



Práctica Tema 4
Servidor de Aplicaciones



Vamos a trabajar con una imagen de **Tomcat** en Docker para el uso de su motor Catalina. Como en la práctica anterior hay varios puntos en y varios niveles. De nuevo vuestro trabajo será seguir los pasos de cada punto para **sacar las capturas mínimas para demostrar que el punto está completado**, y decir los **problemas que habéis encontrado y cómo los habéis solucionado**. Si añadís recomendaciones como buenas prácticas encontradas, relacionadas con los puntos, también se tendrá en cuenta.

Esta será tu estructura de carpetas aproximadamente:

```
C:
|- pom.xml
|- src/
|  |- main/
|     |- docker-compose.yml
|     |- server.xml
|     |- tomcat-users.xml
|     |- certs/
|        |- tomcat.crt
|        |- tomcat.key
|     |- java/
|        |- calculadora
|           |- CalculadoraServlet.java
|     |- webapp/
|        |- index.jsp
|        |- login-error.jsp
|        |- login.jsp
|        |- logout.jsp
|        |- css/
|           |- estilo.css
|        |- WEB-INF/
|           |- web.xml
```



Paso 1: Crea tu archivo docker-compose y comprueba que tu contenedor se levanta con normalidad.

docker-compose: Este archivo creará el contenedor docker con la imagen de Tomcat.

```
version: "3.8"

services:
  tomcat:
    image: tomcat:10.1
    container_name: tomcat_t3
    ports:
      - "8080:8080"
      - "8443:8443"
    restart: unless-stopped
```

Paso 2: Generar los certificados autofirmados SSL/TLS y modificar docker-compose en consecuencia. Comprueba que puedes entrar por “https”.

RECUERDA que estamos trabajando con contenedores, luego una vez generes los certificados asegúrate de copiarlos en el contenedor a través de tu archivo docker-compose. Tomcat buscará los certificados en:

```
"/usr/local/tomcat/certs"
```

Paso 3: Instala Maven y comprueba que lo has hecho correctamente.

Para esta práctica es necesario instalar maven:

```
sudo apt install maven
```

Revisa que lo has hecho correctamente:

```
mvn --version
```



Paso 4: Crea el archivo pom.xml y asegúrate de que concuerde con la versión de java que está usando maven.

Archivo pom.xml: Este archivo permite indicar la configuración para la construcción de nuestro proyecto (archivo.war)

```
<!--  
pom.xml = configuración de Maven.  
Maven se encarga de:  
- Descargar dependencias  
- Compilar el código Java  
- Empaquetar la aplicación como WAR para desplegar en Tomcat  
-->  
<project xmlns="http://maven.apache.org/POM/4.0.0">  
  <modelVersion>4.0.0</modelVersion>  
  
  <groupId>calculadora</groupId> <!-- Identidad del proyecto -->  
  <artifactId>calculadora</artifactId> <!-- artifactId: nombre del proyecto/artefacto -->  
  <version>1.0.0</version>  
  <packaging>war</packaging> <!-- packaging: indica que el resultado final será un WAR -->  
  
  <properties>  
    <!--  
    Versión de Java con la que compilamos.  
    Recomendación docente: usar 17 para evitar diferencias entre equipos.  
    -->  
    <maven.compiler.release>17</maven.compiler.release>  
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  
  </properties>  
  
  <dependencies>  
    <!--  
    API de Servlets Jakarta (para Tomcat 10).  
    scope=provided significa:  
    - Se necesita para COMPILAR  
    - NO se empaqueta dentro del WAR  
    - Porque Tomcat ya incluye esta librería en el servidor.  
    -->  
    <dependency>  
      <groupId>jakarta.servlet</groupId>  
      <artifactId>jakarta.servlet-api</artifactId>  
      <version>6.0.0</version>
```



```
<scope>provided</scope>
</dependency>
</dependencies>

<build>
  <!--
    finalName controla el nombre del fichero generado, generará:
    target/calculadora.war
    (Esto es útil para Docker)
  -->
  <finalName>calculadora</finalName>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-war-plugin</artifactId>
      <version>3.4.0</version>
      <configuration>
        <!--Si falta web.xml, NO falla el build.-->
        <failOnMissingWebXml>false</failOnMissingWebXml>
      </configuration>
    </plugin>
  </plugins>
</build>
</project>
```

Paso 5: Crea tu archivo server.xml, copiando el generado por Tomcat en tu proyecto.

