

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, 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.



Materiales formativos de F.P. Online propiedad del Ministerio de Educación, Cultura y Deporte.

[Aviso Legal](#)

1.- Arquitectura y configuración básica del servidor de aplicaciones

Caso práctico

María, debe conocer la arquitectura del servidor de aplicaciones que ha montado sobre el sistema Ubuntu, pues sobre Tomcat los miembros de **BK programación** puedan desplegar, en dicho servidor, las aplicaciones web que consideren necesarias.



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, la escalabilidad y el mantenimiento. Vamos a ver estas características con ejemplos.

- La **alta disponibilidad** hace referencia a que 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** es la capacidad de hacer crecer un sistema cuando se incrementa la carga de trabajo (el número de peticiones). Cada máquina tiene una capacidad finita de recursos y por lo tanto sólo puede servir un número limitado de peticiones. Si, por ejemplo, tenemos una tienda que incrementa la demanda de servicio, debemos ser capaces de incorporar nuevas máquinas para dar servicio.
- El **mantenimiento** tiene que ver con la 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.

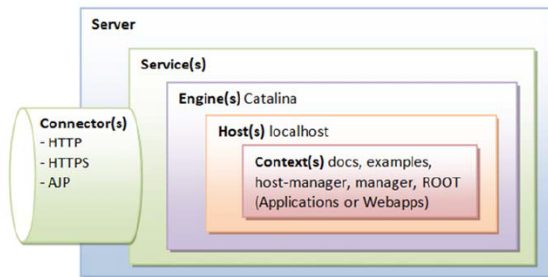


Figura de: http://www.ntu.edu.sg/home/ehchua/programming/howto/Tomcat_More.html

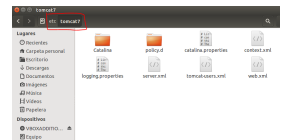
Estructura de directorios

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

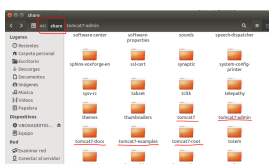
Los directorios que se incluyen en una instalación de Tomcat se ubican generalmente en el directorio donde hemos descomprimido Tomcat. Sin embargo, en esta instalación están definidos en dos ubicaciones: `/usr/share/tomcat7` y `/var/lib/tomcat7`. Sin embargo hay otro directorio afectado en la instalación `/etc/tomcat7`

Los directorios comunes son:

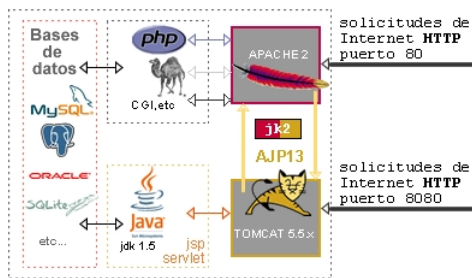
- `bin`: contiene los binarios y scripts de inicio de Tomcat.
- `conf`: la configuración global de Tomcat. En esta instalación es un enlace simbólico a `/etc/tomcat7`. Tiene algunos archivos que merece la pena destacar.



- `catalina.policy`: Este archivo contiene la política de seguridad relacionada con Java e impide que los Servlets o JSPs la sobrescriban por motivos de seguridad. En esta instalación se encuentra en `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`: El archivo de contexto común a todas las aplicaciones. 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.
- `logging.properties`: Establece las políticas generales para el registro de actividad del servidor, aplicaciones o paquetes.
- `server.xml`: ya hemos visto que es el fichero principal de configuración de Tomcat y que 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.
- `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`: contienen 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.
- `logs`: aquí van los archivos de registro. En esta instalación es un enlace simbólico a `/var/log/tomcat7`
- `temp`: este directorio es opcional. En esta instalación no está creado ni activado por defecto. Se usa para los archivos temporales que necesita Tomcat durante su ejecución.
- `webapps`: aquí se ubican las aplicaciones propiamente dichas. Ahora solo hay una que se denomina ROOT. Podría haber más pero como hemos instalado la documentación, ejemplos y administradores aparte se encuentran en su propio directorio en `/usr/share/tomcat7-admin`, `tomcat7-docs` y `tomcat7-examples`.
- `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 se define por una estructura XML que sigue la organización del servidor de aplicaciones.

