

Resumen tema 3

Despliegue de Aplicaciones Web

5 de diciembre de 2022

CIFP Carlos III - Cartagena

Santiago Francisco San Pablo Raposo

2º curso DAW

Contenido

# Índice de ilustraciones.

# Índice de tablas.

Resumen del tema 3.

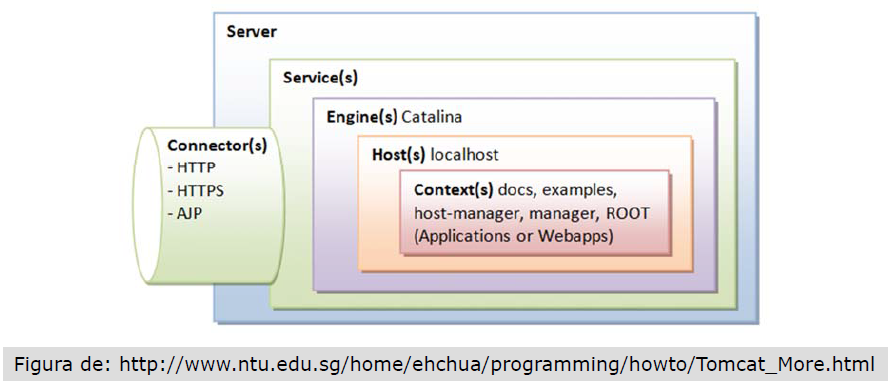
Se deben evaluar los siguientes parámetros a la hora de poner en funcionamiento un servidor de aplicaciones:

* **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, 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 de:
  + El número de conexiones simultáneas que se pueden establecer.
* **Herramientas de automatización de tareas** en el servidor (Ant, etc.).

# 1.- Arquitectura y configuración básica del servidor de aplicaciones.

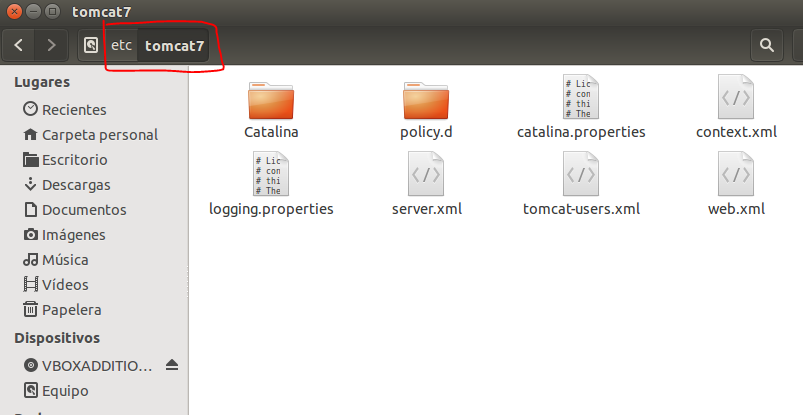
Un servidor de aplicaciones está relacionado con el concepto de sistema distribuido. Un **sistema distribuido mejora tres aspectos fundamentales** en una aplicación:

* **La alta disponibilidad**: un sistema debe estar funcionando las 24 horas del día los 365 días al año.
  + Para poder alcanzar esta característica es necesario el uso de técnicas de balanceo de carga y de recuperación ante fallos (failover).
* **La escalabilidad**: capacidad de hacer crecer un sistema cuando se incrementa la carga de trabajo (el número de peticiones).
  + **Por ejemplo**: si tenemos una tienda que incrementa la demanda de servicio, debemos ser capaces de incorporar nuevas máquinas para dar servicio.
* **El mantenimiento**: versatilidad a la hora de actualizar, depurar fallos y mantener un sistema.
  + La solución al mantenimiento es la construcción de la lógica de negocio en unidades reusables y modulares.



Estructura de directorios

Tomcat es un contenedor de servlets. Ejecuta servlets, y convierte páginas JSP y JSF en servlets. Puede utilizarse como un servidor de aplicaciones Web con HTML, servlets y JSPs, también como complemento al servidor Apache.



**Localización de la estructura de directorios**:

* cd /usr/share/tomcat9
* cd /var/lib/tomcat9
  + **Conf de Tomcat**: cd /etc/tomcat9

**Árbol de directorios**:

|  |  |
| --- | --- |
| Directorio | Descripción |
| Bin | contiene los binarios y scripts de inicio de Tomcat. |
| **Se localiza en**: /usr/share/tomcat9 |
| Conf | La configuración global de Tomcat. |
| **Está en**: /var/lib/tomcat9/conf. 🡪 enlace simbólico 🡪 /etc/tomcat9. |
| Tiene algunos archivos que merece la pena destacar. |

