

Servidor de aplicaciones

Conceptos básicos

El servidor de aplicaciones

Un servidor de aplicaciones es un motor que ofrece **servicios de aplicaciones a clientes** o dispositivos.

Su principal beneficio es la **facilidad de desarrollo de software** ya que incluyen herramientas que no necesitan ser programadas: simplemente hay que **desplegarlas sobre el servidor de aplicaciones**.

Normalmente se almacenarán en contenedores de aplicaciones.



Ejemplos

Ejemplos que podemos encontrar son los siguientes:

- **IIS** permite la ejecución de aplicaciones web con tecnología ASPX (.NET Framework)
 - **Tomcat** integra el servidor web Apache con un contenedor de servlets (JSP, Servlets)
 - **Wildfly** – antiguo JBOSS (Java Enterprise Edition)
 - **Oracle WebLogic** - (Java Enterprise Edition)
 - **Websphere Application Server (IBM)** - (Java Enterprise Edition)
 - **GlassFish** - (Java Enterprise Edition)
 - **Zend Server** - (Apache + PHP)
- 

Diferencias con servidor web

Las características y los servicios que ofrece el servidor de aplicaciones son los siguientes:

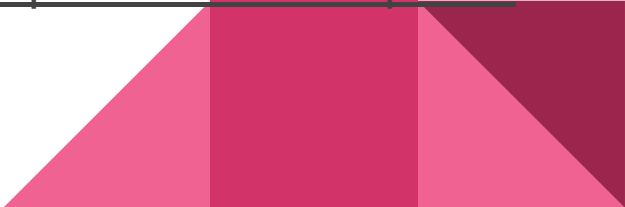
- Sistemas de autenticación (seguridad)
- Implementación de lógica de negocio (p.e. conexión con motores de BBDD)
- Gestión de sesiones de usuario
- Acceso a los componentes o librerías de la plataforma utilizada (Java o .NET)



Tomcat vs GlassFish (u otros)

Tomcat solo soporta los contenedores web de la plataforma JEE mientras que, por ejemplo, GlassFish soporta todos los tipos por lo es realmente un servidor de aplicaciones en un sentido más completo.

Vamos a ver Tomcat por su **simplicidad, facilidad de uso** y configuración, **bajo consumo de recursos y alto rendimiento** y por qué es muy utilizado, pero en este caso no existe un paralelismo con lo que veíamos con Apache y el mercado está mucho más fragmento. Para pequeñas aplicaciones se sigue recomendando Tomcat, pero si necesitamos JEE al completo tendremos que buscar otra opción.



Java vs PHP, Ruby, Python...

Con el tiempo Java ha ido perdiendo pujanza como lenguaje de programación para la web. Últimamente se ha ido quedando para aplicaciones grandes por características como el **balanceo de carga**, **fácil escalabilidad** o las **posibilidades de comunicación** entre sistemas de Backend.

Sin embargo, se considera **muy pesada para las aplicaciones más pequeñas** y está siendo rápidamente sustituido por lenguajes como PHP, Ruby o Python.



Arquitectura TOMCAT

El motor de servlets de Tomcat a menudo se presenta en combinación con el servidor HTTP Apache u otros servidores web.

Tomcat puede funcionar como servidor web de manera autónoma.

Utiliza una estructura en la que varios componentes se organizan de manera jerárquica. Estos componentes son *Server*, *Service*, *Engine*, *Host*, *Connector* y *Context*.



Arquitectura TOMCAT

Catalina

Es el contenedor de servlets. Con Catalina se pueden implementar Realms para proporcionar sistemas de autenticación y acceder a recursos por medio de usuarios, contraseñas y roles.

Coyote

Es un conector que admite HTTP1.1, que permite a Tomcat actuar como un servidor web.



Arquitectura TOMCAT

Jasper

Compila ficheros JSP de manera que se puedan conlleva como a servlets dirigidos por Catalina. Necesita el paquete de librerías JDK.

Clusterización

Tomcat permite la utilización de clustering, por lo que puede dirigir una gran cantidad de peticiones de manera eficiente.



Arquitectura TOMCAT

High availability

Permite actualizar el servidor fácilmente sin afectar el uso de las aplicaciones desplegadas.

Load balancing

Permite distribuir la carga de trabajo entre varios ordenadores, redes, CPU, etc.



Estructura de directorios de Tomcat

Los directorios que se incluyen en una instalación de Tomcat se ubican generalmente en el directorio donde descomprimos Tomcat.

