ventaja de este proceso es que se permite cierto grado de automatización, de tal manera que cualquier cambio en la aplicación web se puede poner en producción rápidamente.

Autoevaluación

¿Cuál de las siguientes opciones no es un paso obligatorio y necesario para poder ejecutar una tarea de despliegue siguiendo todo el proceso anterior?

- Hacer que Ant use la librería catalina-lib.jar de Tomcat.
- Subir la aplicación a desplegar al servidor.
- Usar <taskdef> en el archivo build.xml.
- Configurar el archivo context.xml.

7.- Seguridad en aplicaciones web.

Caso práctico

Una de las primeras preocupaciones que se encuentran los administradores de servidores es la seguridad y protección de los mismos frente a posibles ataques o accesos incontrolados, por dicha causa, **María** se ha puesto a investigar las opciones a configurar, y herramientas a utilizar, para bloquear las posibles vulnerabilidades de los servidores web junto con los problemas de seguridad en las aplicaciones web.



(CC) BY-NC-ND irosell

Un servidor de aplicaciones es, usualmente, un software que proporciona una serie de servicios de aplicación a un número indeterminado de computadoras cliente que acceden a dichos servicios vía web; las principales ventajas de este tipo de tecnología es la centralización y disminución de la complejidad en el desarrollo de aplicaciones, sin embargo las aplicaciones web están así más expuestas a ataques.

Hoy en día existen aplicaciones web para casi todo y que tienen acceso a información muy valiosa como, por ejemplo, números de tarjetas de crédito, cuentas bancarias, historiales médicos, información personal, etc. Con lo cual, representan un objetivo interesante al que atacar; estos ataques se pueden clasificar en base a tres niveles:

- ✓ Ataques a la computadora del usuario (cliente).
- ✓ Ataques al servidor.
- ✓ Ataques al flujo de información que se transmite entre cliente y servidor.

En cada uno de los niveles anteriores es necesario garantizar una seguridad mínima para conseguir la seguridad de todo el proceso. A nivel de usuario éstos deben contar con navegadores y plataformas seguras, libres de virus; a nivel del servidor hay que garantizar que los datos no sean modificados sin autorización (integridad) y que sólo sea distribuida a las personas autorizadas (control de acceso) y, en lo que se refiere al tránsito de la información, ésta no debe ser leída (confidencialidad), modificada o destruida por terceros, al mismo tiempo que hay que garantizar un canal de comunicación fiable que no se interrumpa con relativa facilidad.

Para conseguir aplicaciones web seguras hay que establecer mecanismos que garanticen:

- ✓ Autenticación: permite identificar, en todo momento, quién es el usuario que está accediendo. Para conseguirlo existen varios métodos:
 - ➡ Autenticación básica (BASIC): solicitud de usuario y clave.
 - ➡ Autenticación con certificados.
 - HTTP DIGEST AUTH (HTTP Autenticación de texto implícita).
 - HTTP NTLM AUTH (HTTP Autentication Microsoft NT Lan Manager).
- ✓ Autorización: permite, una vez autenticado, determinar a qué datos y módulos de la aplicación puede acceder el usuario.
- ✓ Validación de los datos enviados por el cliente al servidor web, ya que se pueden manipular desde el cliente.
- ✓ Protección frente a la inyección de comandos SQL: esta es una técnica usada para explotar aplicaciones web que no validan la información suministrada por el cliente, para generar de esta forma consultas SQL peligrosas.

Para conseguir aplicaciones web seguras hay que utilizar una serie de mecanismos y herramientas entre las cuales destacamos:

- ✓ Deshabilitación de servicios y cuentas no utilizadas.
- ✓ Actualización del sistema operativo y aplicaciones ( parches).
- ✓ Fortaleza en las contraseñas.
- ✓ Utilización de firewalls.
- ✓ Back-ups periódicos.
- ✓ Análisis periódico de  logs.
- ✓ Verificación periódica de servicios activos.
- ✓ Cifrado del tráfico (usando protocolos como HTTPS).
- ✓ Establecimiento de políticas de seguridad.

Condiciones y términos de uso de los materiales

Materiales desarrollados inicialmente por el Ministerio de Educación, Cultura y Deporte y actualizados por el profesorado de la Junta de Andalucía bajo licencia Creative Commons BY-NC-SA.