**Archivos de conf** (están en /var/lib/tomcat9/conf o /etc/tomcat9 salvo que se especifique lo contrario):

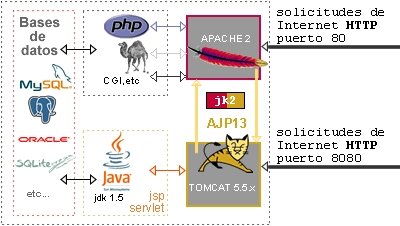
* **catalina.policy**: contiene la política de seguridad relacionada con Java e impide que los Servlets o JSPs la sobrescriban por motivos de seguridad.
  + **Se localiza en**: /etc/tomcat9/policy.d/03catalina.policy
* **catalina.properties**: Contiene los archivos .JAR que no pueden sobrescribirse por motivos de seguridad y otros de uso común.
* **context.xml**: Se utiliza principalmente para informar de dónde se puede encontrar el archivo web.xml de las propias aplicaciones.
  + Contiene la configuración que será común a todos los elementos Context.
  + La configuración específica de cada context (dentro de cada virtualhost, por cada aplicación), vendrá dada por su propio archivo.
* **logging.properties**: Establece las políticas generales para el registro de actividad del servidor, aplicaciones o paquetes.
* **server.xml**: es el **fichero principal de configuración** de Tomcat y tiene mucho que ver con su arquitectura.
* **tomcat-users.xml**: Contiene los usuarios, contraseñas y roles usados para el control de acceso.
  + Es el archivo donde se encuentra la información de seguridad para las aplicaciones de administración de Tomcat.
  + Todos los valores son por defecto así que deben cambiarse en caso de “descomentar” las líneas de configuración.
* **web.xml**: un descriptor de despliegue por defecto con la configuración compartida por todas las aplicaciones
  + Es un archivo con directivas de funcionamiento de las aplicaciones.
  + Además de todos estos archivos, **existe un subdirectorio para cada motor con un subdirectorio localhost donde irá otro archivo de contexto específico para cada aplicación**. Este archivo tiene la forma nombre-de-aplicación.xml.
    - **En este caso el motor es Catalina** así que están en Catalina/localhost. Puedes ver que hay uno por cada paquete adicional que instalamos, documentación, ejemplos y aplicaciones de administración.
* **lib.xml**: continen todos los .JAR comunes a todas las aplicaciones.
  + Aquí van archivos de Tomcat, APIs de JSPs, etc y en él podemos ubicar archivos comunes a las diferentes aplicaciones como MySQL JDBC
  + Está en /usr/share/tomcat9/lib.
* **logs**: aquí van los archivos de registro.
  + Se localiza en el siguiente enlace simbólico: /var/log/tomcat9.
* **temp**: este directorio es opcional. Por defecto, no está creado ni activado.
  + Se usa para los archivos temporales que necesita Tomcat en su ejecución.
* **webapps**: aquí se ubican las aplicaciones propiamente dichas.
  + Por defecto, solo hay una que se denomina ROOT. Se pueden instalar más.
  + Las que hemos instalado de **documentación, ejemplos y administradores aparte se encuentran en su propio directorio** en /usr/share/tomcat9-admin, tomcat9-docs y tomcat9-examples.

PowerPoint

Descripción generada automáticamente

* **work**: es un directorio para los archivos en uso, cuando se compilan los JSPs, etc.

## 1.1.- Arquitectura de aplicación web.

Además de la estructura de directorios, **la arquitectura de los archivos de configuración se define por una estructura XML** que sigue la organización del servidor de aplicaciones (server.xml).

|  |  |
| --- | --- |
| Directiva | Descripción |
| Server | Es el primer elemento superior y **representa una instancia de Tomcat**. Es el **equivalente al servidor en sí** con un puerto asociado. |
| **Pueden existir varios en diferentes puertos** y a veces **se hace para que si una aplicación** falla arrastrando al servidor, **esto no afecte a otras aplicaciones**. |
| <Server port="-1" shutdown="SHUTDOWN"> |
| Service | El **servicio** **agrupa un contenedor de tipo Engine** con un conjunto de conectores y otros contenedores. |
| **El motor suele ser Catalina** y los **conectores** por defecto HTTP y AJP. |
| **<Service name="Catalina">** |
| Connector | Los conectores **sirven para comunicar las aplicaciones con clientes** (por ejemplo, un navegador u otros servidores). |
| Representan el **punto donde se reciben las peticiones y se les asigna un puerto IP en el servidor**. |
| Containers | Tomcat llama contenedores a:   * Engine (nivel más alto) * Host * Context (nivel más bajo) * Cluster |
| **El de nivel más alto es Engine y el más bajo Context**. Algunos componentes como Realm o Valve pueden ubicarse dentro de un contenedor. |
| Engine | El motor **procesa las peticiones** y es un componente que representa el motor de Servlets Catalina. |
| Examina las cabeceras HTTP para determinar a qué host (o virtual host) o context se le debe pasar cada petición. |
| * **Cuando Tomcat se utiliza como servidor autónomo**: se usa el motor por defecto. * **Cuando Tomcat se usa dando soporte a un servidor web**: el Engine se sobrescribe porque el servidor web ya ha determinado el destino correcto para las peticiones. |
| **Un motor puede tener**:   * O bien, **un grupo de hosts** que representan un grupo de aplicaciones web. * O bien, **Context** que representa a una única aplicación. |
| Host | **Define un host por defecto o múltiples hosts virtuales** en Tomcat. |
| **En Tomcat los hosts virtuales se diferencian por nombres de dominio distintos**, por ejemplo www.aplicacion1.es y www.aplicacion2.es. Cada uno soporta varios Context. |
| Context | Este elemento es **equivalente a una aplicación web**. Hay que informar al motor y al host de la localización de la carpeta raíz de aplicación. |
| También se puede **habilitar la recarga dinámica** (dynamic reload) para que al modificar alguna clase de la aplicación se modifique en la ejecución. Esta opción carga mucho el servidor por lo que se recomienda para pruebas pero no en producción. |
| En un contexto también **se pueden establecer páginas de error específicas** para armonizarlas con la apariencia de la aplicación. |
| **Puede** contener parámetros de inicio para **establecer control de acceso en la aplicación**. |
| Cluster | **En caso de que tengamos más de un servidor Tomcat atendiendo las peticiones**, este elemento nos permite configurarlo. |
| Es capaz de replicar las sesiones y los parámetros de cada Context. |
| Queda muy por encima del contenido del curso. |
| Realm | Se puede aplicar al nivel de Engine, Host o Context. |
| Se utiliza **para autenticación y autorización de usuarios o grupos**. |
| **Se pueden usar con**:   * Archivos de texto. * Servidores LDAP. * Bases de datos. |
| Valve | Se usa para interceptar peticiones antes de pasárselas a las aplicaciones. Afecta al tiempo de respuesta a la petición. |
| **Esto nos permite pre-procesar las peticiones** para:   * Bloquear algunas, registrar accesos… * Registrar detalles de la conexión (en archivos log) * Establecer un único punto de acceso para todas las aplicaciones de un host o para todos los hosts de un servidor. |
| **Puede establecerse para cualquier contenedor**: Engine, Host, and Context, y/o Cluster. Hay un concepto similar en los servlets que se denomina Filtros (Filters). |