Sin embargo, en una instalación UNIX están definidos en dos ubicaciones: `/usr/share/tomcat[VERSION]` y `/var/lib/tomcat[VERSION]`. Sin embargo, hay otro directorio afectado en la instalación `/etc/tomcat[VERSION]/`

[VERSION] corresponde al número de versión de Tomcat (actualmente 10)



Estructura de directorios de Tomcat

/bin: Contiene los binarios y scripts de inicio de Tomcat.

/conf: La configuración global de Tomcat.

/lib: Contiene 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: Los archivos de registro.

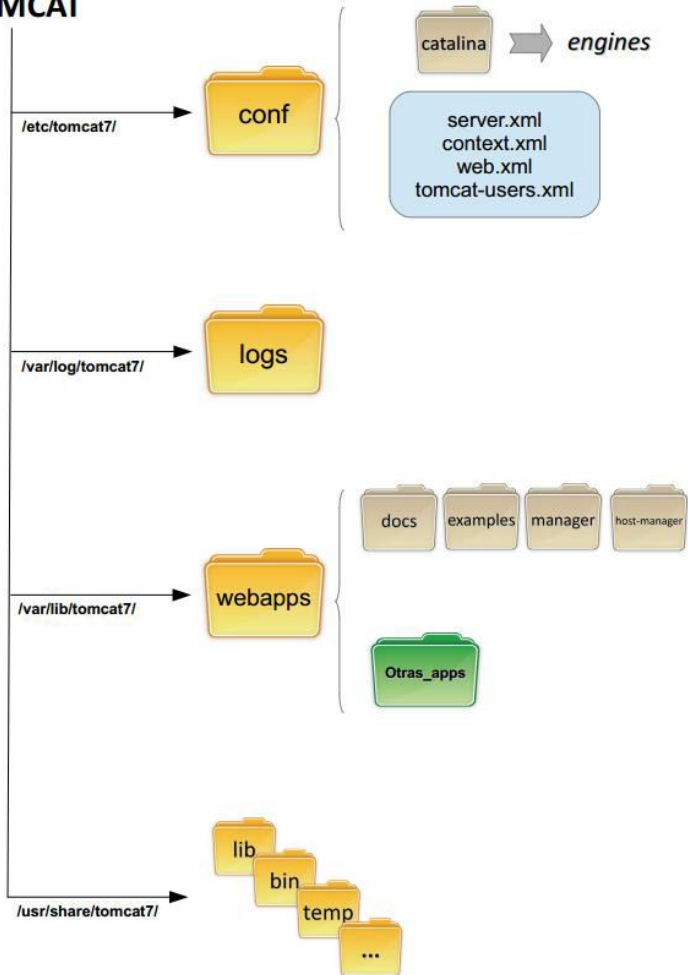
/temp: Este directorio es opcional. Se usa para los archivos temporales que necesita Tomcat durante su ejecución.

/webapps: Aquí se ubican las aplicaciones propiamente dichas. Por defecto solo hay una que se denomina ROOT. Si se ha instalado la documentación, ejemplos y administradores, estos se encuentran en su propio directorio en /usr/share/tomcat10-admin, tomcat10-docs y tomcat10-examples.

/work: Archivos temporales, páginas JSP precompiladas y otros archivos intermedios.



TOMCAT



Contenido de la carpeta /conf

- ✓ **catalina.policy**: Este archivo contiene la política de seguridad relacionada con Java e impide que los Servlets o JSPs la sobrescriben por motivos de seguridad.
- ✓ **catalina.properties**: Contiene los archivos .JAR que no pueden sobrescribirse por motivos de seguridad y otros de uso común.
- ✓ **logging.properties**: Establece las políticas generales para el registro de actividad del servidor, aplicaciones o paquetes.



Contenido de la carpeta /conf

- ✓ **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 antes de usarlos.



Contenido de la carpeta /conf

✓ **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.

Los descriptors de **cada aplicación sobrescribirá la información escrita en este descriptor con sus propios web.xml**.

No confundir con este web.xml específico de aplicación.



