


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 servidores de aplicaciones, sería muy importante evaluar muchos parámetros que afectarían al correcto funcionamiento de los servidores, además de las necesidades de los mismos. Entre los parámetros a evaluar cabe destacar los siguientes:

- ✓ Seguridad de los servidores de aplicaciones: medidas de seguridad a aplicar para evitar posibles ataques o intrusiones.
- ✓ Dimensionamiento del servidor donde se estudian las necesidades físicas del equipo servidor.
- ✓ Tipo de servidor a instalar, características específicas del software de servidor seleccionado (Tomcat, Jboss/Wildfly, etc.).
- ✓ Despliegue de aplicaciones en el servidor donde habría que establecer qué herramientas se deberían utilizar.
- ✓ Administración de las conexiones remotas a los servidores.
- ✓ Escalabilidad de los servidores, a tener en cuenta en función del número de conexiones simultáneas que se pueden establecer.
- ✓ Herramientas de automatización de tareas en el servidor (Ant, etc.).

Debido a la cantidad de parámetros que hay que administrar para poner en correcto funcionamiento los servidores de aplicaciones, **Ada** ha decidido que sus empleados se documenten de todos y cada uno de ellos y, si cabe, la posibilidad realizar algún curso de formación sobre la administración de servidores de aplicaciones.



[Ministerio de Educación y Formación Profesional](#) (Dominio público)

Materiales formativos de FP Online propiedad del Ministerio de Educación y Formación Profesional.

[Aviso Legal](#)

1.- Protección del servidor de aplicaciones.



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.



[pxfuel](#) (CC0)

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: solicitud de usuario y clave.
 - Autenticación con certificados.
 - HTTP DIGEST AUTH (HTTP Autenticación de texto implícita).
 - HTTP NTLM AUTH (HTTP Authentication Microsoft NT Lan Manager).
- ✓ Autorización: permite, una vez autenticado, determinar a qué datos y módulos de la aplicación puede acceder el usuario.
- ✓ Validación de entradas, ya que se puede manipular el código de validación del lado del cliente.
- ✓ Inyección de comandos SQL: técnica para explotar aplicaciones web que no validan la información suministrada por el cliente para generar 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 periódicas.
- ✓ Análisis periódico de 📁 logs.
- ✓ Verificación periódica de servicios activos.
- ✓ Cifrado del tráfico.
- ✓ Establecimiento de políticas de seguridad.



Para saber más

Esta web surge con el objetivo de concienciar y ayudar a la gente para aumentar la seguridad en la red, en ella aparece, de forma actualizada, amenazas, ataques, recomendaciones de seguridad, etc.

[Seguridad en la red.](#)

2.- Despliegue de aplicaciones en Tomcat.



Caso práctico

María, ha montado una máquina Debian 6 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 práctica de despliegue de aplicaciones web 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.



Desplegar un servlet consiste en situar una serie de archivos en un contenedor web para que los clientes puedan

acceder a su funcionalidad; una aplicación web es un conjunto de servlets , páginas HTML, JSP, clases y otros recursos que se pueden empaquetar de una forma determinada.



Tomcat ([Apache](http://www.apache.org))

Una aplicación web puede ser desplegada en diferentes servidores web manteniendo su funcionalidad y sin ningún tipo de modificación en su código debido a la especificación servlet 2.2. Las aplicaciones web deben organizarse según la siguiente estructura de directorios:

- ✓ Directorio principal (raíz): Contendrá los ficheros estáticos (HTML, imágenes, etc...) y JSPs.
 - Carpeta **WEB-INF**: contiene el fichero "*web.xml*" (descriptor de la aplicación), encargado de configurar la aplicación.
 - Subcarpeta **classes**: contiene los ficheros compilados (servlets, beans).
 - Subcarpeta **lib**: librerías adicionales.
 - Resto de carpetas para ficheros estáticos.

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

- ✓ Por medio de archivos WAR.
- ✓ Editando los archivos *web.xml* y *server.xml*, este método es el que se pasa a tratar a continuación.

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

La carpeta **www** contiene a su vez una carpeta, con el nombre y ruta del proyecto, que contiene los ficheros que forman la interfaz (`HTML`, `js`, `css`...). La carpeta **bin** contiene las clases de java de la aplicación.

Para desplegar la aplicación en Tomcat se deben realizar los siguientes pasos:

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

Vamos a partir de una máquina con el sistema operativo Ubuntu 20.04 la cual tenemos el servidor Tomcat corriendo para mostrar el proceso creación y despliegue de aplicaciones. Debido a que pretendemos montar una plataforma LAMP, por sus ventajas derivadas de las características del software libre, instalaremos también los siguientes componentes: MySQL y PHP.

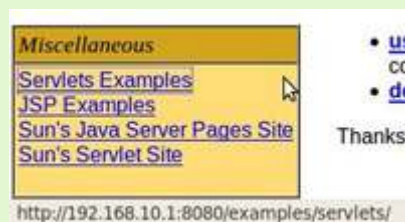
Recordemos, **en primer lugar destacar que, para instalar cualquier versión de Tomcat es necesario tener instalado JDK (Kit de desarrollo de Java), ya que el objetivo es que las peticiones a Apache se redirijan a Tomcat empleando un conector proporcionado por Java en este caso.**

2.1.- Creación de una aplicación web.



Caso práctico

En la empresa **BK programación**, **Juan** ha decidido documentar los métodos que resulten más útiles y sencillos a seguir para la creación de una aplicación web, de manera que pueda desplegarse sin ningún tipo de dificultad en el servidor de aplicaciones Tomcat que **María** ha montado. De esta de manera, los clientes tendrán disponibles todas las funcionalidades de las aplicaciones desarrolladas en la empresa.



Tomcat ([Apache](#))

El servidor de aplicaciones Tomcat cuenta con una serie de ejemplos, tanto de servlets como de JSP, que sirven de ayuda para aprender a realizar las tareas creación y despliegue de aplicaciones web.

Es muy interesante crear dos variables de entorno: `JAVA_HOME` que indique la ubicación de los archivos binarios de Java y `CATALINA_HOME` que apunta a la ubicación de los scripts de Tomcat, para ello podemos añadir el siguiente código al archivo `/etc/profile`.

```
CATALINA_HOME=/opt/tomcat
JAVA_HOME=/usr/lib/jvm/java-1.11.0-openjdk-amd64
PATH=$PATH:$JAVA_HOME/bin:$CATALINA_HOME
export PATH JAVA_HOME CATALINA_HOME
```

Actualizamos las variables de entorno mediante el comando:

```
source /etc/profile
```

El lenguaje Javascript se ejecuta del lado del cliente, es un lenguaje interpretado de scripting que no permite acceder a información local del cliente ni puede conectarse a otros equipos de red.

En primer lugar crearemos una carpeta con el nombre que nos interese para identificar la aplicación, en este ejemplo hemos optado por **Aplic_Web** una estructura como la de la siguiente imagen:

```
Aplic_Web/  
├── index.jsp  
└── WEB-INF  
    ├── classes  
    └── lib
```

La aplicación que pretendemos desarrollar contiene un archivo al que llamaremos **index.jsp** muy sencillo con el siguiente contenido:

```
<html>  
  
<head>  
  <title>C.F. DESARROLLO DE APLICACIONES WEB</title>  
  <script language="Javascript">  
    function popup() { alert("U.T. 3: CONFIGURACION Y ADMINISTRACION DE SERVIDORES DE  
  </script>  
</head>  
  
<body>  
  <h1 align=center>DESPLIEGUE DE APLICACIONES WEB</h1>  
  <div align=center>  
    <form><input type="button" value="UNIDAD 3" onclick="popup()"></form>  
  </div>  
</body>  
  
</html>
```

para acabar, solamente nos quedaría hacer una copia de la carpeta de nuestra aplicación en **\$CATALINA_HOME/webapps** y si, posteriormente desde un navegador, accedemos en local a http://127.0.0.1:8080/Aplic_Web tendríamos la aplicación funcionando.



