Deploying Spring Boot to WebSphere Liberty Profile

Here's a complete guide to running a simple Spring Boot application on WebSphere Liberty Profile (WLP). This example creates a Spring Boot application with a basic REST endpoint that returns "Hello World!"and deploys it to WLP using the `dropins` directory. We'll assume you've already set up WLP with the `javaee-8.0` feature enabled, as this provides the necessary Servlet support.

Step-by-Step Guide

1. Create a Spring Boot Application Start by creating a new Spring Boot project using Spring Initializr (or your preferred method). We'll use Maven, Java, and the Spring Web dependency to build a simple web application.

- **Generate the Project:**
  - Go to Spring Initializr.
  - Configure the following:
    - **Project:** Maven
    - **Language:** Java
    - **Spring Boot Version:** 2.7.x (or the latest stable version)
    - **Group:** `com.example`
    - **Artifact:** `demo`
    - **Dependencies:** Spring Web
  - Click "Generate"to download the project ZIP, then unzip it and open it in your IDE.
- **Add a Simple REST Controller:** Inside `src/main/java/com/example/demo`, create a file named `HelloController.java` with this content:

```java
package com.example.demo;

import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class HelloController {
    @GetMapping("/")
    public String hello() {
        return "Hello World!";
    }
}
```

This creates a REST endpoint at the root path (`/`) that returns “Hello World!”as plain text.

2. Configure the Application for WAR Deployment By default, Spring Boot packages applications as JAR files with an embedded server (e.g., Tomcat). To deploy on WLP, we need to package it as a WAR file and configure it to work with WLP’s Servlet container.

- **Modify the Main Application Class:** Edit `src/main/java/com/example/demo/DemoApplication.java` to extend `SpringBootServletInitializer`, which allows the app to run in an external Servlet container like WLP:

```
package com.example.demo;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.boot.builder.SpringApplicationBuilder;
import org.springframework.boot.web.servlet.support.SpringBootServletInitializer;

@SpringBootApplication
public class DemoApplication extends SpringBootServletInitializer {

    @Override
    protected SpringApplicationBuilder configure(SpringApplicationBuilder application) {
        return application.sources(DemoApplication.class);
    }

    public static void main(String[] args) {
        SpringApplication.run(DemoApplication.class, args);
    }
}
```

- **Update `pom.xml` for WAR Packaging:** Open `pom.xml` and make these changes:
  - Set the packaging to WAR by adding this line near the top (below `<modelVersion>`):

    ```
    <packaging>war</packaging>
    ```

  - Mark the embedded Tomcat dependency as `provided` so it’s not included in the WAR (WLP provides its own Servlet container). Modify the `spring-boot-starter-web` dependency (which includes Tomcat) like this:

    ```
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    ```

Add this below it:

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-tomcat</artifactId>
    <scope>provided</scope>
</dependency>
```

Your `pom.xml` dependencies section should now look something like this:

```
<dependencies>
  <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web</artifactId>
  </dependency>
  <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-tomcat</artifactId>
      <scope>provided</scope>
  </dependency>
  <!-- Other dependencies like spring-boot-starter-test may remain -->
</dependencies>
```

3. Build the WAR File Compile and package the application into a WAR file using Maven.

- **Run the Build Command:** From the project root directory (where `pom.xml` is located), execute:

  ```
  mvn clean package
  ```

  This generates the WAR file in the `target` directory, e.g., `target/demo-0.0.1-SNAPSHOT.war`.

- **Rename the WAR File (Optional):** For a cleaner URL, rename the WAR file to `myapp.war`:

  ```
  mv target/demo-0.0.1-SNAPSHOT.war target/myapp.war
  ```

  This simplifies the context root to `/myapp` instead of `/demo-0.0.1-SNAPSHOT`.

4. Deploy the WAR File on WLP Deploy the WAR file to WLP using the `dropins` directory, which enables automatic deployment.

- **Locate the `dropins` Directory:** Find your WLP server's `dropins` directory. If WLP is installed at `/opt/ibm/wlp` and your server is named `myServer`, the path is:

  ```
  /opt/ibm/wlp/usr/servers/myServer/dropins
  ```

- **Copy the WAR File:** Move the WAR file to the `dropins` directory:

  ```
  cp target/myapp.war /opt/ibm/wlp/usr/servers/myServer/dropins/
  ```

- **Start the Server (if Not Running):** If WLP isn't running, start it:

  ```
  /opt/ibm/wlp/bin/server start myServer
  ```

  If it's already running, it will detect and deploy the WAR file automatically.

- **Verify Deployment:** Check the server logs or console for a message like:

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

  - Logs are in `/opt/ibm/wlp/usr/servers/myServer/logs/console.log` (background mode) or displayed in the terminal (foreground mode with `./server run myServer`).

5. **Access the Application** Test the deployed Spring Boot application in a browser.

   - **Open Your Browser:** Navigate to:

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

     - `9080` is WLP's default HTTP port.
     - `/myapp` is the context root from the WAR file name.
     - `/` matches the `@GetMapping("/")` in the controller.

   - **Expected Result:** You should see:

     ```
     Hello World!
     ```

     displayed as plain text.

Notes

- **Context Root:** The context root (`/myapp`) is derived from the WAR file name. Adjust it by renaming the WAR file as needed.
- **Port Number:** WLP uses `9080` by default for HTTP. If your server uses a different port, update the URL accordingly.
- **Java Version:** Ensure WLP and your build environment use a compatible Java version (e.g., Java 8 or 11 for Spring Boot 2.7.x).
- **WLP Features:** The `javaee-8.0` feature includes Servlet 4.0 support, which is compatible with Spring Boot 2.7.x's requirements (Servlet 3.1+).

- **Troubleshooting:**
  - If the page doesn't load, check `/opt/ibm/wlp/usr/servers/myServer/logs/messages.log` for errors.
  - Verify the WAR file is in `dropins` and the server is running.

Summary

This guide demonstrates how to: 1. Create a Spring Boot application with a REST endpoint. 2. Configure it for WAR deployment by extending `SpringBootServletInitializer` and setting Tomcat to `provided`. 3. Build and deploy the WAR file to WLP's `dropins` directory. 4. Access the "Hello World!"output at `http://localhost:9080/myapp/`.

This approach leverages WLP's automatic deployment capabilities and Spring Boot's flexibility, providing a simple yet functional web application on WebSphere Liberty Profile.