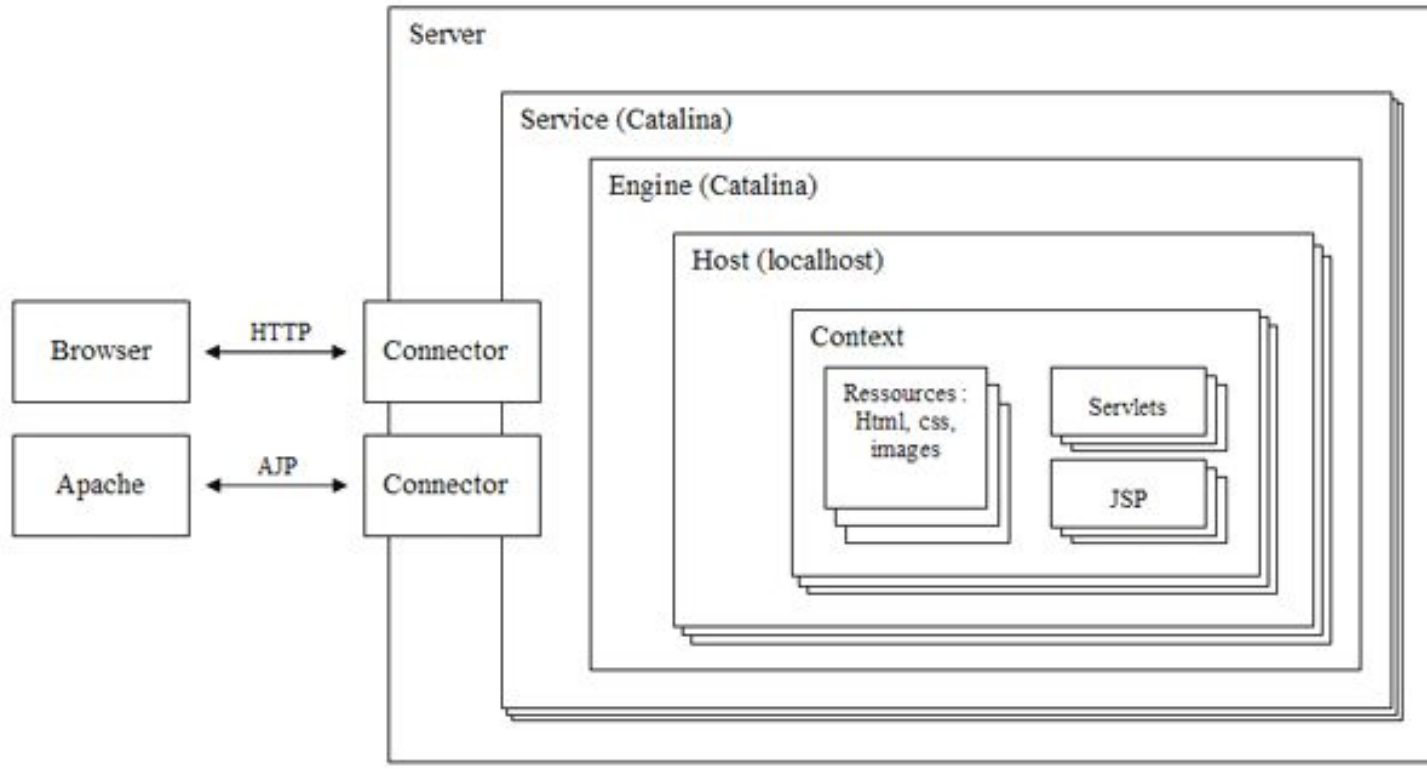
Contenido de la carpeta /conf

✓ **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.

Además de este context.xml, puede existir un fichero de contexto específico para cada aplicación. Este archivo tiene la forma **nombre-de-aplicación.xml**.




Arquitectura de Tomcat (Conceptos)



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, esto no afecte a otras aplicaciones.

Contiene varios **Listeners** que escuchan y responden a eventos. También usa **GlobalNamingResources** que sirven para permitir a clientes software hechos en Java encontrar objetos y datos mediante su nombre. Por ejemplo, aquí se podría indicar un recurso global como una base de datos MySQL.



Service

Agrupar un **contenedor de tipo Engine con un conjunto de conectores**. El motor suele ser Catalina y los conectores por defecto HTTP y AJP (Apache JServ Protocol).

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 (pipeline para Catalina) pueden ubicarse dentro de un contenedor.



Engine (Motor) - Catalina

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 qué **host** (virtualhost) o **context** se le debe pasar cada petición.

Cuando Tomcat se utiliza como servidor autónomo se usa Coyote como servidor web. Cuando Tomcat se usa dando soporte a un servidor web (como Apache) se sobrescribe Coyote porque el servidor web ya ha determinado el destino correcto para las peticiones.

Un Engine puede contener **Host** que representa 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 (similar a los virtualhosts de apache).

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 - Gestión de múltiples context

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 (**trabaja en paralelo**).

Su funcionamiento queda 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.