Reflexiona

Si el equipo en el que hemos desarrollado la aplicación anterior, y en donde se ha puesto a funcionar, pertenece a una red de computadores y tiene la IP: 192.168.10.1. ¿Podríamos acceder desde otros computadores a la aplicación web? En caso afirmativo, ¿cual sería la URL que deberíamos teclear?

2.2.- Despliegue de una aplicación web.

Uno de los objetivos que se persigue en el momento de desarrollar aplicaciones web, es que éstas puedan ser desplegadas en diferentes servidores web, manteniendo su funcionalidad y sin ninguna modificación de código.

Los WARs simplemente son archivos Java de una aplicación web con una extensión diferente para diferenciarlos de los comunmente usados JARs.

Antes de la especificación Servlet 2.2, era bastante diferente desplegar servlets entre diferentes contenedores de servlets, anteriormente también llamados motores servlet. La especificación 2.2 estandarizó el despliegue entre contenedores, llevando así la portabilidad del código Java un paso más allá.

El método más sencillo para desplegar una aplicación, que sobre todo se utiliza durante la etapa de desarrollo de la misma, es el realizado en el punto anterior, es decir, copiar la carpeta correspondiente a nuestra aplicación en la carpeta `$CATALINA_HOME/webapps`, teniendo en cuenta que la variable `$CATALINA_HOME` es la ruta de los scripts que emplea Tomcat.

Siguendo con la aplicación desarrollada en el punto anterior (Aplic_Web), vamos a crear un fichero descriptor del despliegue `web.xml` **que es el encargado de describir las características de despliegue de la aplicación.**

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee
  version="2.5">
  <display-name>Descriptor Aplicacion Aplic_Web</display-name>
  <description>
    Mi primer descriptor web.xml.
  </description>
</web-app>
```

Este archivo lo situaremos en la carpeta WEB-INF perteneciente a la aplicación en desarrollo, de forma que la estructura de la carpeta resultante sería el mostrado en esta imagen:



```
Aplic_Web/
├── index.jsp
└── WEB-INF
    ├── classes
    ├── lib
    └── web.xml
```

3 directories, 2 files

Elaboración propia (CC0)

Una vez consideramos terminada nuestra aplicación web podremos generar el archivo .WAR perteneciente a la aplicación, para ello podemos aplicar los siguientes comandos:

`javac -d WEB-INF/classes *.java` este comando tiene como finalidad la compilación de las clases Java de nuestra aplicación.

`jar cvf Aplic_Web.war *` para crear el archivo .WAR.

Una vez hecho lo anterior podríamos acceder vía web a: `http://127.0.0.1:8080` y, en el apartado "*Administration*", accedemos a la opción "*Tomcat Manager*" y desde la ventana resultante tenemos las opciones que aparecen en la siguiente imagen para desplegar el archivo .WAR:



Tomcat ([Apache](#))



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.](#)

2.3.- Implementar el registro de acceso.



Caso práctico

Sobre las aplicaciones web que han sido desarrolladas por la empresa BK programación y que ya están accesibles para sus clientes, se ha considerado realizar de algún modo un seguimiento, de manera que se pueda comprobar los accesos que han tenido, en qué momento y qué recursos son más demandados; para ello **Juan**, junto con **María**, han configurado el servidor Tomcat para poder adaptar los logs, de manera que puedan obtener información sobre los accesos a sus aplicaciones.



[LinuxAndUbuntu](#) (CC BY-SA)

Para conseguir obtener y poder configurar los registros de acceso a un servidor de aplicaciones Tomcat, como es nuestro caso, empezaremos hablando de las válvulas de registro de acceso de Tomcat, ya que será el método que emplearemos.

Las válvulas del Tomcat son una tecnología introducida a partir de Tomcat 4 que permite asociar una instancia de una clase Java a un contenedor "*Catalina*". Esta configuración permite que la clase asociada actúe como un pre-procesador de las peticiones. Estas clases se llaman válvulas, y deben implementar la interfaz "*org.apache.catalina.Valve*" interface o extender de la clase "*org.apache.catalina.valves.ValveBase*". Las válvulas son propias de Tomcat y no pueden ser usadas en otros contenedores de servlet.

Las válvulas disponibles son:

- ✔ **Access Log Valve:** está implementada por la clase `org.apache.catalina.valves.AccessLogValve`. Crea ficheros de log para rastrear el acceso a la información de los clientes, registrando información como, por ejemplo, actividad de la sesión del usuario, información de la autenticación del usuario, entre otras. Por ejemplo, el siguiente código:

```
<Valve className="org.apache.catalina.valves.AccessLogValve" directory="logs" prefix
```

Indicará que los logs de acceso se almacenarán en el directorio `$CATALINA_HOME/logs` y los archivos de log tendrán la nomenclatura con prefijo: `localhost_access_log` y sufijo `.txt` probablemente entre sufijo y prefijo se añadirá la fecha en la que se crea dicho archivo.

- ✔ **Remote Address Filter:** permite comparar 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 donde a los clientes que tengan una IP que comienza por 127 se les va a denegar la solicitud.

- ✔ **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.*"/>
```

- ✔ **Request Dumper:** es una herramienta de depuración que escribe en el log el detalle de cada petición realizada.

```
<Valve className="org.apache.catalina.valves.RequestDumperValve"/>
```

Cualquier acceso a localhost:8080 tendrá asociado una serie de entradas en los logs.

- ✔ **Single Sign On:** cuando queremos que los usuarios puedan identificarse en cualquier aplicación de nuestro virtual host, y que su identidad sea reconocida por cualquier aplicación que esté en ese host.

```
<Valve className="org.apache.catalina.authenticator.SingleSignOn"/>
```

Podemos implementar los ejemplos anteriores en `$CATALINA_HOME/conf/server.xml`, de este modo dichos cambios afectarán a cualquier aplicación desplegada en el servidor.



Citas para pensar

"Los sistemas nuevos generan problemas nuevos."

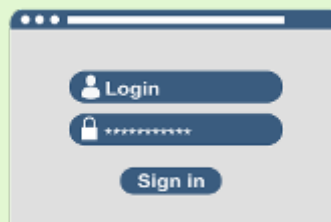
Ley de Murphy.

2.4.- Sesiones persistentes.



Caso práctico

Sobre las aplicaciones web que han sido desarrolladas por la empresa **BK programación** y que ya están accesibles para sus clientes, **Ada** ha solicitado a **Juan** y **María** cómo poder, de algún modo, garantizar las sesiones, estableciendo en la configuración de Tomcat sesiones persistentes que aseguren sesiones fiables a las aplicaciones en caso de caída del servidor o de pérdida de conexión.



[Gerd Altmann \(Pixabay\)](#)

Las sesiones activas por parte de clientes a aplicaciones web alojadas en servidores web Tomcat, por defecto, están configuradas para mantenerse en caso de posibles pérdidas de conexión con el servidor o posibles reinicios del mismo; a pesar de todo ello es posible establecer un control mayor sobre dichas sesiones.

Por lo que respecta a las sesiones inactivas (pero todavía no caducadas) es posible configurarlas de forma que se almacenen en disco liberando, como consecuencia de ello, los recursos de memoria asociados. Al parar Tomcat las sesiones activas se vuelcan a disco de manera que, al volver a arrancarlo, se podrán restaurar.

Las sesiones con un tiempo de vida que supere un límite se copian automáticamente a disco por seguridad para evitar posibles bloqueos de sesión.

Para configurar las sesiones persistentes tendremos que gestionar el elemento `<Manager>` como un subelemento de `<Context>` de forma que podemos actuar a dos niveles en función de si pretendemos que la configuración establecida se aplique a todas las aplicaciones del servidor o a una aplicación concreta.

Si configuramos las sesiones persistentes de forma global tenemos que manipular el archivo `/conf/context.xml`, mientras que si queremos configurar las sesiones a nivel local a una aplicación web determinada tendríamos que adaptar el archivo `<CATALINA_HOME>/conf/context.xml` correspondiente a la aplicación.