**Para saber más**: el servidor de aplicaciones está definido en server.xml

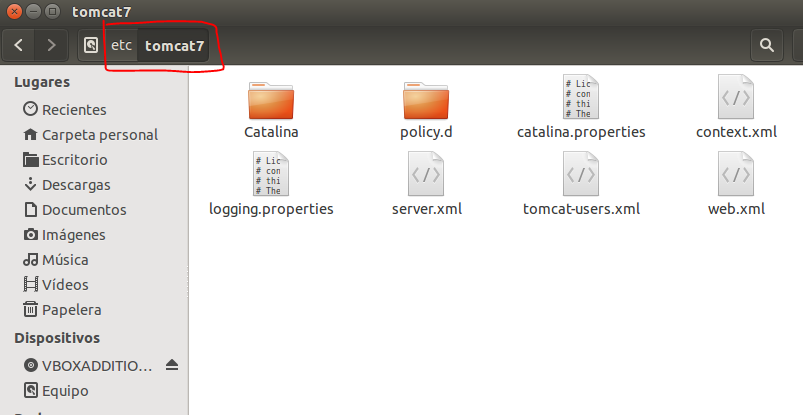
## 1.2.- Configuración básica del servidor de aplicaciones.

**Orientación para el alumnado**: en una configuración estándar de un servidor de aplicaciones, este contiene los archivos necesarios para el funcionamiento y configuración de la aplicación:

* **Los ejecutables de la aplicación**,
* Los archivos de configuración de la aplicación y
* Los archivos de configuración de escritorio (declaración de acciones, imágenes, ayuda).

En Tomcat, **la configuración se realiza a través de** uno o más **ficheros XML**. Los principales son: **server.xml, context.xml y web.xml**.

Tomcat busca estos archivos en el directorio especificado por $CATALINA\_BASE, en un subdirectorio /conf que es un enlace a /etc/tomcat9.



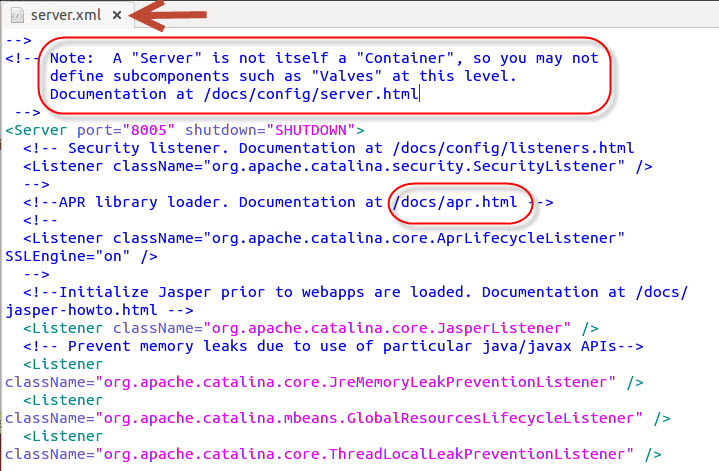
### 1.2.1.- Server.xml (descriptor de servidor).

En el archivo de configuración por defecto podemos ver que se establece un único servicio y una única instancia del servidor.

**¿Qué puede contener el elemento <Server>?**

|  |  |
| --- | --- |
| Elemento | Descripción |
| <Service> | Un grupo de conectores asociados con un motor. Es necesario al menos uno. |
| <Listener> | Clases que **escuchan y manejan eventos** que tienen que ver con el **ciclo de vida del servidor**, por ejemplo después de arrancar. |
| <GlobalNaming Resources> | Recursos globales que pueden ser usados en esta instancia del servidor por los componentes que los necesiten, por ejemplo una base de datos. |

**Para saber más**: En los archivos de configuración de Tomcat podemos ver continuas referencias a los archivos de ayuda de cada uno de los elementos de Tomcat.



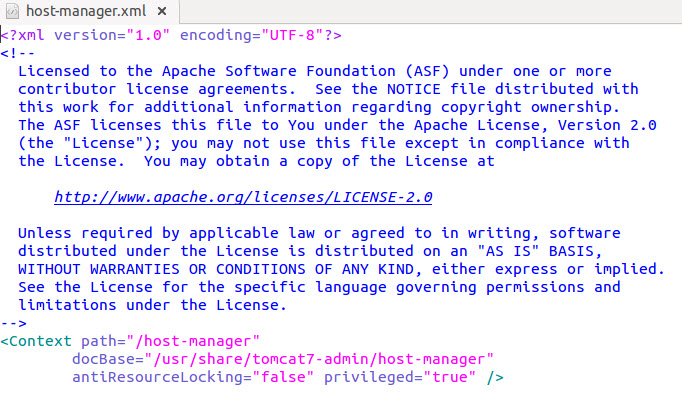
|  |  |
| --- | --- |
| Elemento | Descripción |
| <Engine> | El motor es quien procesa las peticiones. |
| El nombre es el que le demos a la instancia y defaultHost indica a qué host virtual se le pasará una petición en caso de que no se especifique ninguno |
| **Aclaración**: el mismo motor puede procesar peticiones dirigidas a múltiples hosts virtuales de los especificados en este archivo. |
| <Engine name="Catalina" defaultHost="localhost"> |
| **Un motor puede contener**:   * Uno o más <Host> * Uno o ningún <Context> * Uno o ningún <Realm> * Varios o ningún <Valve> y <Listener> |
| <Realm> | **Mecanismo de seguridad que sirve para autenticar usuarios y establecer seguridad** a nivel de contenedor. |
| **Por defecto, Tomcat lee los usuarios del archivo tomcat-users.xml**, pero sería mejor configurarlo para que usara una BB.DD o un servidor LDAP. |
|  |
| <Host> | Este elemento representa un host virtual en Tomcat. |
| **La configuración por defecto solo define localhost**.  Si tenemos configurado el servidor DNS con un nombre para nuestro servidor usaremos éste. |
| **El atributo appBase establece el directorio raíz de las aplicaciones**. Se establece a partir de <CATALINA\_BASE> si no se indica lo contrario. |
| Por defecto la URL de cada aplicación es la resultante de añadir su directorio raíz a la del servidor. |
| En nuestro ejemplo hemos instalado cuatro aplicaciones docs, examples, host-manager y manager. También se establece ROOT que indica la aplicación por defecto si no añadimos otras cada una en su directorio. |
| El **atributo unpackWARs** indica si este tipo de archivos debe ser descomprimido. En caso de no descomprimirlos, la ejecución será un poco más lenta. |
| El **atributo autoDeploy** indica si el despliegue de una aplicación que situemos en el directorio debe ser automático o no. |
| <Host name="localhost" appBase="webapps" unpackWARs="true" autoDeploy="true"> |
| Además, **tiene un <Valve> para registrar los accesos**: |

### 1.2.2.- Context.xml (descriptor de contexto).

**El context en Tomcat** se puede establecer a muchos niveles y **se aplicará el más específico en cada aplicación**.

**Los descriptores de contexto** de **administración de cada aplicación** **se encuentran en**: $CATALINA\_BASE/conf/nombre\_motor/nombre\_host.