- Si lo aplicamos a nivel de **motor**, los usuarios o grupos tendrán idénticos permisos en todas las aplicaciones a la hora de acceder a objetos o recursos.
- Si lo aplicamos en un **Host**, los permisos serán para todas las aplicaciones de dicho entorno.
- Si lo aplicamos en un **Context** se establecerán por cada aplicación. Sin embargo, aunque un usuario tenga los mismos permisos para todas las aplicaciones debe seguir autenticandose en cada una de ellas por separado.

Los permisos se heredan y por eso se da la situación descrita en el párrafo anterior, pero si los permisos de un usuario o grupo se sobrescriben en un entorno de rango inferior los últimos serán los que se apliquen.



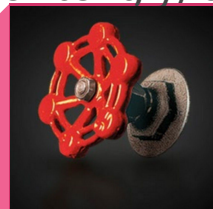
Valve

Se usa para **interceptar peticiones antes de pasarlas a las aplicaciones**.

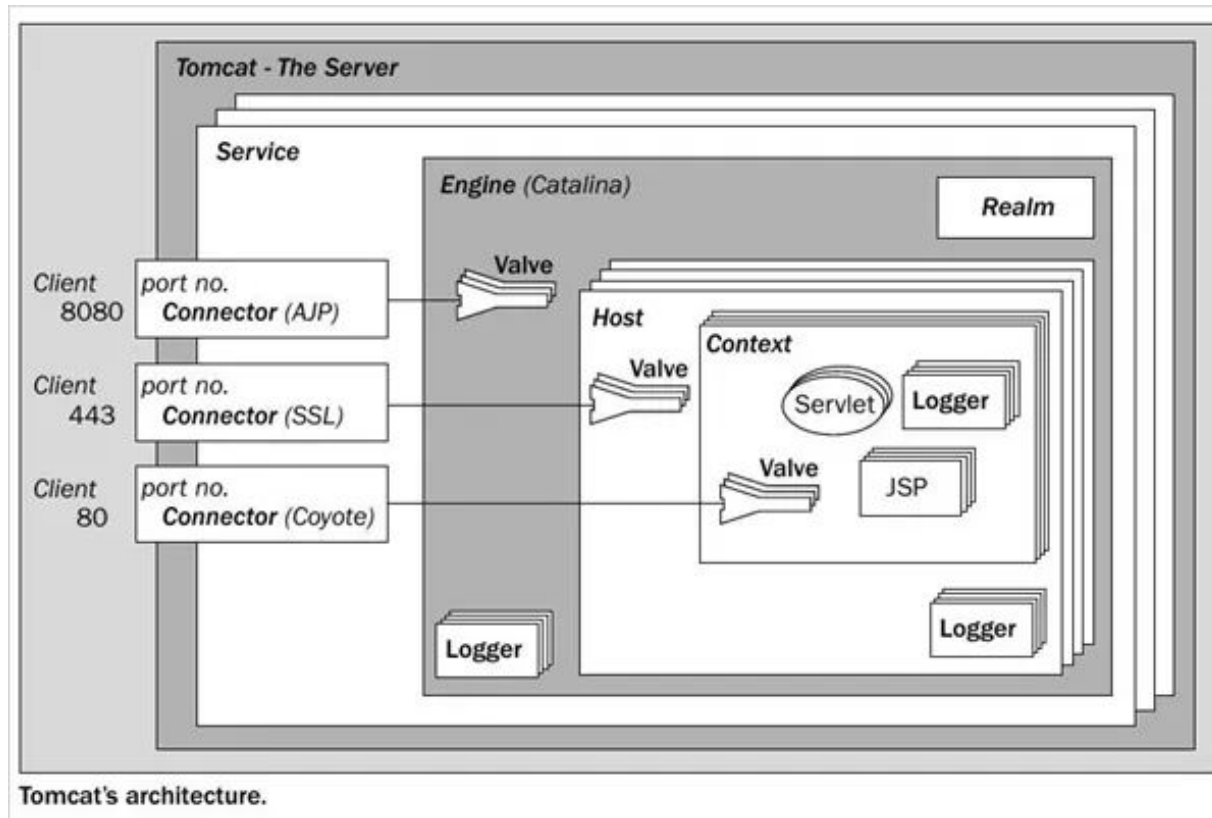
Esto nos permite **pre-procesar** las peticiones para bloquear algunas, registrar accesos, registrar detalles de la conexión (en archivos log), o establecer un único punto de acceso para todas las aplicaciones de un host o para todos los hosts de un servidor.

Afectan al tiempo de respuesta a la petición.

Puede establecerse para cualquier contenedor Engine, Host, and Context, y/o Cluster.



Arquitectura de Tomcat (avanzado)



Arquitectura de Tomcat (simplificado - service)