Un ejemplo de configuración podría ser el siguiente (se emplean comentarios para explicar cada uno de los parámetros):

```
<Context>
  <!-- classname especifica la clase del servidor que implementa el gestor, es recomend
  <Manager className="org.apache.catalina.session.PersistentManager"
    <!--saveOnRestart=true para indicar que se guarden todas las sesiones al reiniciar
    saveOnRestart="true"
    <!--maxActiveSession cuando se supera el límite aquí establecido se comienzan a en
    maxActiveSession="3"
    <!--minIdleSwap establece el número mínimo de segundos que transcurren antes de qu
    minIdleSwap="0"
    <!--maxIdleSwap indica el número máximo de segundos que transcurren antes de que u
    maxIdleSwap="60"
    <!--maxIdleBackup para indicar el número de segundos desde que una sesión estuvo a
    maxIdleBackup="5">
    <!--Store indica cómo y donde almacenar la sesión, están disponibles las siguientes
    <Store className="org.apache.catalina.session.FileStore" />
  </Manager>
</Context>
```

2.5.- Configurar Tomcat en cluster.



Caso práctico

Una vez que se han puesto las aplicaciones que **BK programación** ha terminado de desarrollar en el servidor de Tomcat, se ha observado un incremento exponencial en el número de clientes que acceden a los servicios de dichas aplicaciones, motivo por el cual se ha pensado en establecer algún tipo de cluster sobre el servidor para poder atender eficientemente a las peticiones de los usuarios.



[Br1dotcom](#) (CC BY)

Debido al incremento de las aplicaciones web, la escalabilidad y la disponibilidad se transforma en un recurso transcendental para garantizar el servicio eficiente a los clientes web; la implementación de clustering para los servidores de aplicaciones web es una solución eficaz y relativamente sencilla.

La implementación de clustering con Tomcat provee:

- ✓ **Escalabilidad:** si para ofrecer un servicio solicitado por un cliente web, un servidor web invierte un tiempo "T", para satisfacer un número elevado de servicios, cabe preguntarse cuánto es el tiempo invertido. La respuesta ideal a la cuestión anterior sería que el tiempo invertido fuese lo más próximo posible al tiempo invertido en una única petición, es decir lo más cercano posible a "T". Para ello existen dos posibles soluciones: escalado horizontal (implica el incremento del número de servidores), escalado vertical (implica el incremento de los recursos del propio servidor).
- ✓ **Alta disponibilidad:** Tomcat provee failover; en el motor del servidor existen dos tipos de failover provistos por clustering:
 - *Request-level failover:* Si un servidor cae, los siguientes requerimientos se redireccionarán a otros servidores activos.
 - *Session-level failover:* En el caso de que un servidor deje de dar servicio, otro servidor del cluster debería proporcionar la sesión a los clientes consiguiendo reducir al mínimo la pérdida de conexión, ello implica replicar la sesión en el cluster en la nueva máquina en el mínimo tiempo posible.
- ✓ **Balanceo de carga:** establecer un método de reparto de la carga de peticiones entre los servidores del cluster, de modo que se minimice el tiempo de respuesta a

las solicitudes de los clientes; se consigue empleando algoritmos de distribución de carga.

Las soluciones de clustering típicas ofrecen un paradigma de servidor que consiste en ofrecer un sistema basado en ejecución distribuida, a pesar de que existe limitación respecto a la escalabilidad, podemos observar el esquema de Jakarta Tomcat server engine works.

[Clustering/Session Replication How-To \(en inglés\)](#)

El conector del servidor de cluster recibe la petición desde los clientes, y el procesador del servidor de cluster encapsula las peticiones en los objetos "RequestEntry" y los escribe en JavaSpace. El conector del Worker del cluster toma dichas peticiones y el procesador del worker del cluster resuelve las peticiones.

Para establecer una configuración de cluster en Tomcat podremos seguir los siguientes pasos:

- ✓ Todos los atributos de sesion deben implementar `java.io.Serializable`.
- ✓ Descomentar el elemento `Cluster` en `server.xml`.
- ✓ Descomentar `Valve` (`ReplicationValve`) en `server.xml`
- ✓ Si las múltiples instancias de Tomcat están en la misma máquina el parámetro `tcpListenPort` tiene que ser único para cada una de las instancias.
- ✓ Establecer en el archivo `web.xml` el elemento `<distributable/>` o bien definirlo de forma `<Context distributable="true"/>`.
- ✓ El atributo `jvmRoutes` tiene que estar definido en el "Engine" `<Engine name="Catalina" jvmRoute="nodeX">` estableciendo su valor al nombre de la instancia en el cluster.
- ✓ Sincronizar la hora de todos los nodos con un servicio NTP.
- ✓ Configurar el parámetro `loadbalancer` en modo "sticky session".



Para saber más

Estas webs documentan los pasos a seguir para montar un cluster horizontal formado por dos servidores con una instancia de Tomcat corriendo en cada uno de ellos.

[Cluster mediante Tomcat](#)

[Clustering de Apache Tomcat](#)

3.- El servidor de aplicaciones JBoss/Wildfly.



Caso práctico

Los empleados de **BK programación** han oído hablar de la importancia del servidor de aplicaciones JBoss/Wildfly, ya que se trata de un servidor de código abierto orientado a aplicaciones e-bussines; siendo, por todo ello, una plataforma que ha adquirido una gran importancia en el mercado, tanto de particulares como de grandes empresas, y que merece la pena estudiar su comportamiento para poder implantar.

Del mismo modo, es interesante establecer el modo a operar para la instalación y configuración del servidor JBoss/Wildfly, así como de todos y cada uno de los pasos necesarios para poder realizar el despliegue de aplicaciones.



[Marsupilami](#) (Dominio público)

El servidor JBoss es un proyecto de código abierto, con el que se consigue un servidor de aplicaciones basado en J2EE, e implementado al 100 % en Java.

Mientras el Tomcat es un Servlet Container, JBoss es un Application Server, que soporta funciones de J2EE, las más importantes son los EJB's y el clustering. Tomcat por sí solo simplemente sirve para JSP's y servlets.

JBoss es un servidor de aplicaciones basado en Java mientras que Tomcat es un contenedor de servlets.

Uno de los rasgos más importantes de JBoss es su apoyo a la implementación "en caliente". Lo que significa es que implementar un nuevo EJB es tan simple como copiar el archivo correspondiente en el directorio correspondiente. Si esto se hace mientras el bean ya está cargado, JBOSS lo descarga automáticamente, y entonces carga la nueva versión.

JBoss está compuesto por dos partes: un "Servlet Engine" y un "EJB Engine", dentro del "Servlet Engine" se ejecutan exclusivamente las clásicas aplicaciones de un servidor (JSP's y Servlets), mientras el "EJB Engine(Container)" es reservado para aplicaciones desarrolladas alrededor de EJB's o Enterprise Java Bean's.

JBoss es el primer servidor de aplicaciones de código abierto preparado para la producción y certificado J2EE 1.4, ofrece una plataforma de alto rendimiento para aplicaciones de e-business. Combinando una arquitectura orientada a servicios revolucionaria con una licencia de código abierto, JBoss puede ser descargado, utilizado,

incrustado y distribuido sin restricciones por la licencia. Por este motivo, es la plataforma más popular de middleware para desarrolladores, vendedores independientes de software y, también, para grandes empresas.

Entre las características destacadas de JBoss destacamos las siguientes:

- ✓ Producto de licencia de código abierto sin coste adicional.
- ✓ Cumple los estándares.
- ✓ Confiable a nivel empresa.
- ✓ Incrustable, orientado a arquitectura de servicios.
- ✓ Flexibilidad consistente.
- ✓ Servicios de middleware para cualquier objeto de Java.

El creador de la primera versión de JBoss fué Marc Fleury quién fundó una empresa de servicios llamada JBoss Inc., adquirida en 2006 por Red Hat.