En nuestro caso están en /etc/tomcat9/Catalina/localhost. **Por ejemplo** el de host-manager:



* **Atributo path**: indica cómo se accederá a la aplicación en nuestro servidor.
  + El problema para cambiarlo es que habría que modificar todos los archivos xml relacionados para que Tomcat siga encontrando la aplicación.
* **Atributo docBase**: es el directorio de despliegue de la aplicación.
  + Los **descriptores de contexto context.xml** específicos de cada aplicación web **están en el directorio de cada aplicación en /META\_INF**.
    - El de la misma aplicación está en /usr/share/tomcat9-admin/host-manager/META-INF.

**Context.xml de /etc/tomcat9** **contiene los parámetros que vayan a ser comunes a todas las aplicaciones**. Si no se sobrescriben en algún contexto más concreto se aplicarán los genéricos.

### 1.2.3.- web.xml (descriptor de despliegue) (descriptor de aplicación)

Hablamos a nivel de aplicación web. Cualquier **aplicación web** en Java debe tener un descriptor de despliegue.

Igual que en el descriptor de contexto, pueden establecerse varios y se aplicarán las directivas más concretas.

**Localización de cada descriptor**: dentro de la cerpeta /WEB-INF de cada aplicación web.

**Por ejemplo**:

* **Concreto**: /usr/share/tomcat9-admin/host-manager/WEB-INF.
* **Común**: a todas las aplicaciones en /etc/tomcat/web.xml

# 2.- Despliegue de aplicaciones en Tomcat.

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.

Una **aplicación web** **puede ser** **desplegada en diferentes servidores web** manteniendo su funcionalidad y **sin necesidad de ningún tipo de modificación en su código** debido a la especificación servlet 2.2.

Las aplicaciones web se organizan 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 (jar de cada aplicación).
  + **META-INF**: contiene el archivo de contexto context.xml
  + **Resto de carpetas para ficheros estáticos**.

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

* **Por medio de archivos WAR**.
* **Manualmente**: 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**.
  + Carpeta con el nombre y ruta del proyecto, que contiene los ficheros que forman la interfaz (HTML, js, css…)
* **bin**: contiene las clases de java de la aplicación
* **src**
* **tomcat**
* **gwt-cache**.

¿Cómo desplegar una aplicación en Tomcat

1. Copiar la carpeta 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 la siguiente estructura de carpetas dentro de la carpeta del proyecto**:
   * **La carpeta WEB-INF** (respetando las mayúsculas). Dentro de WEB-INF:
     + **lib**: a esta carpeta hay que copiar todas las librerías (.jar) que necesite la aplicación para su funcionamiento.
     + **classes**: aquí deberá copiarse el contenido de la carpeta bin.
4. **Crear en WEB-INF** **el fichero web.xml**, con las rutas de los servlets utilizados en la aplicación.
5. **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.

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.

El servidor de aplicaciones Tomcat cuenta con una serie de ejemplos, tanto de servlets como de JSP.

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

1. **Crearemos una carpeta** que será el **árbol** de nuestro proyecto. **Por ejemplo**:



1. **La aplicación** que pretendemos desarrollar **contiene un archivo al que llamaremos index.jsp**.

<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 APLICACIONES");}

</script></head>

<body><h1 align=center>DESPLIEGUE DE APLICACIONES WEB</h1>

<div align=center>

<form><input type="button" value="UNIDAD 3"

onclick="popup()"></form>

</body></html>

1. 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, 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, sin perder funcionalidad y sin necesidad de modificar su código.

**Los WARs simplemente son archivos Java como los JARs, pero para las aplicaciones web**.

Antes de la especificación Servlet 2.2, era bastante diferente desplegar servlets entre diferentes contenedores de servlets.

**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.

Siguiendo 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/web-app\_2\_5.xsd"

version="2.5">

<display-name>Descriptor Aplicacion Aplic\_Web</display-name>

<description>

Mi primer descriptor web.xml.

</description>

</web-app>

**Ubicación del archivo**: en la carpeta WEB-INF perteneciente a la aplicación en desarrollo.

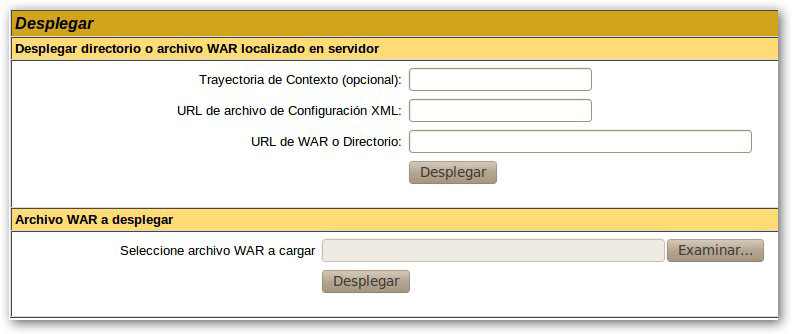


**Para generar el archivo .WAR:** una vez hemos terminado:

**javac -d WEB-INF/classes \*.java** este comando tiene como finalidad la compilación de las clases Java de nuestra aplicación. En nuestro caso no hay clases java.