Server: Es el primer elemento superior y representa una instancia de Tomcat. Es 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.

Service: El servicio agrupa un contenedor de tipo Engine con un conjunto de conectores. El motor suele ser Catalina y los conectores por defecto HTTP y AJP.

Connector: Los conectores sirven para comunicar las aplicaciones con clientes (por ejemplo un navegador web u otros servidores). Representan el punto donde se reciben las peticiones y se les asigna un puerto IP en el servidor.

Containers: Tomcat se refiere a Engine, Host, Context, y Cluster como contenedores. 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 (por ejemplo de las tramas HTTP) para determinar a 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 se sobrescribe porque el servidor web ya ha determinado el destino correcto para las peticiones.

Un motor puede contener Hosts que representan un grupo de aplicaciones web o Context que representa a una única aplicación. Tomcat se puede configurar con un único host o múltiples hosts virtuales como Apache.

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. Pueden usarse con archivos de texto, servidores LDAP o bases de datos por ejemplo.

Valve: Se usa para interceptar peticiones antes de pasárselas a las aplicaciones. Esto nos permite pre-procesar las peticiones para bloquear algunas, registrar accesos, registrar detalles de la conexión (en archivos log), o establecer un único punto de acceso para todas las aplicaciones de un host o para todos los hosts de un servidor. Afectan al tiempo de respuesta a la petición.

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

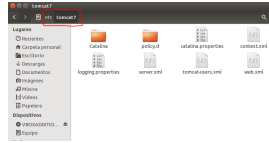
El servidor de aplicaciones está definido en [server.xml](#)

1.2. Configuración básica del servidor de aplicaciones

Orientaciones para el alumnado

En una configuración estándar de un servidor de aplicaciones, éste contiene todos los archivos necesarios para el funcionamiento y configuración de la aplicación. Estos archivos serán: los ejecutables de la aplicación, los archivos de configuración de la aplicación y 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/tomcat7.



server.xml

En el archivo de configuración por defecto podemos ver que se establece un único servicio y una única instancia del servidor. El elemento <Server> tiene la siguiente forma, escuchando por un puerto, que por seguridad no es el 8080, en el que el que se inicia una instancia del servidor (JVM) y en el que escucha por si llegan señales de apagado (shutdown):

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

El elemento <Server> puede contener otros tres:

- <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.
- <GlobalNamingResources>: 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 podemos ver continuas referencias a los archivos de ayuda de cada uno de los elementos de Tomcat

I. Server.xml

<Service>

El propósito de un servicio es agrupar un motor que procese las peticiones con uno o más conectores que gestionen los protocolos de comunicación. El servicio por defecto es el motor Catalina.

```
<Service name="Catalina">
```

Hemos dado al servicio el nombre del motor, pero no es necesario. Un <Service> contiene al menos un <Connector> y solo un <Engine> que es obligatorio.

<Connector>

Este elemento tiene mucho que ver con los dos modos de funcionamiento de Tomcat:

- Como servidor único: En el que Tomcat realiza las funciones de servidor de aplicaciones y de servidor web.
- Como servidor de aplicaciones: En el que Tomcat colabora con un servidor web que hace de frontend. El servidor web dirige todas peticiones de JSPs y Servlets a Tomcat.

```
<Connector port="8080" protocol="HTTP/1.1"
connectionTimeout="20000"
URIEncoding="UTF-8"
redirectPort="8443" />
```

En un entorno con acceso público se suele usar la segunda configuración ya que el servidor web está mucho más preparado en términos de seguridad y privacidad. Los dos conectores más comunes son HTTP y AJP. El segundo es usado para conectar con los servidores en el modo colaborativo (Apache u otros). Ambos pueden funcionar con SSL para mejorar la seguridad.