JBoss es líder del mercado en ofrecer soluciones middleware Open Source de nivel empresarial. JBoss Middleware Enterprise está compuesto por un conjunto de plataformas y frameworks certificados y soportados con el nivel de calidad profesional que ofrece Red Hat.

Las soluciones de JBoss Enterprise Middleware se distribuyen vía las "JBoss Subscription", que incluyen el software certificado y actualizaciones, herramientas de gestión, políticas de mantenimiento a largo plazo y un soporte técnico líder en la industria. Las suscripciones están disponibles tanto para uso en producción como para desarrollo.

Las plataformas JBoss Enterprise, que se detallan a continuación, integran múltiples proyectos y componentes, los más populares de la comunidad JBoss.org en distribuciones certificadas, estables y seguras, con una única vía de parches y actualizaciones.

✓ **JBoss Enterprise Application Platform.**

- ◆ JBoss EAP es el nombre del servidor de aplicaciones Java EE que Red Hat produce y soporta. La última versión es la 7 en este momento e implementa Java EE 7.
- ◆ Diseñado para construir, desplegar y albergar servicios, y aplicaciones Java.
- ◆ Integra el servidor de aplicaciones JBoss AS en cluster, un sistema de mapeo y persistencia O/R y, además, un potente framework para la construcción de aplicaciones de nueva generación Web 2.0.
- ◆ Está basado en suscripciones.

✓ **JBoss AS/Wildfly**

- ◆ WildFly y JBoss AS son las versiones comunitarias de JBoss.
- ◆ El término "JBoss Application Server" se usaba desde el principio, sin embargo para evitar confusiones con la versión de soporte (JBoss EAP), se renombró a WildFly.
- ◆ JBoss AS/WildFly están orientados a pruebas y desarrolladores.
- ◆ Algunas versiones son en realidad bastante estables y *podrían* ejecutarse en producción, pero sin soporte de Red Hat, sólo de la propia comunidad.

✓ **JBoss Web Server**

- ◆ JBoss Web Server está pensado para desplegar aplicaciones basadas en Java Server Pages (JSP), Java Servlet, PHP y CGI.
- ◆ Está pensado para aplicaciones medianas y grandes.
- ◆ Combina el servidor web Apache y el servidor de aplicaciones Tomcat bajo una suscripción de soporte.



Autoevaluación

¿Cuáles de las siguientes son características del servidor de aplicaciones JBoss/Wildfly?

☐ Es de código abierto.

☐ Está implementado en su totalidad en Java.

☐ Es únicamente un "EJB Container".

☐ Funciona únicamente en servidores Microsoft Windows.

☐ Está orientado a arquitectura de servicios.

Mostrar retroalimentación

Solución

1. Correcto
2. Correcto
3. Correcto
4. Incorrecto
5. Correcto

3.1.- Instalación y configuración básica.

Vamos a partir de una máquina Ubuntu 20.04 LTS, en la que realizaremos el proceso de instalación y configuración básica del servidor JBoss/Wildfly y que vamos a estructurar en los siguientes pasos:



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

1.- Descargar e instalar el Java Development Kit (JDK): En primer lugar, destacar que, para instalar cualquier versión de JBoss/Wildfly, es necesario tener instalado JDK (Kit de desarrollo de Java), ya que se trata de un servidor de aplicaciones basado e implementado al 100 % en Java, como se ha dicho anteriormente, y puede ser ejecutado en cualquier sistema en el que se encuentre operativo un JDK en su versión 1.5 o superior. Empezamos instalando el JDK. Para instalar la versión JDK (OpenJDK), ejecutamos el comando:

```
# apt update
# apt install default-jdk
```

2.- Crear el usuario de Wildfly: Es recomendable ejecutar Wildfly con una cuenta de usuario no root, con privilegios mínimos. Para ello crearemos un grupo `wildfly` y un usuario llamado `wildfly` que agregaremos al grupo creado; podemos hacerlo del siguiente modo:

```
sudo groupadd -r wildfly
sudo useradd -r -g wildfly -d /opt/wildfly -s /sbin/nologin wildfly
```

3.- Descargar e instalar de Wildfly 19.1.0: Se pueden descargar las distintas versiones del servidor Wildfly del siguiente enlace, en este caso hemos decidido descargar el paquete `wildfly-19.1.0.Final.zip`.

[Wildfly](#)

Para proceder a su instalación primeramente lo descargamos a la carpeta `/tmp`:

```
WILDFLY_VERSION=19.1.0.Final
sudo wget https://download.jboss.org/wildfly/$WILDFLY_VERSION/wildfly-$WILDFLY_VERSION
```

A continuación descomprimos el archivo y lo movemos al directorio `/opt`:

```
sudo tar xf /tmp/wildfly-$WILDFLY_VERSION.tar.gz -C /opt/
```

Creamos el enlace simbólico para Wildfly:

```
sudo ln -s /opt/wildfly-$WILDFLY_VERSION /opt/wildfly
```

Le pasamos la propiedad del directorio al usuario `wildfly`:

```
sudo chown -RH wildfly: /opt/wildfly
```

4.- **Configurar** `system.d` y **archivo** `wildfly.conf`. Creamos el directorio para guardar el archivo de configuración:

```
sudo mkdir -p /etc/wildfly
```

Copiamos el archivo de configuración al directorio:

```
sudo cp /opt/wildfly/docs/contrib/scripts/systemd/wildfly.conf /etc/wildfly/
```

Copiamos el script `launch.sh` script al directorio `/opt/wildfly/bin/`:

```
sudo cp /opt/wildfly/docs/contrib/scripts/systemd/launch.sh /opt/wildfly/bin/
```

Le ponemos el atributo ejecutable a los ficheros del directorio `bin`:

```
sudo sh -c 'chmod +x /opt/wildfly/bin/*.sh'
```

Finalmente, copiamos el archivo del servicio al directorio `/etc/systemd/system/`:

```
sudo cp /opt/wildfly/docs/contrib/scripts/systemd/wildfly.service /etc/systemd/system/
```

Recargamos `system.d`, arrancamos el servicio Wildfly y comprobamos que esté funcionando:

```
sudo systemctl daemon-reload
sudo systemctl start wildfly
sudo systemctl status wildfly
```

5.- Configurar la autenticación de Wildfly. Agregamos un usuario ejecutando el script `add.user.sh`:

```
sudo /opt/wildfly/bin/add-user.sh
```

Nos aparecerá una pregunta, y seleccionamos la opción **a**.

```
What type of user do you wish to add?
a) Management User (mgmt-users.properties)
b) Application User (application-users.properties)
(a):
```

A continuación añadimos un usuario, en el ejemplo `usuario/usuario`:

```
Enter the details of the new user to add.
Using realm 'ManagementRealm' as discovered from the existing property files.
Username : usuario
Password recommendations are listed below. To modify these restrictions edit the add-user.sh script
- The password should be different from the username
- The password should not be one of the following restricted values {root, admin, a
- The password should contain at least 8 characters, 1 alphabetic character(s), 1 d
Password :
WFLYDM0098: The password should be different from the username
Are you sure you want to use the password entered yes/no? y
Re-enter Password :
What groups do you want this user to belong to? (Please enter a comma separated list
About to add user 'usuario' for realm 'ManagementRealm'
Is this correct yes/no? y
Added user 'usuario' to file '/opt/wildfly-19.1.0.Final/standalone/configuration/mgm
Added user 'usuario' to file '/opt/wildfly-19.1.0.Final/domain/configuration/mgmt-us
Added user 'usuario' with groups  to file '/opt/wildfly-19.1.0.Final/standalone/conf
Added user 'usuario' with groups  to file '/opt/wildfly-19.1.0.Final/domain/configur
Is this new user going to be used for one AS process to connect to another AS proces
e.g. for a slave host controller connecting to the master or for a Remoting connecti
yes/no? y
To represent the user add the following to the server-identities definition <secret
```

Por último, ya podemos verificar la instalación de Wildfly, tecleando:

```
sudo systemctl status wildfly
```

O accediendo desde el navegador en las direcciones:

```
http://<dirección_IP>:8080 ó http://localhost:8080
```

6.- **Acceder a la consola de administración de Wildfly:** Asegurarse que Wildfly se ha iniciado y de que conseguimos acceder a la consola Wildfly desde las siguientes direcciones:

```
http://<dirección_IP>:9990 ó http://localhost:9990
```

Y accedemos con el usuario creado anteriormente.