La imagen de Tomcat que has usado para crear el contenedor tiene una versión del archivo server.xml. Ve a la carpeta de tu proyecto y copia el archivo del contenedor a tu proyecto en local. Recuerda, tomcat_t3 es el nombre que le hemos dado al contenedor:

```
docker cp tomcat_t3:/usr/local/tomcat/conf/server.xml ./server.xml
```

Tomcat buscará el archivo server.xml en:

```
"/usr/local/tomcat/conf/server.xml"
```



Paso 6: Modifica el server.xml para indicar donde encontrará el contenedor los certificados.

Una vez obtenido el archivo server.xml del archivo anterior, debemos buscar y descomentar el bloque que indica donde encontrar los certificados. Recuerda que debemos indicar la ruta del contenedor, no de local, y el nombre del archivo de los certificados. Dentro de ese bloque debes tener algo así:

```
<SSLHostConfig>
  <Certificate certificateFile="Ruta del archivo.crt"
               certificateKeyFile="Ruta del archivo.key"
               type="RSA" />
</SSLHostConfig>
```

Paso 7: Crea tu aplicación calculadora en dos pasos, un archivo java que contendrá la lógica y un index.jsp que será tu interfaz.

Enlace al archivo.java → [ejemploJose.java](#)

Enlace al archivo.jsp → [indexEjemploJose.jsp](#)

Estos archivos son de guía, recuerda que debes hacer cada uno de estos a tu manera, la creatividad tendrá un peso importante en esta práctica.

Paso 8: Añade tu archivo web.xml e intenta probar a generar el archivo.war con maven.

Enlace al archivo web.xml → [ejemploJoseWeb.xml](#)

Para compilar el proyecto debemos estar en el directorio donde se encuentre nuestro pom.xml y ejecutar:

```
mvn clean package
```



Si la compilación de maven ha ido como debe, se ha tenido que generar un directorio target donde se habrán generado muchas carpetas y también el **ARCHIVO.war**.

Tomcat buscará el archivo.war en:

```
"/usr/local/tomcat/webapps/archivo.war"
```

Recuerda que estamos ejecutando el servicio en nuestro contenedor, piensa en todo momento que archivos necesita el contenedor para poder desplegar el servicio.

Paso 9: Añade roles a tu app con el archivo tomcat-users.xml y comprobar su correcto funcionamiento, tanto de los usuarios con sus contraseñas como con las restricciones impuestas a los roles.

tomcat-users.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<tomcat-users>

  <!-- Roles de la calculadora -->
  <role rolename="calc_basic"/>
  <role rolename="calc_full"/>

  <!-- Usuario con calculadora básica -->
  <user username="basic_user" password="123" roles="calc_basic"/>

  <!-- Usuario con calculadora completa -->
  <user username="full_user" password="321" roles="calc_full"/>

</tomcat-users>
```

Tomcat buscará los usuarios en:

```
"/usr/local/tomcat/conf/tomcat-users.xml"
```

Enlace archivo login.jsp → [ejemploLoginJose.jsp](#)

Enlace archivo login-error.jsp → [ejemploLoginErrorJose.jsp](#)

Enlace archivo logout.jsp → [ejemploLogoutJose.jsp](#)



Entrega:

Es necesario entregar el proyecto con **todos los archivos** en un **.RAR** con el nombre, **Apellido1_Nombre_PracticaT3.rar**, añadir el **enlace al proyecto** o el propio **proyecto**, además de un pequeño documento que incluye los roles definidos, y dos tipos de usuarios con sus contraseñas. Añade en el documento las **capturas del progreso en los puntos** y los **problemas que has tenido** al realizar los puntos, si los has tenido y **cómo los has resuelto**. Además, incluye las **buenas prácticas** que hayas encontrado durante la práctica.

Resumen de lo que debe contener la tu práctica

- **NOMBRE** → **Apellido1_Nombre_PracticaT3** (Si no, no se corrige, quien avisa no es traidor)
- **Carpeta del proyecto** junto el documento con las capturas(**PORTADA**)
- **Problemas encontrados y cómo los has resuelto**
- **Investiga algunas de las buenas prácticas que se llevan a cabo hoy día.**

(SI NO SE CUMPLEN LAS REGLAS DE ENTREGA, SE CONSIDERARÁ QUE LA PRÁCTICA NO SE HA ENTREGADO)