**jar cvf Aplic\_Web.war .** para crear el archivo .WAR con el nombre Aplic\_Web, posteriormente al desplegar la aplicación será el nombre que nos aparezca como carpeta dentro de webapps.

**Para desplegar el archivo .WAR**: visita <http://127.0.0.1/8080>, y en el apartado “Administration”, ve a “Tomcat Manager”:



**Para saber más**: [sobre el funcionamiento, configuración, instalación, administración, etc. del servidor de aplicaciones Tomcat](Esta%20web%20muestra,%20de%20forma%20amplia,%20el%20funcionamiento,%20configuración,%20instalación,%20administración,%20etc.%20del%20servidor%20de%20aplicaciones%20Tomcat,%20donde%20también%20podemos%20encontrar%20cómo%20desplegar%20aplicaciones.), y cómo desplegar aplicaciones.

## 2.3.- Implementar el registro de acceso.

**Caso práctico**: vamos a ver cómo realizar un seguimiento, de manera que se pueda comprobar los accesos que han tenido, en qué momento y qué recursos son más demandados.

Para conseguir obtener y poder configurar los registros de acceso a un servidor de aplicaciones Tomcat, mencionar **las válvulas de registro** de acceso de Tomcat.

**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**.

**¿Dónde está?** En el directorio /etc/tomcat9 podemos ver **dos archivos que hacen referencia a estos registros**:

* El primero es **logging.properties y** el segundo **server.xml**.
* **Ambos hacen referencia al directorio $CATALINA\_BASE/logs** que en nuestra instalación se encuentra como enlace simbólico en /var/lib/tomcat9/logs.

**En este directorio podemos ver 4 tipos de archivos de registro**:

1. **catalina.fecha.log**: Estos archivos guardan la actividad del motor durante un día determinado.
2. **catalina.out**: es un compendio de los anteriores.
3. **localhost.fecha.log**: guardan la actividad del sitio.
4. **localhost\_access\_log.fecha.txt**: son registros de acceso a las aplicaciones del servidor durante un día. Son los que vamos a ver. Aparecen configurados en el archivo server.xml dentro del elemento <Host>

**Explicación de la etiqueta Valve que aparece en server.xml principal**:

<Valve className="org.apache.catalina.valves.AccessLogValve" directory="logs" prefix="localhost\_access\_log." suffix=".txt" pattern="%h %l %u %t "%r" %s %b"/>

**AccessLogValve** registra el acceso. Como está incluida en el elemento <Host> registra todos los accesos a localhost.

**Relacionado con <Valve> se encuentra <Filter>**. Estas dos tecnologías sirven para interceptar las peticiones y respuestas de HTTP y procesarlas para realizar algún tipo de acción antes de que sigan su camino.

Una **gran ventaja** es que **no dependen de cada aplicación** sino que pueden implementarse para todo el sitio.

* **Valve**: es un desarrollo asociado a Tomcat.
* **Filter**: pertenece a la API de Servlets.

**Se pueden colocar en <Engine>, <Host> o <Context>** afectando al entorno concreto según corresponda.

<Filter>

**Los filtros son una interfaz de los Servlets**, por lo tanto **no son exclusivos de Tomcat**. Tienen un comportamiento muy **similar a las válvulas** pero **se configuran en** el descriptor de despliegue de **cada aplicación**.

Aunque podemos implementar nuestros propios filtros que implementen la interfaz, **Tomcat incluye algunos** que pueden consultarse en la página de la documentación correspondiente.

**Para saber más**: [Documentación oficial en Tomcat sobre Filter](https://tomcat.apache.org/tomcat-7.0-doc/config/filter.html).

## 2.4.- Sesiones persistentes.

**Caso práctico**: nuestro cliente nos solicita 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.

**Las sesiones activas** por parte de clientes a aplicaciones web **en Tomcat, por defecto, están configuradas para mantenerse en caso de posibles pérdidas de conexión** con el servidor o posibles reinicios de este; 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.

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

**Para configurar las sesiones persistentes**: se configuran en el context de la aplicación web. Tendremos que gestionar el elemento <Manager> como un subelement de <Context>

**Podremos actuar** a nivel de aplicación concreta o a nivel del servidor (todas las apps).

* **De forma global**: en <CATALINA\_HOME>/conf/context.xml
* **Cada aplicación**: adaptar el archivo <CATALINA\_HOME>/conf/context.xml correspondiente a la aplicación.

**Ejemplo de configuración** (se emplean comentarios para explicar cada uno de los parámetros):

<Context>

<!-- classname especifica la clase del servidor que implementa el gestor, es recomendable utilizar el org.apache.catalina.session.PersistentManager -->

<Manager className="org.apache.catalina.session.PersistentManager"

<!--saveOnRestart=true para indicar que se guarden todas las sesiones al reiniciar el servidor -->

saveOnRestart="true"

<!--maxActiveSession cuando se supera el límite aquí establecido se comienzan a enviar a disco las nuevas sesiones. Se establece un valor -1 para indicar ilimitadas sesiones-->

maxActiveSession="3"

<!--minIdleSwap establece el número mínimo de segundos que transcurren antes de que una sesión pueda copiarse al disco duro -->

minIdleSwap="0"

<!--maxIdleSwap indica el número máximo de segundos que transcurren antes de que una sesión pueda copiarse al disco duro -->

maxIdleSwap="60"

<!--maxIdleBackup para indicar el número de segundos desde que una sesión estuvo activa por última vez hasta que se envíe al disco. La sesión no es eliminada de memoria. Permite restauración de la sesión en caso de caída del servidor. -->

maxIdleBackup="5">

<!--Store indica cómo y donde almacenar la sesión, están disponibles las siguientes implementaciones: org.apache.catalina.session.FileStore y org.apache.catalina.session.JDBCStore -->

<Store className="org.apache.catalina.session.FileStore"/>

</Manager>

</Context>

# 3.- Construcción y despliegue automático con Ant.

**Caso práctico**: hemos pensado en la automatización del proceso con la ayuda de Ant, para la realización de tareas mecánicas y repetitivas, normalmente durante la fase de compilación y construcción.