Antes de cualquier uso leer detenidamente el siguiente [Aviso legal](#)

Historial de actualizaciones

Versión: 02.00.00

Fecha subida: 20/06/18

Autoría: Salvador Romero Villegas

Ubicación: Varios apartados

Mejora (tipo 1): Se ha revisado la parte que describe la utilización e instalación de Ant, y se han enmendado algunas imprecisiones y párrafos con textos inconexos. Se ha actualizado la versión de Ant.

Ubicación: Varios apartados

Mejora (tipo 3): - Apartado 2: Moverlo. - Apartado 3: Modificar el contenido de la página Instalación y Configuración Básica de Tomcat, para que funcione y se adapte a versiones de Debian más recientes. Se ha hecho una pequeña adaptación, pero para la versión que está en el paquete que en principio está muy desfasada. Añadir contenido que explique la instalación usando un JDK externo, como el de Oracle. Añadir contenido que explique la instalación de Tomcat desde repositorio. Añadir contenido que explique las diferentes versiones de Tomcat, de qué versión de Java dependen y qué proporcionan. Añadir contenido que explique que es un Servlet y su mecanismo de funcionamiento, que se habla mucho de servlet pero no se dice cómo funciona realmente, o se trata muy sucintamente. Añadir contenido que explique que es JSP y cómo se utiliza en Tomcat de forma general (sin entrar en la sintaxis de JSP de forma profunda). Añadir contenido que explique la arquitectura de tomcat: conectores, servidor, servicio, engine, host y contexto. Añadir contenido que explique la estructura de directorios de tomcat y los archivos de configuración (en especial el archivo server.xml). Revisar el contenido del apartado 3.2. Despliegue de aplicaciones web, tiene bastantes incoherencias (por ejemplo, directorios que se dan por sentado que tienen que existir, y que en principio no son necesarios y no se sabe exactamente por qué están ahí). Revisar el apartado 3.2.3 para indicar dónde se pueden poner los Valve y mostrando ejemplos dentro del archivo de configuración server.xml), a nivel de host o engine, etc. Revisar el apartado 3.2.4. Se hablan de sesiones persistentes pero no se responde a la pregunta de ¿qué es una sesión y para qué se usa? Es difícil entender que es una sesión persistente sin saber qué es una sesión, ni la gestión que se hacen en Tomcat (y en general Java EE) se hace de las sesiones. Además, se hace referencia a

archivos de configuración que no se han tratado previamente, como por ejemplo “elemento manager> como un elemento context>” que exactamente no se sabe que son. Tampoco se contextualiza en un ejemplo real, con lo que así es difícil que el alumnado lo visualice.

Reescribir el apartado Integración de Tomcat con Apache para que básicamente funcione lo que explica (usando mod_jk, por ejemplo) - Apartado 4: La versión a la que hace referencia está muy obsoleta y sería necesario reescribirlo para utilizar WildFly en vez de JbossAS (que ya no existe). Reescribir el proceso de instalación de Jboss para adaptarlo a WildFly y a versiones de Debian/Ubuntu recientes (el que hay escrito llega un paso que no funciona, paso 6 sino recuerdo mal). Aclarar que es un EJB y otros conceptos introducidos en la línea de un Java EE y que son propios de un servidor de aplicaciones y que quedan muy vagamente explicados, partiendo de que la estructura de una aplicación web y archivos WAR ya se ve en Tomcat (apartado 3) y que en principio, no van a cambiar, o en todo caso, indicar en que aspectos cambia de Tomcat a WildFly la composición de un archivo WAR. Poner un ejemplo real de una aplicación empresarial y su despliegue usando WildFly, porque se explica, pero no se dice ni como se despliega ni repliega una aplicación. - Apartado 6: Apartado 6.1 el paquete tomcat6-admin no es necesario instalarlo salvo que tomcat se instale desde el repositorio, cosa que no se describe en el apartado 3 (la instalación de tomcat desde repositorio). Además, aquí se describe una carpeta de configuración (y dos archivos de configuración que en principio no existen en una instalación por defecto) del engine catalina que debería describirse en el apartado 3 (y que no está descrito). Apartado 6.1 se indica que se tiene que crear un usuario con rol “manager” y debe ser “manager-gui”, lo he cambiado en la última revisión de la unidad. Apartado 6.2. describe contenido que tendría que ser revisado y que debería ir en el apartado 3, dado que describe los conectores y la estructura de Tomcat de forma muy sucinta, cosa que debería de haberse explicado antes. Es cierto que este contenido está relacionado con lo que se trata aquí, pero parece lógico que al ser algo general de tomcat, se trate antes. Apartado 6.2. se hace referencia a una aplicación llamada “hola” que es la primera vez que se nombra. Este apartado va sobre el application manager de tomcat... pero no se explica en realidad como usarlo para gestionar las aplicaciones. En general, no se contextualiza en un caso real. - Apartado 7: La aplicación OpenCart no corresponde con una aplicación web usada en un servidor de aplicaciones/contenedor de servlets tipo Tomcat o WildFly. Sería conveniente instalar una desplegable en un contenedor de servlets. La instalación descrita está obsoleta (la versión a la que hace referencia se ha cambiado por la actual, porque la anterior no era descargable), pero el proceso de instalación es impreciso y debería ser revisado en profundidad, y actualizad