El puerto por defecto para HTTP es 8080. Se puede cambiar y si por ejemplo Tomcat va a estar en producción como un servidor en solitario podríamos modificarlo al puerto estándar de HTTP, 80.

II. Server.xml

<Engine>

El motor es quien procesa las peticiones realmente. 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 ya que el mismo motor puede procesar peticiones dirigidas a múltiples hosts virtuales de los especificados en este archivo.

```
<Engine name="Catalina" defaultHost="localhost">
```

Un motor contiene uno o más <Host>, uno o ningún <Context>, uno o ningún <Realm>, multiples <Valve> y <Listener> aunque puede no tener ninguno.

<Realm>

Es un mecanismo de seguridad que sirve para autenticar usuarios y establecer seguridad a nivel de contenedor. La configuración por defecto hace que Tomcat lea los usuarios del archivo tomcat-users.xml pero obviamente sería mejor configurarlo para que se usara una base de datos o servidor LDAP.

```
<Realm className="org.apache.catalina.realm.LockOutRealm">
  <!-- This Realm uses the UserDatabase configured in the global JNDI
  | resources under the key "UserDatabase". Any edits
  | that are performed against this UserDatabase are immediately
  | available for use by the Realm. -->
  <Realm className="org.apache.catalina.realm.UserDatabaseRealm"
    resourceName="UserDatabase"/>
</Realm>
```

<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 o no. En caso de no descomprimirlos, la ejecución será un poco más lenta.

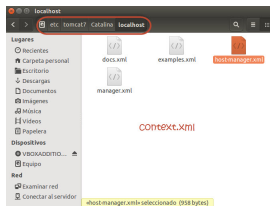
autoDeploy indica si el despliegue de una aplicación que situemos en el directorio debe ser automático o no.

```
<Host name="localhost" appBase="webapps" unpackWARs="true" autoDeploy="true">
```

Además contiene un <Valve> para registrar los accesos:

```
  <!-- Access log processes all example.
  | Documentation at: /docs/config/valve.html
  | Note: The pattern used is equivalent to using pattern="common" -->
  <Valve className="org.apache.catalina.valves.AccessLogValve"
    directory="logs"
    prefix="localhost_access_log." suffix=".txt"
    pattern="%h %l %u %t &quot;%r&quot; %s %b" />
```

I. Context.xml



El contexto en Tomcat se puede establecer a mucho 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/tomcat/Catalina/localhost. Por ejemplo el de host-manager

```
host-manager.xml x
<?xml version="1.0" encoding="UTF-8"?>
<!--
Licensed to the Apache Software Foundation (ASF) under one or more
contributor license agreements. See the NOTICE file distributed with
this work for additional information regarding copyright ownership.
The ASF licenses this file to You under the Apache License, Version 2.0
(the "License"); you may not use this file except in compliance with
the License. You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

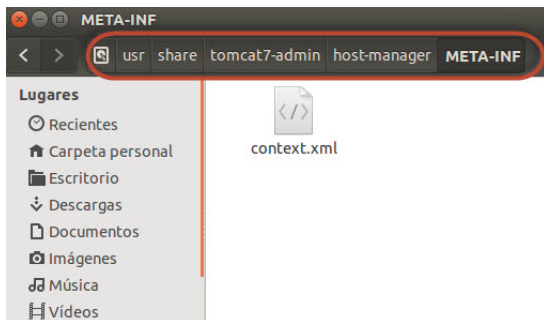
Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the license for the specific language governing permissions and
limitations under the License.
-->
<Context path="/host-manager"
docBase="/usr/share/tomcat7-admin/host-manager"
antiResourceLocking="false" privileged="true" />
```

El 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.

docBase es el directorio de despliegue de la aplicación.