ANT (“Another Neat Tool”, "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, 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 una herramienta que permite automatizar el proceso de ensamblado de aplicaciones web:

* Ensamblado = construcción + despliegue.
* Es similar al "*make*" empleado en Linux, pero desarrollado en Java; posee la ventaja de ser una solución multiplataforma de manera mucho más sencilla.

**Funciones de Ant**:

* Compilación.
* Generación de documentación.
* Empaquetamiento.
* Ejecución, etc.

**Ventajas**:

* **Automatiza tareas**, para que al final con un solo comando, puedas:
  + Compilar desde cero tu 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.

**Características**:

* **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)
* **Además, es fácilmente extenisble e integrable con muchas herramientas de 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...; **Todo esto lo hace Ant por ti de forma automatizada a golpe de clic**.

**Limitaciones**:

* **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.

**Requisitos**:

* **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.

**Cómo 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**: Para empezar a trabajar con Ant: [Apache Ant - Welcome](https://ant.apache.org/).

## 3.1.- Instalación y configuración de Ant.

1. Comprobamos que tenemos Java instalado (mínimo versión JDK 1.7).

java -version

1. Descargamos el paquete binario de Ant:

http://ant.apache.org/bindownload.cgi o bien  
wget http://apache.rediris.es//ant/binaries/apache-ant-1.9.6-bin.tar.gz

1. Una vez hemos descargado el archivo binario lo descomprimimos empleando la instrucción:

tar -zxvf apache-ant-1.9.6-bin.tar.gz

1. Luego movemos la carpeta "apache-ant-1.9.6" creada a "/usr/local".
2. **Creamos la variable ANT\_HOME** y actualizamos 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-1.9.6.
   * **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 de la siguiente manera, agregando al archivo /etc/profile el siguiente contenido:

ANT\_HOME=/usr/local/apache-ant-1.9.6/

PATH=$PATH:$ANT\_HOME/bin

1. Actualizamos el sistema para que recoja los cambios realizados en las variables del entorno.

source /etc/profile

1. Para comprobar que ant se ha instalado correctamente, desde una consola de Shell, ejecutamos el comando 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**: [vídeo sobre cómo realizar la instalación del paquete Ant en Windows 7](https://www.youtube.com/watch?v=bcY4ZF1jt4o).

## 3.2.- El archivo build.xml.

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

Etiquetas que podemos usar para formar el contenido de este archivo:

* **project**: Este es el elemento raíz del fichero XML, se corresponde a nuestra aplicación Java.
* **target**: un target 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 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.

**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>

**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 task, 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}.

**Cómo compilar el código una vez tenemos nuestro build.xml**:

1. Estar situados en el directorio donde tenemos nuestro build.xml
2. Ejecutar el siguiente comando:

[PATH\_TO\_ANT]ant

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

[PATH\_TO\_ANT]ant nombre\_objetivo

[PATH\_TO\_ANT]ant compilar

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

### 3.2.1.- Otro ejemplo de build.xml.

**Debes conocer**: [Manual de Ant](https://es.wikibooks.org/wiki/Manual_b%C3%A1sico_de_ANT).

<project name="MyProject" default="dist" basedir=".">

<description>

simple example build file

</description>

<property name="src" location="src"/>

<property name="build" location="build"/>

<property name="dist" location="dist"/>

<target name="init">

<tstamp/>

<mkdir dir="${build}"/>

<mkdir dir="${src}"/>

</target>

<target name="compile" depends="init" description="compile the source " >

<javac srcdir="${src}" destdir="${build}"/>

</target>

<target name="dist" depends="compile" description="generate the distribution" >

<mkdir dir="${dist}/lib"/>

<jar jarfile="${dist}/lib/MyProject-${DSTAMP}.jar" basedir="${build}"/>

</target>

<target name="clean" description="clean up" >

<delete dir="${build}"/>

<delete dir="${dist}"/>

</target>

</project>

**La primera línea** se corresponde con el **proyecto** que compone el buildfile, en ella pueden aparecer los siguientes atributos:

* **name** = "MyProject": identifica el nombre del proyecto
* **default** = "dist": identifica el objetivo que se ejecuta por defecto, en caso de no especificarle uno en concreto.
* **basedir** = ".": directorio base sobre el que vamos a trabajar. En este ejemplo se trabajaría sobre el directorio actual.

A continuación, aparecen un **listado de propiedades**, su declaración se compone del literal property y los atributos name para especificar el nombre y location para asignarles un valor.

<property name="src" location="src"/>

Lo siguiente que aparece en el ejemplo son los **objetivos**. El primero de ellos es **init**:

<target name="init">

<tstamp/>

<mkdir dir="${build}"/>

</target>

Como se mencionó anteriormente, un **objetivo está compuesto por una o varias tareas**, en este caso tenemos las tareas **tstamp** y **mkdir**, las cuales muestran la fecha y crean el directorio descrito por la propiedad build respectivamente

El **siguiente objetivo** que aparece es **compile**.

<target name="compile" depends="init" description="compile the source " >

<javac srcdir="${src}" destdir="${build}"/>

</target>

En la declaración del objetivo **podemos ver los siguientes atributos**:

* **name** = "compile": identifica el nombre del objetivo
* **depends** = "init": identifica el objetivo que se debe ejecutar antes de lanzarse el actual.
  + En este caso se debe ejecutar previamente el objetivo init (mostrar la fecha del sistema y crear un directorio) antes de ejecutar este objetivo.
* **description** = "compile the source": breve descripción de lo que hace el objetivo.

El **cuerpo del objetivo está formado por la tarea javac**, que toma los ficheros fuente ubicados en la ruta indicada por el atributo srcdir y los compila en destdir.

El **siguiente objetivo** del buildfile es **dist**:

<target name="dist" depends="compile" description="generate the distribution" >

<mkdir dir="${dist}/lib"/>

<jar jarfile="${dist}/lib/MyProject-${DSTAMP}.jar" basedir="${build}"/>

</target>

Antes de lanzar sus tareas se debe haber ejecutado **compile** para ejecutarse.