Wildfly ([GNU/LGPL](#))

Para realizar la instalación y configuración básica del servidor JBoss 6.0, debemos seguir, de forma secuencial, todos y cada uno de los siguientes pasos:

- 1.- **Descargar e instalar el Java Development Kit (JDK).**
- 2.- **Crear el usuario de Wildfly.**
- 3.- **Descargar e instalar de Wildfly 19.1.0.**
- 4.- **Configurar `system.d` y archivo `wildfly.conf`.**
- 5.- **Configurar la autenticación de Wildfly.**
- 6.- **Acceder a la consola de administración de Wildfly.**

3.2.- Despliegue de aplicaciones empresariales.

La estructura de una aplicación web en su forma más sencilla, debe contener la siguiente estructura de directorios:

- ✔ META-INF/
- ✔ manifest.mf
- ✔ WEB-INF/
- ✔ classes/
- ✔ src/
- ✔ lib/
- ✔ web.xml



[Espacio CAMON](#) ([CC BY-NC-SA](#))

Conteniendo la carpeta **META-INF**, el archivo **manifest.mf**, que contiene la lista de contenidos de la aplicación, y que son generados al momento de crearla. El directorio **WEB-INF** contiene todos los archivos necesarios para ejecutar la aplicación, y estructura su contenido en las carpetas **classes** que contiene las clases compiladas para la aplicación, **lib** con las librerías necesarias para la aplicación y **src**, con el código fuente de la aplicación.

Una vez que la aplicación JEE está correctamente construida, se realiza el empaquetado con el comando:

```
# jar cvf nombre_aplicacion.jar carpetas/ficheros_a_empaquetar
```

Una vez tenemos la aplicación .jar para desplegarla, únicamente la copiamos a la carpeta "\$JBoss_HOME/server/default/deploy" y el propio JBoss nos dará un mensaje similar a *deploy, ctxPath = / nombre_aplicacion*, lo que quiere decir que la aplicación ha sido desplegada correctamente; esto se conoce como despliegue en caliente.

3.3.- Estructura de carpetas de una aplicación empresarial. Archivo EAR.

En el mundo Java EE tenemos tres posibles tipos de aplicaciones: aplicaciones web, objetos distribuidos EJBs y aplicaciones empresariales, que no son más que un conjunto de las dos anteriores aplicaciones.

Una aplicación empresarial Java EE (archivo .EAR) es un conjunto de módulos, siendo un módulo una aplicación web completa (empaquetada en un archivo .war) o conjunto de objetos distribuidos EJBs (empaquetados en un archivo .jar).

Podemos resumir que la estructura del archivo EAR es:

- ✓ `/*.war`: Archivos war.
- ✓ `/*.jar`: Archivos (ejb) jar.
- ✓ `/META-INF/application.xml`: Descriptor de despliegue del módulo EAR, en donde se dan de alta y se declaran el nombre y descripción de la aplicación que se despliega, y los diferentes módulos web y EJB que forman la aplicación.

Vamos a suponer una estructura lo más sencilla posible para una aplicación web como la siguiente, y que es la que constituye el archivo "*aplicacion.war*":

```
aplicacion/  
├── index.html  
└── WEB-INF  
    └── web.xml  
  
1 directory, 2 files
```

donde observamos una página estática "*index.html*" y un descriptor del despliegue "*web.xml*", a partir de esta estructura pretendemos construir nuestro propio archivo EAR que contendrá un solo archivo WAR con una página HTML estática.

Una vez situados en la carpeta "*aplicacion*", mediante el comando `# jar cvf aplicacion.war *` generaremos el archivo .WAR correspondiente a la aplicación; podremos comprobar que se trata de un formato similar a los archivos .zip probando a abrirlo con un programa compresor.

Para construir el archivo .EAR, como mínimo, tendremos que crear un descriptor de despliegue al que llamaremos "*aplicacion.xml*", para ello creamos una carpeta llamada "*temporal*" en donde situamos el archivo "*aplicacion.war*"; en la misma ruta creamos una carpeta llamada "*META-INF*" donde vamos a crear el descriptor; quedando la estructura del siguiente modo:

```
temporal/  
├── aplicacion.war  
└── META-INF  
    └── aplicacion.xml  
  
1 directory, 2 files
```

Nos situamos dentro de la carpeta "*temporal*" y creamos el archivo .ear mediante el comando:

```
# jar cvf aplicacion.ear *
```

y tendremos así el archivo .ear correspondiente a la aplicación creada.



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 cvf permite generar archivos .war.

- ☐ El comando #jar cvf permite generar archivos .ear.

- ☐ Un archivo .ear puede contener archivos .jar.

Mostrar retroalimentación

Solución

1. Incorrecto
2. Correcto
3. Correcto
4. Correcto
5. Correcto

4.- 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.



[The Apache Ant Project Team](#) (Apache 2.0)

A la hora de implantar dicha herramienta se han propuesto, además, documentar el procedimiento de instalación, configuración y puesta en funcionamiento de dicha herramienta.

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 aspectos y/o funciones de las que **Ant** se va a ocupar:

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

Es utilizado en la mayoría de los 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...; 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 unitarias, generar la documentación, empaquetar el programa...

Como limitaciones a tener en cuenta:

- ✓ Al ser una herramienta basada en XML, los archivos Ant deben ser escritos en XML.
- ✓ La mayoría de las antiguas herramientas, como `<javac>`, `<exec>` y `<java>` tienen malas configuraciones por defecto, valores para opciones que no son coherentes con las tareas más recientes.
- ✓ Cuando se expanden las propiedades en una cadena o un elemento de texto, las propiedades no definidas no son planteadas como error, sino que se dejan como una referencia sin expandir.

Para trabajar con **Ant** se necesita:

- ✓ JDK en versión 1.4 o superior, ya que Ant no deja de ser una aplicación Java.
- ✓ Un parser XML. Da igual cual, si se ha bajado la versión binaria de Ant no hay por qué preocuparse, porque ya incluye uno.

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



ANT (Another Neat Tool)

- 1.- ¿Qué es?
- 2.- ¿Para qué sirve?
- 3.- Ventajas.
- 4.- ¿Cómo funciona?

ANT (Another Neat Tool)

1. ¿Qué es?

- ✓ Es una herramienta que permite automatizar el proceso de ensamblado de aplicaciones web.
 - ➡ **Ensamblado = construcción + despliegue**
 - ➡ Similar a la herramienta make de Linux.

ANT (Another Neat Tool)

2.- ¿Para qué sirve?

- ✓ Se ocupa de:
 - ➡ Compilación.
 - ➡ Generar documentación.
 - ➡ Empaquetamiento.
 - ➡ Ejecución...

ANT (Another Neat Tool)

3.- Ventajas.

- ✓ Automatiza tareas, para que al final con un solo comando:
 - ➡ se puede compilar desde cero el proyecto
 - ➡ ejecutar pruebas unitarias
 - ➡ generar la documentación
 - ➡ empaquetar el programa...
- ✓ No depende de los comandos shell de cada sistema operativo, ya que se basa en archivos XML y clases Java, siendo idónea como solución multi-plataforma.

ANT (Another Neat Tool)

4.- ¿Cómo funciona?

- ✓ Funciona a partir de un script de ensamblado en formato XML llamado build.xml, definido en base a **proyecto**, **targets** y **tasks**.
 - ➡ Proyecto:
 - Uno por archivo y contiene targets.
 - ➡ Target:
 - Con un nombre y dependencias hacia otros targets.
 - Contiene un conjunto de **tasks**.
 - ➡ Tasks:
 - Operaciones básicas (javac, java, jar, etc...)