Los descriptores de contexto específicos de cada aplicación web están en el directorio de cada aplicación en /META-INF

El de la misma aplicación está en /usr/share/tomcat7-admin/host-manager/META-INF

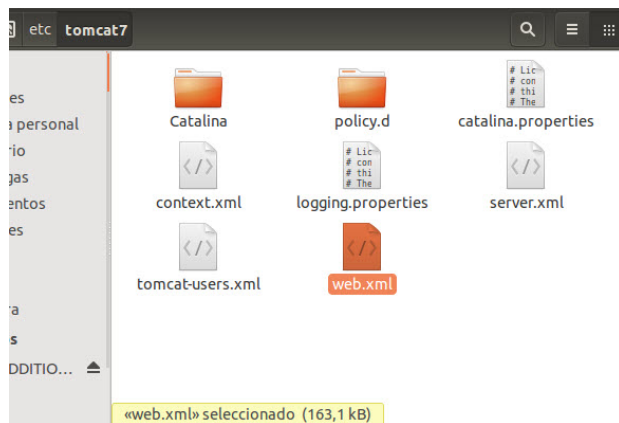


Context.xml contiene los parámetros que vayan a ser comunes a todas las aplicaciones. Si no se sobrescriben en algún contexto más concreto se aplicarán los genéricos.

```
<Context>
<WatchedResource>WEB-INF/web.xml</WatchedResource>
</Context>
```

I. web.xml

Cualquier aplicación web en Java debe tener un descriptor de despliegue. Como sucedía con el contexto, puede establecerse varios y se aplicarán las directivas más concretas. Cada aplicación tiene su propio descriptor y va en /WEB-INF
Por ejemplo /usr/share/tomcat7-admin/host-manager/WEB-INF. Existe uno común a todas las aplicaciones en /etc/tomcat/web.xml



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.

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 (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**, **bin**, **src**, **tomcat**, **gwt-cache**.

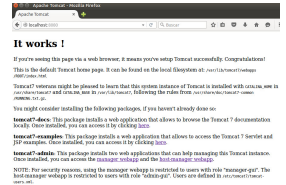
La carpeta **www** contiene a su vez una carpeta, con el nombre y ruta del proyecto, que contiene los ficheros que forman la interfaz (**HTML**, **js**, **css**...). La carpeta **bin** contiene las clases de java de la aplicación.

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 14.10 en la cual tenemos el servidor Tomcat.

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.

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.

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 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>
</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 comúnmente 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.

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>
```

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:



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. 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.

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:



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.



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.

En el directorio /etc/tomcat7 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/tomcat7/logs.

En ese directorio podemos ver cuatro 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>

```
<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. Una gran diferencia entre ellas es que <Valve> es un desarrollo asociado a Tomcat mientras que <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.

Citas para pensar

"Los sistemas nuevos generan problemas nuevos."

Ley de Murphy.

Para saber más

[Documentación oficial en Tomcat sobre Filter](#)

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.



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 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 -->
    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

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

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



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?



Resumen textual alternativo

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](#)

3.1.- Instalación y configuración de Ant.

Vamos a partir de una máquina con el sistema operativo Ubuntu 14.10 en donde realizaremos la instalación de **Ant (versión 1.9.6)**, en primer lugar comprobamos si tenemos instalado Java, podemos hacerlo empleando el siguiente comando:

```
#java -version
```

recordemos que, como requisito para la instalación de Ant, es muy recomendable una versión JDK 1.7 ó superior.

