# RESUMEN TEMA 3.

# CONFIGURACIÓN Y ADMINISTRACIÓN DE SERVIDORES DE APLICACIONES.

## 1.- Protección del servidor de aplicaciones.

Un servidor de aplicaciones es, usualmente, un software que proporciona una serie de servicios de aplicación aun número indeterminado de computadoras cliente que acceden a dichos servicios vía web.

Ventaja principal la centralización y disminución de la complejidad en el desarrollo.

Aplicaciones más expuestas a ataques y se pueden clasificar en tres niveles:

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

En cada uno de los niveles es necesario garantizar una seguridad mínima para conseguir la seguridad de todo el proceso.

* Usuario: contar con navegadores y plataformas seguras, libres de virus.
* Servidor: tiene que garantizar que los datos son integros y que solo sea distribuida a las personas autorizadas.
* Tránsito de la información: ésta no debe ser leída (confidencialidad), modificada o destruida por terceros, y un canal fiable e ininterrumpido.

Mecanismos de seguridad:

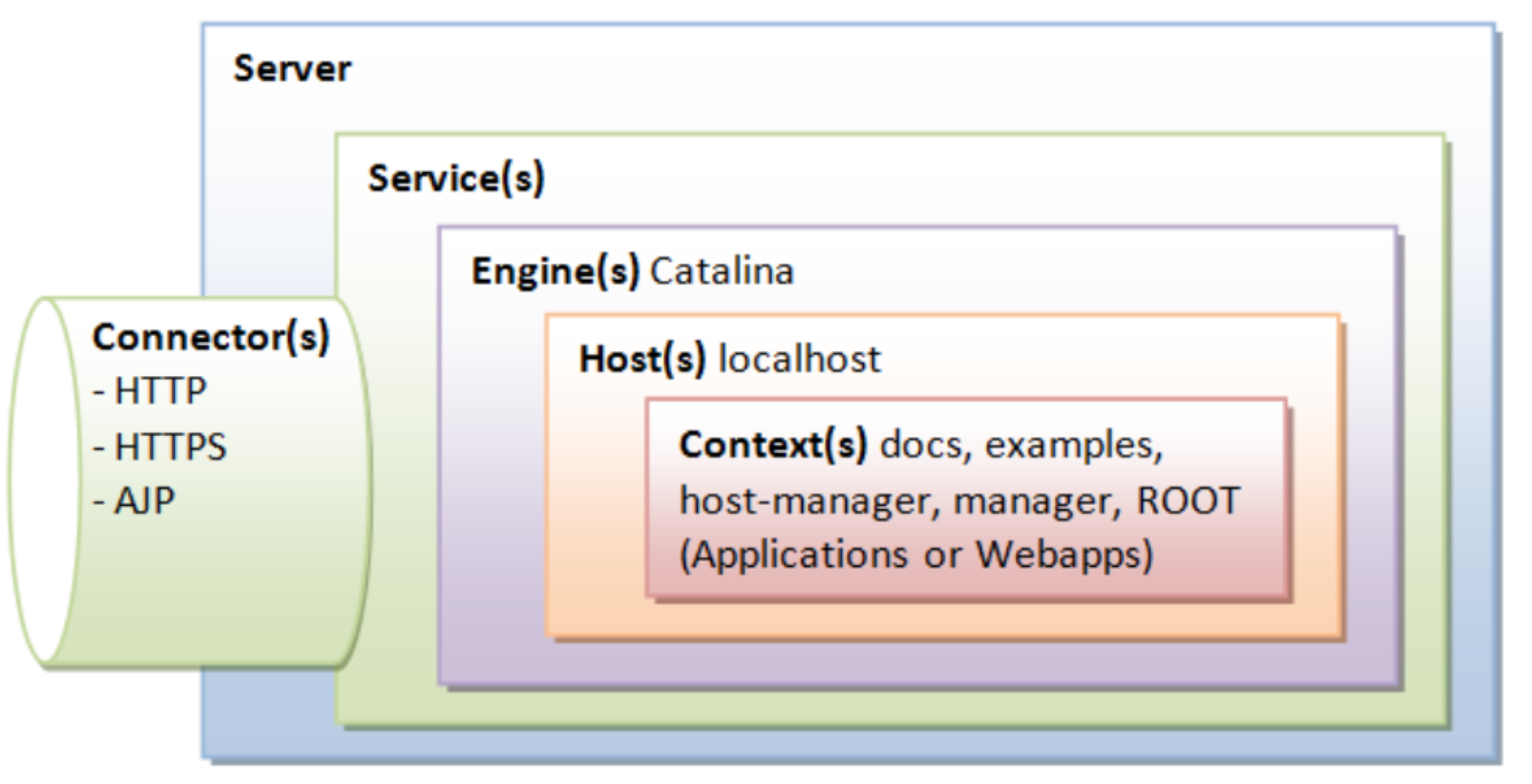
* **Autenticación**: permite identificar, en todo momento, quién es el usuario que esta accediendo. Métodos:
  + Autenticación básica: solicitud de usuario y clave.
  + Autenticación con certificados.
    - HTTP DIGEST AUTH (HTTP Autenticación de texto implícita)
    - HTTP NTLM AUTH (HTTP Autentication Microsoft NT Lan Manager)
* **Autorización**: permite, una vez autenticado, determinar a qué datos y módulos de la aplicación puede acceder el usuario.
* **Validación de entradas**, ya que se puede manipular el código de validación del lado del cliente.
* **Inyección de comandos SQL:** técnica para explotar aplicaciones web que no validad la información suministrada por el cliente para generar consultas SQL peligrosas.