Ubicación: general y tarea

Mejora (tipo 1): Avisar de que Jboss ya no funciona con JDK8, solo con JDK7. Ahora el proyecto se llama Wildfly

Versión: 01.01.04

Fecha subida: 31/01/18

Autoría: María Yolanda Jiménez Capel

Ubicación: 1 Introducción a las aplicaciones Web y servidores de aplicaciones

Mejora (tipo 1): He cambiado el enlace roto por el correcto que es el siguiente:<https://www.incubaweb.com/las-10-mejores-aplicaciones-web-20/>

Versión: 01.01.03

Fecha subida: 06/12/16

Autoría: Salvador Romero Villegas

Ubicación: 3.1, 3.2, 3.2.2

Mejora (tipo 1): El anterior apartado 3.1. se le ha cambiado el nombre de Iniciar tomcat a Integrar tomcat con apache que es más coherente, y se ha movido al final, siendo ahora el apartado 3.3.

3.1. (El nuevo 3.1, no el anterior que ahora es 3.3) Se ha modificado para que básicamente funcione, porque tal y como estaba no funcionaba ni se indicaba como iniciar o parar tomcat, cosa básica.

3.2. Se han quitado referencias a LAMP (apache y php que no venían al caso)

3.2.2. Se ha modificado para que realmente funcione, porque no funcionaba lo que ponía.

Ubicación: 5.3

Mejora (tipo 1): En el apartado 5.3 nos indica que para acceder al panel de administración de tomcat tenemos que añadir a tomcat-users.xml

```
role rolename=manager>
```

```
user username=usuario> password=clave> roles=manager/>
```

lo primero es que a la primera etiqueta rola le falta el simbolo / y lo segundo es que ese rol no sirve, al menos en tomcat 7 debe ser manager-gui

Y un pego, pero al menos a mi me ha retrasado un poco, hay que ejecutar catalina.sh como superusuario....

Versión: 01.01.02

Fecha subida: 05/07/16

Autoría: Julio Gómez López

Ubicación: No especificada.

Mejora (tipo 1): A partir de la unificación de las unidades 1 y 2, se han pasado contenidos a la unidad 3.

Materiales realizados por Narciso Jáimez Toro

Versión: 01.01.01

Fecha subida: 03/12/15

Autoría: Jesús Manuel Marín Navarro

Ubicación: 2, 3, 5.3

Mejora (tipo 1): Se fusionan las mejoras de semipresencial y distancia del año pasado.

En Apartado 2 se edita el Para Saber más sobre ASP y ASP.net

Añadida la referencia a Wildfly y JDK7/8

Versión: 01.01.00

Fecha subida: 10/01/15

Autoría: Jesús Manuel Marín Navarro

Añadida aclaración sobre ASP y ASP.NET

Añadido y corregido Glosario

Versión: 01.00.01

Fecha subida: 02/01/15

**Autoría: Manuel Alberto Domínguez
Vega**

incorporacion de credenciales a las imagenes y videos, correccion de erratas e incorporacion de las definiciones. (S)

Versión: 01.00.00

Fecha subida: 30/10/13

Autoría: Jesús Manuel Marín Navarro

Mejora: *No especificada.*