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](#)

4.1.- Instalación y configuración de Ant.

Vamos a partir de una máquina con el sistema operativo Ubuntu 20.04.LTS en donde realizaremos la instalación de **Ant**, en primer lugar comprobamos si tenemos instalado Java, podemos hacerlo empleando el siguiente comando:

```
# java -version
```



[S.alt](#) (CC BY-NC-ND)

recordemos que, como requisito para la instalación de Ant, es imprescindible una versión JDK 1.4 ó superior.

Posteriormente procederemos a la descargar del paquete binario de Ant, que podemos descargarlo de la siguiente forma:

```
# wget http://ant.apache.org/bindownload.cgi/apache-ant-1.8.2-bin.tar.gz
```

y una vez hemos descargado el archivo binario lo descomprimos empleando la instrucción:

```
# tar -zxvf apache-ant-1.8.2-bin.tar.gz
```

luego movemos la carpeta "*apache-ant-1.8.2*" creada a "*/usr/local*".

Lo único que falta es crear la variable **ANT_HOME** y actualizar la variable **PATH**.

- ✔ **ANT_HOME**: Indica el directorio raíz de instalación de Ant, de acuerdo a las instrucciones anteriores esta ruta sería : */usr/local/apache-ant-1.8.2*.
- ✔ **PATH**: Define la ruta de acceso para los binarios del sistema; la modificación de esta variable permite acceder a los ejecutables de Ant de cualquier directorio.

Podemos hacerlo agregando al archivo "*/etc/profile*" el siguiente contenido:

```
ANT_HOME=/usr/local/apache-ant-1.8.2/  
  
PATH=$PATH:$ANT_HOME/bin
```

y luego, para que el sistema recoja los cambios realizados, empleamos el comando:
#source /etc/profile.

Para comprobar que **ant** se ha instalado correctamente desde una consola de shell ejecutamos el comando siguiente: *#ant* y deberíamos obtener un mensaje similar a:

```
Buildfile: build.xml does not exist!  
Build failed
```

con lo que la herramienta **ant** estaría correctamente instalada y configurada para desempeñar su función en nuestra máquina.



Debes conocer

En el siguiente vídeo podemos ver que se muestra cómo realizar la instalación del paquete Ant en un equipo con sistema operativo Microsoft Windows 7.

[Resumen textual alternativo](#)

4.2.- El archivo build.xml.

Como hemos dicho, **Ant** se basa en ficheros XML, normalmente configuramos el trabajo a hacer con nuestra aplicación en un fichero llamado **build.xml**, así que vamos a ver algunas de las etiquetas con las que podemos formar el contenido de este archivo.

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

[Hubert.tw](#) (CC BY-NC-SA)

- ✔ **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 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. Ant incluye ya las más básicas, como son `BaseDir` para el directorio base de nuestro proyecto, `ant.file` para el path absoluto del fichero `build.xml`, y `ant.java.version` para la versión de la JVM.

Pasamos a ver un simple ejemplo de archivo **build.xml**:

```
<?xml version="1.0"?>
<project name="ProbandoAnt" default="compilar" basedir=".">
  <!-- propiedades globales del proyecto -->
  <property name="fuente" value="." />
  <property name="destino" value="classes" />
  <target name="compilar">
    <javac srcdir="${fuente}" destdir="${destino}" />
  </target>
</project>
```

Este sencillo fichero requiere poca explicación, simplemente declaramos el proyecto indicando, la acción a realizar por defecto (`default="compilar"`), e indicamos que el directorio base es el actual (`basedir="."`).

Después indicamos en sendas etiquetas **property** los directorios de origen y de destino (`property name="fuente" value="."` y `property name="destino" value="classes"`).

Por último declaramos un **target** llamado `compilar`, que es el que hemos declarado como por defecto.

En este objetivo tenemos una única tarea, la de compilación *javac*, a la que por medio de los atributos `srcdir` 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 es compilar nuestro código, así que, simplemente, estando situados en el directorio donde tenemos nuestro build.xml, desde una ventana de MS-DOS o terminal GNU/Linux, podemos hacer:

```
# [PATH_TO_ANT]ant
```

Esto funciona así porque hemos declarado compilar como el objetivo por defecto, aunque podría ser otro así que por regla general pondríamos:

```
# [PATH_TO_ANT]ant nombre_objetivo
```

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

4.3.- El objetivo .jar.

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:

```
<project name="Proyecto">
</project>
```

Ant, al igual que otras herramientas de construcción, se basa en el concepto de objetivos o targets cuya definición engloba tanto las dependencias previas como los pasos a seguir para conseguirlo.



[Wa7son](#) (CC BY-NC-ND)

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 **.jar** final. Para ello basta incluir dentro de `<project>` las siguientes líneas:

```
<target name="init">
  <mkdir dir="classes" />
  <mkdir dir="build" />
</target>
```

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

Si queremos alcanzar el objetivo **init** basta con realizar:

```
# ant init
Buildfile: build.xml
init:
    [mkdir] Created dir: /home/profesor/proyecto/classes
    [mkdir] Created dir: /home/profesor/proyecto/build
BUILD SUCCESSFUL
Total time: 0 seconds
```

Es hora de compilar nuestro proyecto, vamos a definir el objetivo **compile**. Ahora bien, la compilación depende de la creación del directorio "*classes*" que se realiza en el objetivo anterior. Con esto en cuenta basta con incluir:

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

La dependencia se fija en la declaración del `target` de tal manera que se garantiza su cumplimiento antes de comenzarla. Nuestro código está en el directorio "*src*" y el resultado de la compilación se lleva al directorio "*classes*".

Importante notar que esta vez estamos usando `<javac>` esto es lo que **Ant** denomina tarea. Hay muchas tareas predefinidas.

Con nuestro proyecto compilado vamos a generar el **.jar** que distribuiremos haciendo uso de un nuevo objetivo llamado **build**.

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

Comprobamos que hay una dependencia de *compile* y se utiliza la tarea **jar** que se encarga de empaquetar todo el contenido del directorio *classes* en el fichero **proyecto.jar**.

Finalmente incluiremos un nuevo objetivo para limpiar todo el entorno, el objetivo **clean**:

```
<target name="clean">
    <delete dir="classes" />
    <delete dir="build" />
</target>
```

Elimina los directorios de trabajo dejando el entorno limpio del proceso de compilación. Resumiendo nuestro fichero `build.xml` es:

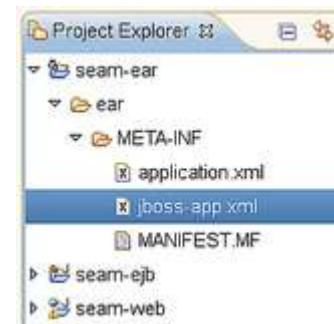
```
<project name="Proyecto">
  <target name="init">
    <mkdir dir="classes" />
    <mkdir dir="build" />
  </target>
  <target name="compile" depends="init">
    <javac srcdir="src" destdir="classes" />
  </target>
  <target name="build" depends="compile">
    <jar destfile="build/proyecto.jar" basedir="classes" />
  </target>
  <target name="clean">
    <delete dir="classes" />
    <delete dir="build" />
  </target>
</project>
```

4.4.- Despliegue de un archivo WAR.

En la arquitectura Java EE, los componentes web y los ficheros con contenido estático, como imágenes, son llamados **recursos web**. Un **módulo web** es la más pequeña unidad de un recurso web que se pueda utilizar y desplegar. Un módulo web Java EE corresponde con una **aplicación web**, como se define en la especificación de Java Servlet.

Además de los componentes web y los recursos web, un módulo web puede contener otros ficheros:

- ✓ Clases utilitarias del lado del servidor (👉 beans para bases de datos, carritos de compras y demás). A menudo estas clases cumplen con la arquitectura JavaBeans.
- ✓ Clases del lado del cliente (👉 applets y clases utilitarias).