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

* Deshabilitación de servicios y cuentas no utilizadas.
* Actualización de sistema operativo y aplicaciones (parches).
* Fortaleza en las contraseñas.
* Utilización de Firewalls.
* Back-ups periódicas.
* Análisis periódico de logs.
* Verificación periódica de servicios activos.
* Cifrado del tráfico.
* Establecimiento de políticas de seguridad.

## 2.- Servidor de aplicaciones web Tomcat.

Conocer la arquitectura de tomcat para entender mejor las posibles configuraciones que se pueden aplicar:



* El servidor es el elemento raíz y contiene uno más servicios, cada uno de ellos con sus propios conectores y motores.
* Los escuchadores están atentos a la aparición de eventos que necesiten ser procesados.
* Los servicios relacionan conectores con motores, un conector se asocia aun puerto TCP para gestionar la comunicación entre el servidor y los clientes.
* Un motor es la instancia que representa el contenedor, puede contener uno o más servidores virtuales (hosts).
* El motor direcciona la petición al servidor virtual dependiendo de su nombre o IP en la cabecera de petición del mensaje HTTP.
* Un realm es una base de datos de usuarios utilizada en el proceso de autenticación.

## 2.1- Creación de una aplicación web.

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

Variables de entorno (/**etc/profile**):

* **JAVA\_HOME**: indica los archivos binarios de Java.
* **CATALINA\_HOME:** indica la ubicación de los scrips de Tomcat.

CATALINA\_HOME=/opt/tomcat

JAVA\_HOME=/usr/lib/jvm/java-1.11.0-openjdk-amd64

PATH=$PATH:$JAVA\_HOME/bin:$CATALINA\_HOME

export PATH JAVA\_HOME CATALINA\_HOME

Actualizamos:

source /etc/profile

Creación:

* Apli\_Web > index.jsp , WEB-INF > classes, lib
* Index.jsp 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 APLIC  </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> |

Generar el archivo .war.

$ javac -d WEB-INF/classes \*.java

$ jar cvf Aplic\_Web.war \*

Javac compila las clases Java de la aplicación y jar crea el archivo .war.

Vía web [**http://locahost**](http://locahost)**:8080** > Manager , tenemos la opción para desplegar el .war

O también podemos hacer una copia a la carpeta **$CATALINA\_HOME/webapps.**

**Luego,** accedemos a <http://ip>-server:8080/Aplic\_Web y tenemos la aplicación desplegada.

## 2.2- Descriptor de despliegue de una aplicación web.

El objetivo de las aplicaciones web es que puedan ser desplegadas en diferentes servidores web, manteniendo su funcionalidad y sin ninguna modificación de código.

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

Antes de la especificación Servlet 2.2 ( llamados motores servlet) no estaba estandarizado el despliegue entre contenedores, después de esta versión la portabilidad de código Java dio un paso más allá.

El método más sencillo para desplegar una aplicación, es copiar a la carpeta **$CATALINA\_HOME/webapps.**

**Web.xml** es un fichero descriptor del despliegue, 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"  xsi=  "http://www.w3.org/2001/XMLSchema-instance"  schemaLocation=  "http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web  version=  "2.5">  <display-name>Descriptor Aplicacion Aplic\_Web</display-name>  <description>  Mi primer descriptor web.xml.  </description>  </web-app> |

Y crearlo dentro de la carpeta WEB-INF.