En este objetivo, aparecen:

* La tarea **mkdir** descrita anteriormente
* La tarea **jar**, esta tarea construye un jar con el nombre MyProject concatenando la fecha del sistema (DSTAMP) a partir de los compilados en el directorio indicado por la propiedad build.

Por último, tenemos el objetivo **clean**:

<target name="clean" description="clean up" >

<delete dir="${build}"/>

<delete dir="${dist}"/>

</target>

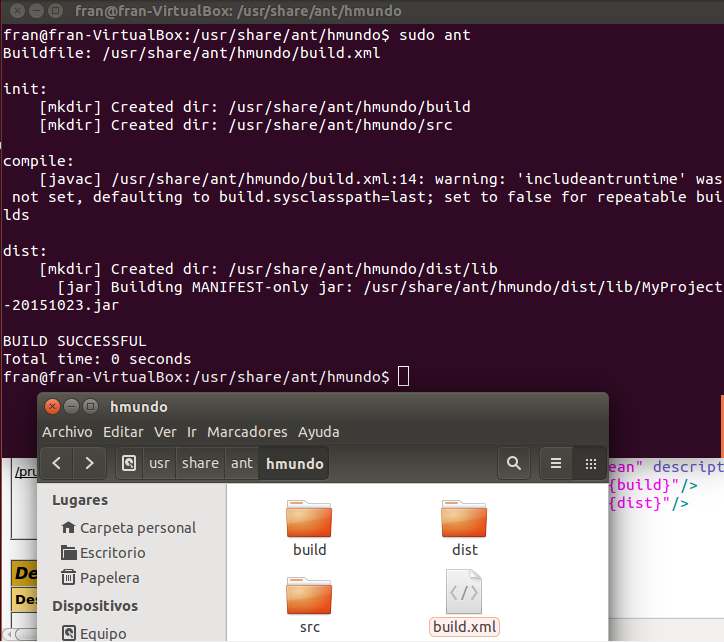
Compuesto por 2 tareas **delete**. Simplemente, borra los directorios temporales necesarios para la creación del jar.

Si repasamos la cabecera del proyecto, vemos que el orden lógico de ejecución de este build.xml es:

1. **Ejecutar objetivo init**
   * mostrar fecha de sistema
   * crear directorio build
2. **Ejecutar objetivo compile**
   * compilar fuentes de src y guardarlos en build
3. **Ejecutar objetivo dist**
   * crear directorio lib
   * crear jar a partir de los compilados

El objetivo **clean** no depende del resto, y tampoco se especifica en la cabecera del proyecto, por lo que **se debe referenciar de forma explícita en la línea de comandos** para que se ejecute.

**Para ejecutarlo**: simplemente escribimos ant donde se encuentre el fichero build.xml y veremos el resultado en el mismo directorio.



## 3.3.- El archivo .jar.

Vamos a explicar este apartado mediante un ejemplo.

1. **Creamos un archivo build.xml** en la raíz de nuestro proyecto y definimos su nombre:

<project name="Proyecto">

</project>

1. Vamos a comenzar **definiendo un objetivo de preparación llamado init** que cree un directorio classes y un directorio build para el .jar final.

<target name="init">

<mkdir dir="classes" />

<mkdir dir="build" />

</target>

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

#ant init

Buildfile: build.xml

init:

[mkdir] Created dir: /home/profesor/proyecto/classes

[mkdir] Created dir: /home/profeosr/proyecto/build

BUILD SUCCESSFUL

Total time: 0 seconds

1. Es hora de **compilar** nuestro proyecto vamos a definir el objetivo **compile**. La compilación depende de la creación del directorio “classes” que e realiza en el objetivo anterior. Bastaría incluir [javac](https://www.ibm.com/docs/es/developer-for-zos/9.5.1?topic=code-using-ant-javac-adapter):

<target name="compile" depends="init">

<javac srcdir="src" destdir="classes" />

</target>

La dependencia se fija en la declaración del target. Nuestro código está en el directorio "src" y el resultado de la compilación se lleva al directorio "classes".

1. 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>

Jar se encarga de empaquetar todo el contenido del directorio classes en el fichero proyecto.jar.

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

<target name="clean">

<delete dir="classes" />

<delete dir="build" />

</target>

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>

## 3.4.- 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 **componentes web** y **recursos web**, un módulo 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)

Un módulo web tiene una estructura específica. **En la raíz** es donde las **páginas JSP, clases y archivos del lado del cliente**, además de los **recursos estáticos** son **almacenados**.

El directorio raíz de los documentos contiene un subdirectorio llamado **WEB-INF**, que contiene los siguientes archivos 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**.

**Cómo se empaqueta en un archivo WAR**:

* Utilizando el mismo comando jar en un directorio ubicado en el formato de un módulo.
* Utilizando la utilidad Ant
* Utilizando la herramienta IDE de su elección.

**Formas de desplegar un módulo web**:

* Como una estructura de ficheros sin empaquetar.
* **Empaquetado en un fichero JAR conocido como un archivo web (WAR)**.

El contenido y uso de los **ficheros WAR** difieren de aquellos ficheros JAR. El módulo web descrito es portátil, **se puede desplegar en cualquier contenedor web que cumpla con la especificación Java Servlet**.

**Requisitos para que un WAR pueda ser desplegado en un servidor de aplicaciones**:

* El fichero debe contener un **descriptor de despliegue** **en tiempo de ejecución**.
  + 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**.

**Tareas Ant que podemos utilizar para la gestión de aplicaciones**:

* **<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.

**Debes conocer**: [manual de Ant](https://es.wikibooks.org/wiki/Manual_b%C3%A1sico_de_ANT).

**Para saber más**:

* [Primeros pasos con ant (chuidiang.org)](https://www.chuidiang.org/java/herramientas/ant.php)
* [Just Sherekan – Blog de Programación » Introducción a Ant (archive.org)](http://web.archive.org/web/20100412055151/http:/sherekan.com.ar/2009/06/21/introduccion-a-ant/)
* [Manual de Ant](http://web.archive.org/web/20100412055151/http:/sherekan.com.ar/2009/06/21/introduccion-a-ant/).

### 3.4.1.- Un ejemplo.

Vamos a ver un ejemplo sobr el uso de ANT.