Eclipse (EPL)

Un módulo web tiene una estructura específica. El directorio más alto de la jerarquía de directorios de un módulo web es el **raíz de documento** de la aplicación. Es donde las páginas JSP, clases y archivos del **lado del cliente**, y los recursos estáticos como imágenes, son almacenados.

El directorio raíz de los documentos contiene un subdirectorio llamado **WEB-INF**, que contiene los siguientes ficheros y directorios:

- ✓ **web.xml**: El descriptor de despliegue de aplicación.
- ✓ **classes**: Un directorio que contiene las clases del lado del servidor: componentes Servlets, clases utilitarias y JavaBean.
- ✓ **tags**: Un directorio que contiene ficheros de etiquetas, que son implementaciones de librerías de etiquetas.
- ✓ **lib**: Un directorio que contiene los archivos JAR de las librerías llamadas por las clases del lado del servidor.

Un módulo web debe ser empaquetado en un WAR en ciertos escenarios de despliegue y cuando se quiera distribuir el módulo web. Se empaqueta un módulo web en un WAR ejecutando el comando `jar` en un directorio ubicado en el formato de un módulo, utilizando la utilidad **Ant** o utilizando la herramienta IDE de su elección.

Un módulo web puede ser desplegado como una estructura de ficheros sin empaquetar o puede ser empaquetado en un fichero JAR conocido como un archivo web (WAR). Dado que el contenido y uso de los ficheros WAR difieren de aquellos ficheros JAR, el nombre del fichero WAR utiliza una extensión **.WAR**. El módulo web descrito es portátil, se puede desplegar en cualquier contenedor web que cumpla con la especificación Java Servlet.

Para desplegar un WAR en un servidor de aplicaciones, el fichero debe contener un **descriptor de despliegue** en tiempo de ejecución. El descriptor de despliegue es un fichero XML que contiene información como el contexto raíz de la aplicación web y la relación de los nombres portátiles de los recursos de aplicación a los recursos del servidor de aplicación.

Existen una serie de tareas para **Ant** que podemos utilizar para la gestión 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.
- ✓ <trycatch>: Evita que falle un `build` aunque falle alguna tarea.

Se pueden emplear diversos tipos de servidores de aplicaciones web junto con la herramienta **Ant**, por ejemplo JBoss/Wildfly o Tomcat.

Para desplegar un WAR con la herramienta Ant, abrimos una ventana de terminal o línea de comando en el directorio donde se ha construido y empaquetado el WAR y ejecutamos `ant deploy`.



Autoevaluación

Rellena los huecos con los conceptos adecuados:

En la arquitectura Java EE, los componentes web y los ficheros con contenido estático, como imágenes, son llamados [].
Un [] es la más pequeña unidad de un recurso web que se pueda utilizar y desplegar.

El [] es un fichero XML que contiene información como el contexto raíz de la aplicación web y la relación de los nombres portátiles de los recursos de aplicación a los recursos del servidor de aplicación.

Para desplegar un WAR con la herramienta Ant, abrimos una ventana de terminal o línea de comando en el directorio donde se ha construido y empaquetado el WAR y ejecutamos [].

Enviar

En la arquitectura Java EE, los componentes web y los ficheros con contenido estático, como imágenes, son llamados **recursos web**.
Un **módulo web** es la más pequeña unidad de un recurso web que se pueda utilizar y desplegar.

El **descriptor de despliegue** es un fichero XML que contiene información como el contexto raíz de la aplicación web y la relación de los nombres portátiles de los recursos de aplicación a los recursos del servidor de aplicación.

Para desplegar un WAR con la herramienta Ant, abrimos una ventana de terminal o línea de comando en el directorio donde se ha construido y empaquetado el WAR y ejecutamos **ant deploy**.

5.- El gestor de aplicaciones Web de Tomcat.



Caso práctico

En la empresa **BK programación** disponen de un servidor de aplicaciones web Tomcat. Debido a las opciones que éste proporciona han decidido profundizar en el funcionamiento de éste, pero centrándose en la administración de aplicaciones a desplegar desde la interfaz web que Tomcat proporciona, el “Gestor de Aplicaciones Web de Tomcat”.



Tomcat ([Apache](#))

Una vez arrancado en el equipo servidor el **Tomcat** mediante el script "*catalina.sh*" que se encuentra en la carpeta */bin* del directorio de instalación de Tomcat, en nuestro caso "*/opt/tomcat/*", desde un navegador podremos acceder a Tomcat mediante la URL:

- ✓ <http://localhost:8080> si accedemos desde la propia máquina en la que está corriendo Tomcat.
- ✓ http://ip_servidor:8080 si accedemos desde cualquier otra máquina de la red.



Tomcat ([Apache](#))

Mediante el enlace "Tomcat Manager" accedemos al gestor de aplicaciones Web de Tomcat. Esta página permite desplegar un proyecto contenido en un fichero de extensión **war**, como ya hemos visto en el punto "2.2 Despliegue de una aplicación web" de este tema, o simplemente copiar la carpeta que contiene de la aplicación a la carpeta **webapps** que se encuentra en el directorio de instalación de Tomcat.

Vamos al Tomcat Manager y allí podremos ver un listado de las aplicaciones web que hay disponibles en el servidor. Podemos comprobar, en nuestro caso, que si tenemos en la carpeta "*/opt/tomcat/webapps/*" la carpeta de la aplicación "*Aplic_Web*" que desarrollamos al principio de este tema, ya se mostraría en el listado que el gestor de aplicaciones de Tomcat nos ofrece, o simplemente accediendo desde un navegador a la URL: **http://ip_servidor:8080/nombre_aplicacion** (en el caso genérico), para nuestro caso podemos probar con http://localhost:8080/Aplic_Web.



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?

5.1.- Configuración del gestor.

Todos los ficheros de configuración se encuentran en la carpeta "*conf*" en la ruta de instalación de Tomcat. Esa ruta la referenciamos anteriormente con la variable de entorno **CATALINA_HOME**, es decir "*\$CATALINA_HOME/conf*". En esta ruta encontramos una carpeta denominada **catalina/localhost/** en donde se almacena la configuración web del Tomcat en dos archivos .xml: **host-manager.xml** y **manager.xml**.



Tomcat ([Apache](#))

Para realizar la administración del servidor desde el entorno web, instalaremos un paquete adicional, Tomcat6-admin, podemos hacerlo mediante la instrucción:


```
# apt-get install tomcat9-admin
```

y, para acceder a la administración, es necesario crear el rol "manager" y un usuario con dicho rol, para ello podemos seguir el siguiente procedimiento:

- ✓ Editamos el archivo de usuarios de Tomcat: `# nano $CATALINA_HOME/conf/tomcat-users.xml`.
- ✓ Añadimos las líneas estableciendo un contenido para y :

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

Los roles disponibles son:

- **manager-gui**: Acceso a la interfaz HTML.
- **manager-status**: Acceso sólo a la página "Server Status".
- **manager-script**: Acceso a la interfaz de texto plano y a la página "Server Status".
- **manager-jmx** — Acceso al proxy  JMX y a la página "Server Status".
- ✓ Reiniciamos el Tomcat y, a través de la URL `http://ip_servidor:8080/manager/html`, podemos desinstalar, recargar e instalar aplicaciones.
- ✓ Para habilitar el host-manager tendríamos que realizar los mismos pasos pero estableciendo el rol *admin*, y desde `http://ip_servidor:8080/host-manager/html` tendríamos el servicio operativo.

Podemos asegurar Tomcat estableciendo que se permita el acceso a este contexto únicamente a las direcciones IP de los equipos desde los que operan los administradores, esto lo podemos configurar en el archivo: "*\$CATALINA_HOME/work/Catalina/localhost/manager/context.xml*".