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

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

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

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

luego movemos la carpeta "*apache-ant-1.9.6*" 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-1.9.6*.
- ✓ **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 agregando al archivo "*/etc/profile*" el siguiente contenido:

```
ANT_HOME=/usr/local/apache-ant-1.9.6/  
  
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:

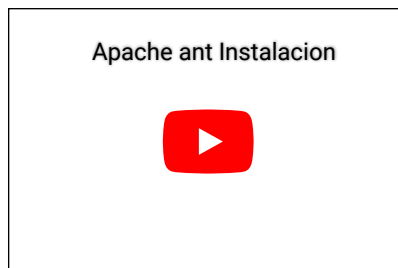
```
Buildfile: build.xml does not exist!  
Build failed
```

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



Debes conocer

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



[Resumen textual alternativo](#)

3.2.- 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>
```

- ✓ **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
```

```
#[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**.

Debes conocer

<https://es.wikibooks.org/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 esta 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 siguiente 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 que compone el 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>
```

En este caso, **dist** depende del objetivo **compile** para ejecutarse, luego, antes de lanzar sus tareas se debe haber ejecutado **compile**. En este objetivo aparece la tarea **mkdir** descrita anteriormente y **jar**, esta tarea construye un jar con el nombre MyProject concatenado con la fecha del sistema (DSTAMP) a partir de los

compilados en el directorio indicado por la propiedad *build*.

Por ultimo 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 por defecto llama al objetivo **dist**, el cual depende de **compile** y este a su vez de **init**, por lo que el orden lógico de ejecución de este build.xml es:

Ejecutar objetivo **init**

- mostrar fecha de sistema
- crear directorio build

Ejecutar objetivo **compile**

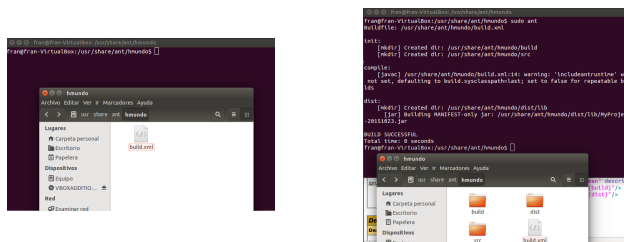
- compilar fuentes de src y guardarlos en build

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 explicita 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 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.



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>
```

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 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).

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.

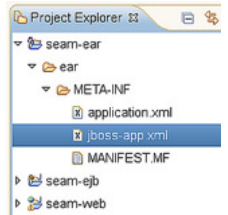
Un módulo web puede ser desplegado como una estructura de ficheros sin empaquetar o puede ser empaquetado en un fichero JAR conocido como un archivo web (WAR). Dado que el contenido y uso de los ficheros WAR difieren de aquellos ficheros JAR, el nombre del fichero WAR utiliza una extensión **.WAR**. 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 o Tomcat.



Debes conocer

- [Manual básico de ANT](#)

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 .

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**.

Para saber más

- [Primeros pasos con ANT](#)
- [ANT otros ejemplos](#)
- [Introducción a ANT](#)

3.4.1. Un ejemplo

Vamos a ver un ejemplo sobre el uso de ANT.

Para ello tenemos los archivos build.xml, .java, web.xml y el index.jsp 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.

Ficheros del ejemplo

[Ficheros para la aplicación práctica.](#)

4.- El gestor de aplicaciones Web de Tomcat.

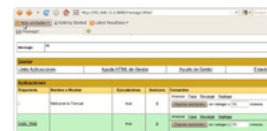
Caso práctico

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".



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 "*/etc/tomcat7/*", 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.



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** que se encuentra en el directorio de instalación de Tomcat.

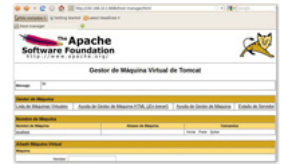
Vamos al Tomcat Manager y allí podremos ver un listado de las aplicaciones web que hay disponibles en el servidor. Podemos comprobar, en nuestro caso, que si tenemos en la carpeta */var/lib/tomcat7/webapps/* la carpeta de la aplicación "*Aplic_Web*" que desarrollamos al principio de este tema, ya se 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?

