Déployer Spring Boot sur WebSphere Liberty Profile

Voici un guide complet pour exécuter une application Spring Boot simple sur WebSphere Liberty Profile (WLP). Cet exemple crée une application Spring Boot avec un point de terminaison REST de base qui retourne “Hello World!”et la déploie sur WLP en utilisant le répertoire `dropins`. Nous supposerons que vous avez déjà configuré WLP avec la fonctionnalité `javaee-8.0` activée, car cela fournit le support Servlet nécessaire.

Guide Étape par Étape

1. Créer une Application Spring Boot Commencez par créer un nouveau projet Spring Boot en utilisant Spring Initializr (ou votre méthode préférée). Nous utiliserons Maven, Java et la dépendance Spring Web pour construire une application web simple.

- **Générer le Projet :**
  - Allez sur Spring Initializr.
  - Configurez les éléments suivants :
    - **Projet :** Maven
    - **Langage :** Java
    - **Version Spring Boot :** 2.7.x (ou la dernière version stable)
    - **Groupe :** `com.example`
    - **Artéfact :** `demo`
    - **Dépendances :** Spring Web
  - Cliquez sur “Generate”pour télécharger le projet ZIP, puis décompressez-le et ouvrez-le dans votre IDE.
- **Ajouter un Contrôleur REST Simple :** À l’intérieur de `src/main/java/com/example/demo`, créez un fichier nommé `HelloController.java` avec ce contenu :

```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!";
```

```
    }
}
```

Cela crée un point de terminaison REST à la racine (/) qui retourne “Hello World!”en tant que texte brut.

2. Configurer l’Application pour le Déploiement WAR Par défaut, Spring Boot emballe les applications en fichiers JAR avec un serveur intégré (par exemple, Tomcat). Pour le déployer sur WLP, nous devons l’emballer en tant que fichier WAR et le configurer pour fonctionner avec le conteneur Servlet de WLP.

- **Modifier la Classe Principale de l’Application :** Éditez `src/main/java/com/example/demo/DemoApplication.jav` pour étendre `SpringBootServletInitializer`, ce qui permet à l’application de s’exécuter dans un conteneur Servlet externe comme 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);
    }
}
```

- **Mettre à Jour `pom.xml` pour le Packaging WAR :** Ouvrez `pom.xml` et apportez ces modifications :
  - Définissez le packaging en WAR en ajoutant cette ligne près du début (sous `<modelVersion>`) :

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

  - Marquez la dépendance Tomcat intégrée comme `provided` pour qu’elle ne soit pas incluse dans le WAR (WLP fournit son propre conteneur Servlet). Modifiez la dépendance `spring-boot-starter-web` (qui inclut Tomcat) comme ceci :

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

Ajoutez ceci en dessous :

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

Votre section dépendances de `pom.xml` devrait maintenant ressembler à ceci :

```
<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>
  <!-- D'autres dépendances comme spring-boot-starter-test peuvent rester -->
</dependencies>
```

3. **Construire le Fichier WAR** Compilez et emballez l'application en un fichier WAR en utilisant Maven.

   - **Exécuter la Commande de Construction :** Depuis le répertoire racine du projet (où se trouve `pom.xml`), exécutez :

     ```
     mvn clean package
     ```

     Cela génère le fichier WAR dans le répertoire `target`, par exemple, `target/demo-0.0.1-SNAPSHOT.war`.

   - **Renommer le Fichier WAR (Optionnel) :** Pour une URL plus propre, renommez le fichier WAR en `myapp.war` :

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

     Cela simplifie la racine du contexte en `/myapp` au lieu de `/demo-0.0.1-SNAPSHOT`.

4. Déployer le Fichier WAR sur WLP Déployez le fichier WAR sur WLP en utilisant le répertoire `dropins`, ce qui permet un déploiement automatique.

- **Localiser le Répertoire `dropins` :** Trouvez le répertoire `dropins` de votre serveur WLP. Si WLP est installé à `/opt/ibm/wlp` et votre serveur s'appelle `myServer`, le chemin est :

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

- **Copier le Fichier WAR :** Déplacez le fichier WAR dans le répertoire `dropins` :

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

- **Démarrer le Serveur (si Non Déjà Démarré) :** Si WLP n'est pas en cours d'exécution, démarrez-le :

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

  S'il est déjà en cours d'exécution, il détectera et déploiera automatiquement le fichier WAR.

- **Vérifier le Déploiement :** Vérifiez les journaux du serveur ou la console pour un message comme :

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

  - Les journaux se trouvent dans `/opt/ibm/wlp/usr/servers/myServer/logs/console.log` (mode arrière-plan) ou sont affichés dans le terminal (mode avant-plan avec `./server run myServer`).

5. Accéder à l'Application Testez l'application Spring Boot déployée dans un navigateur.

- **Ouvrir Votre Navigateur :** Accédez à :

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

  - `9080` est le port HTTP par défaut de WLP.
  - `/myapp` est la racine du contexte du fichier WAR.
  - `/` correspond à l'annotation `@GetMapping("/")` dans le contrôleur.

- **Résultat Attendu :** Vous devriez voir :

  ```
  Hello World!
  ```

  affiché en tant que texte brut.

Notes

- **Racine du Contexte :** La racine du contexte (`/myapp`) est dérivée du nom du fichier WAR. Ajustez-la en renommant le fichier WAR si nécessaire.
- **Numéro de Port :** WLP utilise `9080` par défaut pour HTTP. Si votre serveur utilise un port différent, mettez à jour l'URL en conséquence.
- **Version Java :** Assurez-vous que WLP et votre environnement de construction utilisent une version Java compatible (par exemple, Java 8 ou 11 pour Spring Boot 2.7.x).
- **Fonctionnalités WLP :** La fonctionnalité `javaee-8.0` inclut le support Servlet 4.0, qui est compatible avec les exigences de Spring Boot 2.7.x (Servlet 3.1+).
- **Dépannage :**
  - Si la page ne se charge pas, vérifiez `/opt/ibm/wlp/usr/servers/myServer/logs/messages.log` pour les erreurs.
  - Vérifiez que le fichier WAR est dans `dropins` et que le serveur est en cours d'exécution.

Résumé

Ce guide démontre comment : 1. Créer une application Spring Boot avec un point de terminaison REST. 2. La configurer pour le déploiement WAR en étendant `SpringBootServletInitializer` et en définissant Tomcat comme `provided`. 3. Construire et déployer le fichier WAR dans le répertoire `dropins` de WLP. 4. Accéder à la sortie "Hello World!"à `http://localhost:9080/myapp/`.

Cette approche tire parti des capacités de déploiement automatique de WLP et de la flexibilité de Spring Boot, fournissant une application web simple mais fonctionnelle sur WebSphere Liberty Profile.