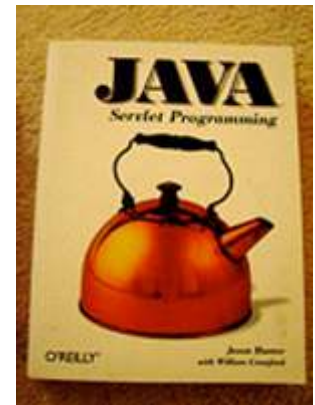
```
<Context path="/manager" privileged="true" antiResourceLocking="false" docBase="/opt/apach
  <Valve className="org.apache.catalina.valves.RemoteAddrValve" allow="127.0.0.1,direcc
</Context>
```

Para acceder a la administración de Tomcat es necesario crear el rol "manager", "admin" y usuario/s con dicho rol, para ello es necesario editar **el archivo de usuarios de Tomcat** `$CATALINA_HOME/conf/Tomcat-users.xml`.

5.2.- Conexión al gestor de aplicaciones web de Tomcat de forma remota.

Un servidor Apache-Tomcat consta de 3 componentes principales:

- ✓ **Catalina:** es el contenedor de Servlet de Tomcat. Implementa las especificaciones de Sun para servlets y Java Server Pages (JSP).
- ✓ **Coyote:** es el conector HTTP que soporta el protocolo HTTP1.1 para el servidor web o para el contenedor de aplicaciones.
Coyote escucha las conexiones entrantes en un puerto TCP determinado y redirige las peticiones al motor Tomcat para así procesar las peticiones y mandar una respuesta de vuelta al cliente.
- ✓ **Jasper:** es el motor JSP de Tomcat; compila las páginas JSP en código java en servlets que puedan ser manejados por Catalina.
En tiempo de ejecución, cualquier cambio en un archivo JSP Jasper lo detecta y lo recompila.



[Ellecer \(CC BY-ND\)](#)

Los modos de operación de Tomcat pueden ser:

1. Servidor de aplicaciones:
 - ✓ Tomcat necesita un servidor que actúe como frontend (Apache, IIS...).
 - ✓ El contenido estático es servido por el frontend.
 - ✓ Las peticiones a servlets y JSPs son redirigidas a Tomcat por el servidor web.
 - ✓ Recibe peticiones en protocolos específicos como AJP que son enviados por el frontend.
2. Standalone:
 - ✓ No hay un servidor web que actúe de frontend.
 - ✓ Todos los contenidos son servidos por Tomcat.
 - ✓ Recibe peticiones HTTP.

Los conectores son los componentes que proporcionan la interfaz externa al servidor, concretamente el conector HTTP1.1 basado en Coyote es el conector por defecto para Tomcat. Los conectores se definen en el archivo:

\$CATALINA_HOME/conf/server.xml , aquí tenemos un ejemplo:

```
<Conector port="8080"
  protocol="HTTP/1.1"
  maxThreads="150"
  connectionTimeout="2000"
  redirectPort="8443"/>
```

debido a establecer medidas de seguridad para conexiones web al servidor, podremos configurar para un conector HTTP/1.1 con SSL lo siguiente:

```
<Conector port="8080"
  protocol="HTTP/1.1"
  maxThreads="150"
  scheme="https"
  secure="true"
  clientAuth="false"
  sslProtocol="TLS"/>
```

en donde vemos que se han establecido los atributos `scheme` para el protocolo, y `secure` para establecer que se trata de un conector SSL.



Autoevaluación

Rellena los huecos con los conceptos adecuados:

Un servidor Apache-Tomcat consta de 3 componentes principales:

- ✓ : es el contenedor de Servlet de Tomcat. Implementa las especificaciones de Sun para servlets y Java Server Pages (JSP).
- ✓ : es el conector HTTP que soporta el protocolo HTTP1.1 para el servidor web o para el contenedor de aplicaciones. Coyote escucha las conexiones entrantes en un puerto determinado y dirige las peticiones al motor para así procesar las peticiones y mandar una respuesta de vuelta al cliente.
- ✓ : Es el motor JSP de ; compila las páginas JSP en código java en servlets que puedan ser manejados por .

Los modos de operación de Tomcat pueden ser y .

Enviar

Un servidor Apache-Tomcat consta de 3 componentes principales:

- ✓ **Catalina**: es el contenedor de Servlet de Tomcat. Implementa las especificaciones de Sun para servlets y Java Server Pages (JSP).
- ✓ **Coyote**: es el conector HTTP que soporta el protocolo HTTP1.1 para el servidor web o para el contenedor de aplicaciones. Coyote escucha las conexiones entrantes en un puerto **TCP** determinado y dirige las peticiones al motor **Tomcat** para así procesar las peticiones y mandar una respuesta de vuelta al cliente.
- ✓ **Jasper**: Es el motor JSP de **Tomcat**; compila las páginas JSP en código java en servlets que puedan ser manejados por **Catalina**. Los modos de operación de Tomcat pueden ser **Servidor de aplicaciones** y **Standalone**.

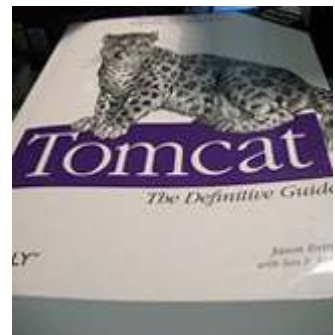


5.3.- Incluir tareas Ant en Tomcat.

Como ya hemos visto anteriormente, **Ant** es una herramienta de construcción de software que permite automatizar tareas repetitivas en el proceso de compilación, enlazado, despliegue, etc.

Tomcat define una serie de librerías que le permiten automatizar tareas como el despliegue y repliegue de aplicaciones web, mediante **Ant**.

Para integrar las dos herramientas anteriores podemos seguir las siguientes operaciones:



[Patrick Johanneson \(CC BY-NC-SA\)](#)

- ✓ Descargar Ant.
- ✓ Descomprimir el fichero.
- ✓ Configurar las variables de entorno `ANT_HOME` para que apunte a la raíz de la distribución.
- ✓ Configurar la variable `PATH` para añadir la ruta hasta el directorio `<ANT_HOME>/bin`.
- ✓ Copiar el fichero `<Tomcat_HOME>/lib/catalina-ant.jar` en `<ANT_HOME>/lib`.

Para instalar una aplicación web, se le indica a Tomcat Manager que un nuevo contexto está disponible, empleando para ello el comando `#ant install` que funciona tanto con archivos `.WAR` como si se indica la ruta al directorio de la aplicación no empaquetada. Es necesario tener en cuenta que el comando anterior no implica un despliegue permanente; si se reinicia Tomcat las aplicaciones previamente instaladas no van a estar disponibles.

Despliegue permanente de aplicaciones web:

- ✓ Sólo funciona con archivos `*.WAR`.
- ✓ No se pueden desplegar directorios no empaquetados.
- ✓ Se sube `*.WAR` al Tomcat y se arranca.
- ✓ Permite el despliegue remoto.
- ✓ Un contenedor web remoto no puede acceder al directorio de la máquina local.

El comando `#ant deploy` se emplea para el despliegue permanente de las aplicaciones, y para ello es necesario:

- ✓ Que el Tomcat Manager se esté ejecutando en la localización especificada por el atributo `url`.
- ✓ El despliegue de una aplicación en el contexto especificado por el atributo `path` y la localización contenida en los archivos de la aplicación web especificada con el atributo `war`.

Podemos establecer el siguiente ejemplo:

```
<deploy url="http://localhost:8080/manager"
  path="mywebapp"
  war="file:/path/to/mywebapp.war"
  username="username" password="password" />
```

El archivo **build.xml** de una aplicación llamada "Hola" para "ant deploy" podría ser el siguiente:

```
<target name="deploy" description="Deploy web application"
        depends="build">
    <deploy url="${url}" username="${username}"
            password="${password}"
            path="${path}" war="file:${build}/${example}.war"/>
</target>
<taskdef name="deploy"
        classname="org.apache.catalina.ant.DeployTask"/>
<property name="url" value="http://localhost:8080/manager"/>
<property name="path" value="/${example}"/>
<property name="example" value="hola" />
```