- ✔ 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.



```
<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.

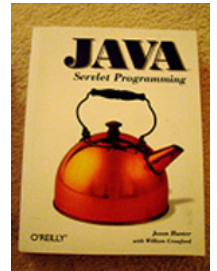
Un servidor Apache-Tomcat consta de 3 componentes principales:

- ✓ **Catalina**: es el contenedor de Servlet de Tomcat. 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.



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 redirige 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 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**.

Los modos de operación de Tomcat pueden ser **Servidor de aplicaciones** y **Standalone**.



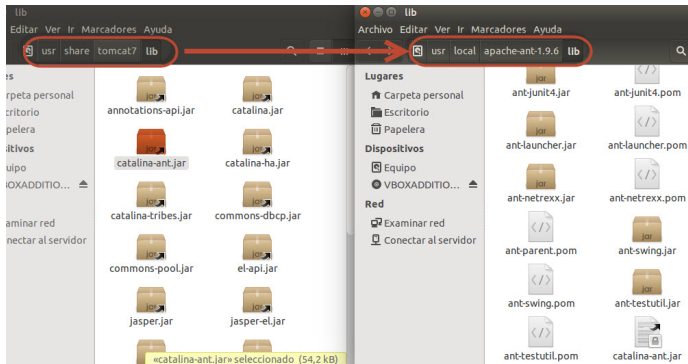
4.3.- Incluir tareas Ant en Tomcat.

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:

- ✓ 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" />
```

Anexo.- Licencias de recursos.

Licencias de recursos utilizados en la Unidad de Trabajo.

Recurso (1)	Datos del recurso (1)	Recurso (2)	Datos del recurso (2)
	Autoría: http://mygnet.net/articulos/tomcat/66/ . Procedencia: http://mygnet.net/articulos/tomcat/66/		Autoría: Luis M. Gallardo D. Licencia: CC-by-nc-nd/2.0. Procedencia: http://www.flickr.com/photos/lgallardo/5
	Autoría: Boltron. Licencia: CC-by-sa/2.0. Procedencia: http://www.flickr.com/photos/boltron/181220090/sizes/t/in/photostream/		Autoría: Br1dotcom. Licencia: CC-by/2.0. Procedencia: http://www.flickr.com/photos/br1dotcom
	Autoría: Marek Goldmann. Licencia: CC-by-nd/2.0. Procedencia: http://www.flickr.com/photos/goldmann/3031615675/sizes/t/in/photostream/		Autoría: Clarkbw. Licencia: CC-by-nc-sa/2.0. Procedencia: http://www.flickr.com/photos/clarkbw/22
	Autoría: Espacio CAMON. Licencia: CC-by-nc-sa/2.0. Procedencia: http://www.flickr.com/photos/tucamon/3861984752/sizes/o/in/photostream/		Autoría: Apache Software Foundation. Licencia: Licencia Apache 2.0 (http://www.apache.org/licenses/LICENSE-2.0). Procedencia: http://ant.apache.org/ima
	Autoría: S.alt. Licencia: CC-by-nc-nd/2.0. Procedencia: http://www.flickr.com/photos/salz/2887001822/sizes/o/in/photostream/		Autoría: Hubert.tw. Licencia: CC-by-nc-sa/2.0. Procedencia: http://www.flickr.com/photos/hub19/309
	Autoría: Wa7son. Licencia: CC-by-nc-nd/2.0. Procedencia: http://www.flickr.com/photos/wa7son/148923726/sizes/l/in/photostream/		Autoría: Marek Goldmann. Licencia: CC-by-nd/2.0. Procedencia: http://www.flickr.com/photos/goldmann/
	Autoría: Ellecer. Licencia: CC-by-nd/2.0. Procedencia: http://www.flickr.com/photos/ellecer/3225440121/sizes/l/in/photostream/		Autoría: Patrick Johanneson. Licencia: CC-by-nc-sa/2.0. Procedencia: http://www.flickr.com/photos/pj/3021543