Desplegando Spring Boot en WebSphere Liberty Profile

Aquí tienes una guía completa para ejecutar una aplicación simple de Spring Boot en WebSphere Liberty Profile (WLP). Este ejemplo crea una aplicación Spring Boot con un punto final REST básico que devuelve "Hello World!"y la despliega en WLP utilizando el directorio `dropins`. Supongamos que ya has configurado WLP con la característica `javaee-8.0` habilitada, ya que esto proporciona el soporte de Servlet necesario.

Guía Paso a Paso

1. Crear una Aplicación Spring Boot Comienza creando un nuevo proyecto Spring Boot utilizando Spring Initializr (o tu método preferido). Usaremos Maven, Java y la dependencia Spring Web para construir una aplicación web simple.

- **Generar el Proyecto:**
  - Ve a Spring Initializr.
  - Configura lo siguiente:
    - **Proyecto:** Maven
    - **Idioma:** Java
    - **Versión de Spring Boot:** 2.7.x (o la versión estable más reciente)
    - **Grupo:** `com.example`
    - **Artefacto:** `demo`
    - **Dependencias:** Spring Web
  - Haz clic en "Generate"para descargar el archivo ZIP del proyecto, luego descomprímelo y ábrelo en tu IDE.
- **Agregar un Controlador REST Simple:** Dentro de `src/main/java/com/example/demo`, crea un archivo llamado `HelloController.java` con el siguiente contenido:

```
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!";
```

```
    }
}
```

Esto crea un punto final REST en la ruta raíz (`/`) que devuelve "Hello World!"como texto plano.

2. Configurar la Aplicación para Despliegue WAR Por defecto, Spring Boot empaqueta las aplicaciones como archivos JAR con un servidor incrustado (por ejemplo, Tomcat). Para desplegar en WLP, necesitamos empaquetarlo como un archivo WAR y configurarlo para que funcione con el contenedor Servlet de WLP.

- **Modificar la Clase Principal de la Aplicación:** Edita `src/main/java/com/example/demo/DemoApplication.java` para extender `SpringBootServletInitializer`, lo que permite que la aplicación se ejecute en un contenedor Servlet externo como WLP:

```java
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);
    }
}
```

- **Actualizar `pom.xml` para Empaquetado WAR:** Abre `pom.xml` y realiza estos cambios:
  - Establece el empaquetado a WAR agregando esta línea cerca de la parte superior (debajo de `<modelVersion>`):

    `<packaging>war</packaging>`
  - Marca la dependencia de Tomcat incrustado como `provided` para que no se incluya en el WAR (WLP proporciona su propio contenedor Servlet). Modifica la dependencia `spring-boot-starter-web` (que incluye Tomcat) de la siguiente manera:

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

Agrega esto debajo de ella:

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

La sección de dependencias de tu `pom.xml` debería verse algo así:

```
<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>
  <!-- Otras dependencias como spring-boot-starter-test pueden permanecer -->
</dependencies>
```

3. **Construir el Archivo WAR** Compila y empaqueta la aplicación en un archivo WAR utilizando Maven.

   - **Ejecutar el Comando de Construcción:** Desde el directorio raíz del proyecto (donde se encuentra `pom.xml`), ejecuta:

     ```
     mvn clean package
     ```

     Esto genera el archivo WAR en el directorio `target`, por ejemplo, `target/demo-0.0.1-SNAPSHOT.war`.

   - **Renombrar el Archivo WAR (Opcional):** Para una URL más limpia, renombra el archivo WAR a `myapp.war`:

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

     Esto simplifica la raíz del contexto a `/myapp` en lugar de `/demo-0.0.1-SNAPSHOT`.

4. Desplegar el Archivo WAR en WLP Despliega el archivo WAR en WLP utilizando el directorio `dropins`, lo que habilita el despliegue automático.

- **Localizar el Directorio `dropins`:** Encuentra el directorio `dropins` del servidor WLP. Si WLP está instalado en `/opt/ibm/wlp` y tu servidor se llama `myServer`, la ruta es:

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

- **Copiar el Archivo WAR:** Mueve el archivo WAR al directorio `dropins`:

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

- **Iniciar el Servidor (si No Está en Ejecución):** Si WLP no está en ejecución, inícialo:

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

  Si ya está en ejecución, detectará y desplegará el archivo WAR automáticamente.

- **Verificar el Despliegue:** Verifica los registros del servidor o la consola para un mensaje como:

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

  - Los registros están en `/opt/ibm/wlp/usr/servers/myServer/logs/console.log` (modo de fondo) o se muestran en la terminal (modo de primer plano con `./server run myServer`).

5. Acceder a la Aplicación Prueba la aplicación Spring Boot desplegada en un navegador.

- **Abrir tu Navegador:** Navega a:

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

  - `9080` es el puerto HTTP predeterminado de WLP.
  - `/myapp` es la raíz del contexto del archivo WAR.
  - `/` coincide con el `@GetMapping("/")` en el controlador.

- **Resultado Esperado:** Deberías ver:

  ```
  Hello World!
  ```

  mostrado como texto plano.

Notas

- **Raíz del Contexto:** La raíz del contexto (`/myapp`) se deriva del nombre del archivo WAR. Ajusta renombrando el archivo WAR según sea necesario.
- **Número de Puerto:** WLP usa `9080` por defecto para HTTP. Si tu servidor usa un puerto diferente, actualiza la URL en consecuencia.
- **Versión de Java:** Asegúrate de que WLP y tu entorno de construcción usen una versión de Java compatible (por ejemplo, Java 8 o 11 para Spring Boot 2.7.x).
- **Características de WLP:** La característica `javaee-8.0` incluye soporte para Servlet 4.0, que es compatible con los requisitos de Spring Boot 2.7.x (Servlet 3.1+).
- **Solución de Problemas:**
  - Si la página no se carga, verifica `/opt/ibm/wlp/usr/servers/myServer/logs/messages.log` para errores.
  - Verifica que el archivo WAR esté en `dropins` y que el servidor esté en ejecución.

Resumen

Esta guía demuestra cómo: 1. Crear una aplicación Spring Boot con un punto final REST. 2. Configurarla para el despliegue WAR extendiendo `SpringBootServletInitializer` y estableciendo Tomcat como `provided`. 3. Construir y desplegar el archivo WAR en el directorio `dropins` de WLP. 4. Acceder a la salida “Hello World!” en `http://localhost:9080/myapp/`.

Este enfoque aprovecha las capacidades de despliegue automático de WLP y la flexibilidad de Spring Boot, proporcionando una aplicación web simple pero funcional en WebSphere Liberty Profile.