Parámetros:

* **<description>:** Descripción de la aplicación.
* **<display-name>:** Nombre corto de la aplicación.
* **<servlet>:** Define los servlets individuales que componen tu aplicación. Incluye sub-elementos como <servlet-name>, <servlet-class> o <jsp-file>, y opcionalmente <init-param> para parámetros de inicialización.
* **<servlet-mapping>:** Asocia los servlets con uno o más patrones de URL. Utiliza <servlet-name> para referenciar un servlet definido y <url-pattern> para especificar el patrón de URL.
* **<listener>.** Define los listeners de eventos del ciclo de vida de tu aplicación, como los ServletContextListeners. Utiliza <listener-class> para especificar la clase del listener.
* **<filter> y <filter-mapping>:** Define los filtros que pueden modificar las solicitudes y respuestas en tu aplicación. <filter> define el filtro y <filter-mapping> asocia el filtro con un servlet o un patrón de URL.
* **<welcome-file-list>:** Enumera los archivos que Tomcat mostrará como páginas de inicio por defecto cuando no se especifica un archivo en la URL. Utiliza **<welcome-file>** para cada nombre de archivo que desees incluir en la lista.
* **<context-param>:** Define parámetros de contexto que son globales para toda la aplicación. Incluye un <param-name> y un <param-value>.
* **<error-page>:** Especifica las páginas que deben mostrarse como respuesta a determinados errores o excepciones. Incluye <error-code> o <exception-type> y <location> para especificar la página de error.
* **<session-config>:** Configura los parámetros de la sesión, como el tiempo de timeout.
* **<session-timeout>** define el tiempo de inactividad en minutos antes de que la sesión expire.
* **<mime-mapping>:** Define las asociaciones entre extensiones de archivo y tipos MIME. Usa **<extension>** para la extensión del archivo y <mime-type> para el tipo MIME correspondiente.
* **<security-constraint>:** Define las restricciones de seguridad para áreas específicas de la aplicación. Incluye sub-elementos como <web-resource-collection>, <auth-constraint>, y <user-data-constraint>.
* **<login-config>:** Configura el método de autenticación para la aplicación. Puede especificar, por ejemplo, la autenticación BASIC, FORM, DIGEST o CLIENT-CERT.
* **<jsp-config>:** Proporciona configuración global para los JSPs, como el tamaño del buffer o la página de error predeterminada.
* **<env-entry>:** Se utiliza para definir variables de entorno para una aplicación web. Usa <env-entry-name>, <env-entry-value> y <env-entry-type> para definir el nombre, valor y tipo de la variable de entorno.

Ficheros de ejemplo en : $CATALINA\_HOME/webapps/examples/WEB-INF

Generar el archivo .WAR.

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

**# jar cvf Aplic\_Web.war \*** para crear el archivo .WAR.

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

# 2.3.- Despliegue de una aplicación web.

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 servlet2.2. Estructura de directorios:

Directorio principal (**raíz**): Contendrá los ficheros estáticos (HTML, imágenes, etc...) y JSPs.

* Carpeta **WEB-INF**: contiene el fichero **"web.xml"** (descriptor de la aplicación), encargado de configurar la aplicación.
  + Subcarpeta **classes**: contiene los ficheros compilados (servlets, beans).
  + Subcarpeta **lib**: librerías adicionales.
* Resto de carpetas para ficheros estáticos.

Una aplicación web suele ser desplegada en tomcat empleando los siguiente métodos:

* Copiando el archivo .war en el directorio de despliegue $CATALINA\_HOME/webapps.
* A través de la aplicación de administración web del servidor Tomcat.

## 2.4.- Implementar el registro de acceso.

Las válvulas de 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.

La válvulas disponibles son:

* **Access Log Valve:** está implementada por la claseorg.apache.catalina.valves.AccessLogValve. Crea ficheros de log para rastrear el acceso a la información de los clientes, registrando información como, por ejemplo, actividad de la sesión del usuario, información de la autenticación del usuario, entre otras.
  + Indicará que los logs de acceso se almacenarán en el directorio $CATALINA\_HOME/logs ylos archivos de log tendrán la nomenclatura con prefijo: localhost\_access\_log y sufijo.txt probablemente entre sufijo y prefijo se añadirá la fecha en la que se crea dicho
    archivo.
* **Remote Address Filter:** permite comparar la dirección IP del cliente con una o más expresiones regulares y, como resultado de ello, denegar o bien permitir la solicitud presentada por el cliente.
* **Remote Host Filter:** es muy parecido al anterior pero con la diferencia que permitecomparar por nombre de equipo en lugar de IP.
* **Request Dumper:** es una herramienta de depuración que escribe en el log el detalle de cada petición realizada.
* **Single Sign On:** cuando queremos que los usuarios puedan identificarse en cualquier aplicación de nuestro virtual host, y que su identidad sea reconocida por cualquier aplicación que esté en ese host.

## 2.5.- Sesiones persistentes.