Para ello tenemos los archivos build.xml, .java, web.xml y el index.jps en un directorio para crear la aplicación. La ejecución de ANT hará que se ejecute el archivo build.xml 'construyendo' el archivo WAR correspondiente a nuestra aplicación y lo copiará, finalmente, en el directorio desde donde Tomcat podrá desplegarlo.

# 4.- El gestor de aplicaciones Web de Tomcat.

**Tomcat se inicia mediante el script** “catalina.sh” localizado en la carpeta /usr/share/tomcat9/bin/catalina.sh

1. 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](http://ip_servidor:8080%20) si accedemos desde cualquier otra máquina de la red.
2. 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**, o simplemente copiar la carpeta que contiene de la aplicación a la carpeta **webapps de Tomcat**.
   1. En el Tomcat Manager podemos ver un listado de las aplicaciones web que hay disponibles en el servidor.
   2. Si tenemos en la carpeta /var/lib/t*omcat9/webapps/* la carpeta de la aplicación "*Aplic\_Web*" que desarrollamos al principio de este tema, ya se mostraría en el listado.

**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? **Pues en mi opinión, prefiero en WAR**.

## 4.1.- Configuración del gestor.

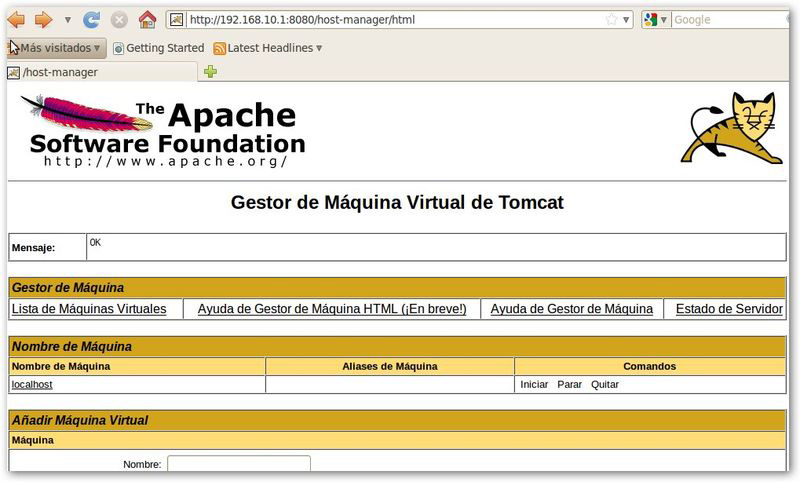
Todos los ficheros de configuración se encuentran en la carpeta conf en la ruta de instalación de Tomcat. Esa ruta está referenciada con la variable de entorno **CATALINA\_HOME**, es decir, $CATALINA\_HOME/conf.

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**.

**Para realizar la administración del servidor** desde el entorno web, debemos tener instalado el paquete correspondiente. Podemos hacerlo desde la consola, o con un gestor de paquetes, como ya vimos.

**Para acceder a la administración**, es necesario crear el rol “manager” y un usuario con dicho rol.

* **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 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: /etc/tomcat9/Catalina/localhost/context.xml.

<Context path="/manager" privileged="true" antiResourceLocking="false" docBase="/usr/share/tomcat7-admin/manager">

<Valve className="org.apache.catalina.valves.RemoteAddrValve" allow="127.0.0.1,direccion\_ip1,direccion\_ip2"/> </Context>

## 4.2.- 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 e implementa las especificaciones también para 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.
  + **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.

**Modos de operación de Tomcat**:

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"

maxTherads="150"

connectionTimeout="2000"

redirectPort="8443"/>

**Si queremos 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"

maxTherads="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.

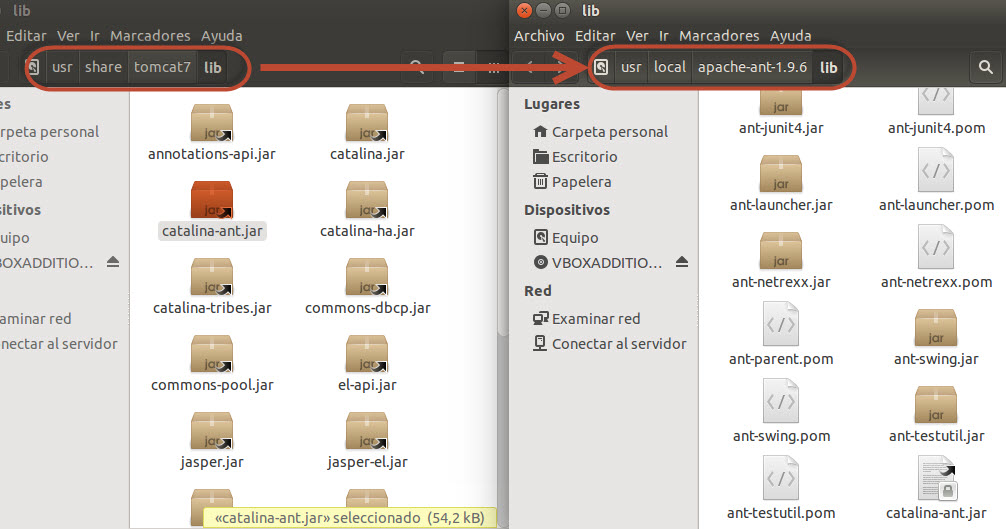
## 4.3.- Incluir tareas Ant en Tomcat.

Como ya hemos visto, **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 Ant con Tomcat**:

1. Descargar Ant.
2. Descomprimir el fichero.
3. Configurar las variables de entorno **ANT\_HOME** para que apunte a la raíz de la distribución.
4. Configurar la variable **PATH** para añadir la ruta hasta el directorio <ANT\_HOME>/bin.
5. 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

sudo ant install

que funciona tanto con archivos .WAR como si se indica la ruta al directorio de la aplicación no empaquetada. Aunque, solo con este comando, desplegaremos temporalmente la aplicación: cuando Tomcat se reinicie, la aplicación dejará de estar desplegada.

**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**.

**Por 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" />