Java Servlet Application with Maven

Here's a complete guide to creating and deploying a simple Java Servlet application on WebSphere Liberty Profile (WLP) using Maven for building the project and the Liberty Maven plugin for deployment. This example creates a basic Servlet that returns "Hello World!"when accessed, builds it into a WAR file with Maven, and deploys it to WLP. We'll assume you've already set up WLP with the `servlet-4.0` feature enabled, as this provides the necessary Servlet support.

Step-by-Step Guide

1. Create the Maven Project Structure

Start by setting up a standard Maven web application structure. You can create this manually or use Maven to generate it.

- **Directory Structure:**

```
SimpleServletApp/
  pom.xml
  src/
      main/
          java/
              com/
                  example/
                      HelloServlet.java
          webapp/
              WEB-INF/
                  web.xml
```

- **Optionally Generate with Maven:** Run this command to create the structure, then adjust as needed:

```
mvn archetype:generate -DgroupId=com.example -DartifactId=simple-servlet-app -DarchetypeArtifactId=maven-
```

  This creates a basic webapp structure, which you'll modify in the next steps.

2. Write the Servlet Code

Create a file named `HelloServlet.java` in `src/main/java/com/example/` with the following content:

```java
package com.example;

import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;

public class HelloServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws IOException {
        resp.setContentType("text/plain");
        resp.getWriter().write("Hello World!");
    }
}
```

- **Explanation:** This Servlet responds to HTTP GET requests with “Hello World!”in plain text. It uses a simple `doGet` method and avoids annotations for compatibility with explicit `web.xml` configuration.

3. Create the `web.xml` Deployment Descriptor

Create a file named `web.xml` in `src/main/webapp/WEB-INF/` with the following content:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0
         version="4.0">
    <servlet>
        <servlet-name>HelloServlet</servlet-name>
        <servlet-class>com.example.HelloServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>HelloServlet</servlet-name>
        <url-pattern>/hello</url-pattern>
    </servlet-mapping>
</web-app>
```

- **Explanation:** The `web.xml` file defines the `HelloServlet` class and maps it to the `/hello` URL pattern. This is necessary since we’re not using `@WebServlet` annotations.

4. Configure the Maven `pom.xml`

Create or update `pom.xml` in the `SimpleServletApp/` directory with the following content:

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.example</groupId>
    <artifactId>simple-servlet-app</artifactId>
    <version>1.0-SNAPSHOT</version>
    <packaging>war</packaging>

    <properties>
        <maven.compiler.source>1.8</maven.compiler.source>
        <maven.compiler.target>1.8</maven.compiler.target>
    </properties>

    <dependencies>
        <!-- Servlet API (provided by WLP) -->
        <dependency>
            <groupId>javax.servlet</groupId>
            <artifactId>javax.servlet-api</artifactId>
            <version>4.0.1</version>
            <scope>provided</scope>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <!-- Maven WAR Plugin to build the WAR file -->
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-war-plugin</artifactId>
                <version>3.3.1</version>
                <configuration>
                    <finalName>myapp</finalName>
                </configuration>
            </plugin>
            <!-- Liberty Maven Plugin for deployment -->
```

```
            <plugin>
                <groupId>io.openliberty.tools</groupId>
                <artifactId>liberty-maven-plugin</artifactId>
                <version>3.3.4</version>
                <configuration>
                    <installDirectory>/opt/ibm/wlp</installDirectory>
                    <serverName>myServer</serverName>
                    <appsDirectory>dropins</appsDirectory>
                    <looseApplication>false</looseApplication>
                    <stripVersion>true</stripVersion>
                </configuration>
            </plugin>
        </plugins>
    </build>
</project>
```

- **Explanation:**
  - **Coordinates:** Defines the project with `groupId`, `artifactId`, and `version`. The `packaging` is set to `war` for a web application.
  - **Properties:** Sets Java 8 as the source and target version.
  - **Dependencies:** Includes the Servlet API with `provided` scope, as it's supplied by WLP at runtime.
  - **Maven WAR Plugin:** Configures the WAR file name to `myapp.war` using `<finalName>`.
  - **Liberty Maven Plugin:** Configures deployment to a Liberty server at `/opt/ibm/wlp`, server name `myServer`, deploying to the `dropins` directory.

5. Build the Project

From the `SimpleServletApp/` directory, build the WAR file using Maven:

```
mvn clean package
```

- **Result:** This compiles the Servlet, packages it with `web.xml` into `target/myapp.war`, and prepares it for deployment.

6. Deploy and Run on WebSphere Liberty

Ensure your Liberty server (`myServer`) is set up with the `servlet-4.0` feature enabled. Check your `server.xml` for:

```
<featureManager>
    <feature>servlet-4.0</feature>
</featureManager>
```

Deploy and run the application using the Liberty Maven plugin:

```
mvn liberty:run
```

- **What Happens:**
  - Starts the Liberty server in the foreground (if not already running).
  - Deploys `myapp.war` to the `dropins` directory automatically.
  - Keeps the server running until stopped.
- **Verify Deployment:** Look for a log message like:

  ```
  [AUDIT   ] CWWKT0016I: Web application available (default_host): http://localhost:9080/myapp/
  ```

  Logs are typically in `/opt/ibm/wlp/usr/servers/myServer/logs/console.log`.

7. Access the Application

Open a browser and navigate to:

```
http://localhost:9080/myapp/hello
```

- **Expected Output:**

  ```
  Hello World!
  ```

- **URL Breakdown:**
  - `9080`: Default HTTP port for WLP.
  - `/myapp`: Context root from the WAR file name (`myapp.war`).
  - `/hello`: URL pattern from `web.xml`.

8. Stop the Server

Since `mvn liberty:run` runs the server in the foreground, stop it by pressing `Ctrl+C` in the terminal.

Notes

- **Prerequisites:**
  - Maven must be installed and configured on your system.
  - Liberty must be installed at `/opt/ibm/wlp`, and the server instance `myServer` must exist. Adjust `installDirectory` and `serverName` in `pom.xml` if your setup differs (e.g., `/usr/local/wlp` or `defaultServer`).
  - The `servlet-4.0` feature must be enabled in `server.xml`.
- **Alternative Deployment:**
  - To build and deploy separately:

    ```
    mvn clean package
    mvn liberty:deploy
    ```

    Start the server manually if needed:

    ```
    /opt/ibm/wlp/bin/server start myServer
    ```
- **Port Configuration:** If your Liberty server uses a different HTTP port, check `server.xml` for `<httpEndpoint>` and adjust the URL accordingly.
- **Context Root:** Rename `<finalName>` in `pom.xml` (e.g., `<finalName>app</finalName>`) to change the context root to `/app`.
- **Troubleshooting:**
  - If deployment fails, verify the `installDirectory` and `serverName` in `pom.xml`.
  - Check logs at `/opt/ibm/wlp/usr/servers/myServer/logs/messages.log` for errors.
  - Ensure Java 8 or later is used, matching the `maven.compiler` settings.

Summary

This guide demonstrates how to: 1. Set up a Maven project with a simple Servlet (`HelloServlet.java`) and `web.xml`. 2. Configure `pom.xml` with the Servlet API, Maven WAR plugin, and Liberty Maven plugin. 3. Build the application into `myapp.war` using `mvn clean package`. 4. Deploy and run it on WLP using `mvn liberty:run`. 5. Access “Hello World!”at `http://localhost:9080/myapp/hello`.

This provides a streamlined, Maven-based approach to developing and deploying a Servlet application on WebSphere Liberty Profile.
