

Spring Boot auf WebSphere Liberty Profile bereitstellen

Hier ist eine vollständige Anleitung zum Ausführen einer einfachen Spring Boot-Anwendung auf WebSphere Liberty Profile (WLP). Dieses Beispiel erstellt eine Spring Boot-Anwendung mit einem grundlegenden REST-Endpunkt, der "Hello World!" zurückgibt, und stellt sie auf WLP mithilfe des `dropins`-Verzeichnisses bereit. Wir gehen davon aus, dass WLP bereits mit dem `javaee-8.0`-Feature eingerichtet ist, da dies die notwendige Servlet-Unterstützung bietet.

Schritt-für-Schritt-Anleitung

1. Erstellen Sie eine Spring Boot-Anwendung Erstellen Sie ein neues Spring Boot-Projekt mit Spring Initializr (oder Ihrer bevorzugten Methode). Wir verwenden Maven, Java und die Spring Web-Abhängigkeit, um eine einfache Webanwendung zu erstellen.

- **Projekt generieren:**

- Gehen Sie zu Spring Initializr.
- Konfigurieren Sie Folgendes:
 - * **Projekt:** Maven
 - * **Sprache:** Java
 - * **Spring Boot-Version:** 2.7.x (oder die neueste stabile Version)
 - * **Gruppe:** `com.example`
 - * **Artifact:** `demo`
 - * **Abhängigkeiten:** Spring Web
- Klicken Sie auf "Generate", um das Projekt-ZIP herunterzuladen, entpacken Sie es und öffnen Sie es in Ihrer IDE.

- **Fügen Sie einen einfachen REST-Controller hinzu:** Erstellen Sie innerhalb von `src/main/java/com/example/d` eine Datei mit dem Namen `HelloController.java` mit folgendem Inhalt:

```
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!";
    }
}

```

Dies erstellt einen REST-Endpunkt am Stammpfad (/), der "Hello World!" als Klartext zurückgibt.

2. Konfigurieren Sie die Anwendung für die WAR-Bereitstellung Standardmäßig verpackt Spring Boot Anwendungen als JAR-Dateien mit einem eingebetteten Server (z.B. Tomcat). Um auf WLP zu bereitstellen, müssen wir es als WAR-Datei verpacken und so konfigurieren, dass es mit dem Servlet-Container von WLP funktioniert.

- **Bearbeiten Sie die Hauptanwendungs-Klasse:** Bearbeiten Sie `src/main/java/com/example/demo/DemoApplication` um `SpringBootServletInitializer` zu erweitern, was es der App ermöglicht, in einem externen Servlet-Container wie WLP zu laufen:

```

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);
    }
}

```

- **Aktualisieren Sie `pom.xml` für die WAR-Verpackung:** Öffnen Sie `pom.xml` und nehmen Sie diese Änderungen vor:

- Setzen Sie die Verpackung auf WAR, indem Sie diese Zeile hinzufügen (unten `<modelVersion>`):

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

- Markieren Sie die eingebettete Tomcat-Abhängigkeit als `provided`, sodass sie nicht in der WAR enthalten ist (WLP stellt seinen eigenen Servlet-Container bereit). Ändern Sie die `spring-boot-starter-web`-Abhängigkeit (die Tomcat enthält) wie folgt:

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

Fügen Sie dies darunter hinzu:

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

Ihr Abschnitt `dependencies` in `pom.xml` sollte nun in etwa so aussehen:

```
<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>
  <!-- Andere Abhängigkeiten wie spring-boot-starter-test können bleiben -->
</dependencies>
```

3. Erstellen Sie die WAR-Datei Kompilieren und verpacken Sie die Anwendung in eine WAR-Datei mit Maven.

- **Führen Sie den Build-Befehl aus:** Vom Projektstammverzeichnis (wo `pom.xml` sich befindet), führen Sie aus:

```
mvn clean package
```

Dies erzeugt die WAR-Datei im `target`-Verzeichnis, z.B. `target/demo-0.0.1-SNAPSHOT.war`.

- **Benennen Sie die WAR-Datei um (optional):** Für eine sauberere URL benennen Sie die WAR-Datei in `myapp.war` um:

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

Dies vereinfacht den Kontextpfad zu `/myapp` anstatt `/demo-0.0.1-SNAPSHOT`.

4. Bereitstellen der WAR-Datei auf WLP Bereitstellen Sie die WAR-Datei auf WLP mithilfe des dropins-Verzeichnisses, was eine automatische Bereitstellung ermöglicht.

- **Lokalisieren Sie das dropins-Verzeichnis:** Finden Sie das dropins-Verzeichnis Ihres WLP-Servers. Wenn WLP bei `/opt/ibm/wlp` installiert ist und Ihr Server `myServer` heißt, ist der Pfad:

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

- **Kopieren Sie die WAR-Datei:** Bewegen Sie die WAR-Datei in das dropins-Verzeichnis:

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

- **Starten Sie den Server (falls er nicht läuft):** Wenn WLP nicht läuft, starten Sie ihn:

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

Wenn er bereits läuft, wird er die WAR-Datei automatisch erkennen und bereitstellen.

- **Überprüfen Sie die Bereitstellung:** Überprüfen Sie die Serverprotokolle oder die Konsole auf eine Nachricht wie:

```
[AUDIT   ] CWWKT0016I: Webanwendung verfügbar (default_host): http://localhost:9080/myapp/
```

- Protokolle befinden sich in `/opt/ibm/wlp/usr/servers/myServer/logs/console.log` (Hintergrundmodus) oder werden im Terminal angezeigt (Vordergrundmodus mit `./server run myServer`).

5. Zugriff auf die Anwendung Testen Sie die bereitgestellte Spring Boot-Anwendung in einem Browser.

- **Öffnen Sie Ihren Browser:** Navigieren Sie zu:

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

- 9080 ist der Standard-HTTP-Port von WLP.
- `/myapp` ist der Kontextpfad aus dem Namen der WAR-Datei.
- `/` entspricht dem `@GetMapping("/")` im Controller.

- **Erwartetes Ergebnis:** Sie sollten sehen:

```
Hello World!
```

als Klartext angezeigt.

Hinweise

- **Kontextpfad:** Der Kontextpfad (/myapp) leitet sich aus dem Namen der WAR-Datei ab. Passen Sie ihn an, indem Sie die WAR-Datei entsprechend umbenennen.
 - **Portnummer:** WLP verwendet standardmäßig 9080 für HTTP. Wenn Ihr Server einen anderen Port verwendet, passen Sie die URL entsprechend an.
 - **Java-Version:** Stellen Sie sicher, dass WLP und Ihre Build-Umgebung eine kompatible Java-Version verwenden (z.B. Java 8 oder 11 für Spring Boot 2.7.x).
 - **WLP-Features:** Das `javaee-8.0`-Feature enthält die Unterstützung für Servlet 4.0, die mit den Anforderungen von Spring Boot 2.7.x (Servlet 3.1+) kompatibel ist.
 - **Fehlerbehebung:**
 - Wenn die Seite nicht geladen wird, überprüfen Sie `/opt/ibm/wlp/usr/servers/myServer/logs/messages.log` auf Fehler.
 - Stellen Sie sicher, dass die WAR-Datei sich im `dropins`-Verzeichnis befindet und der Server läuft.
-

Zusammenfassung

Diese Anleitung zeigt, wie Sie: 1. Eine Spring Boot-Anwendung mit einem REST-Endpunkt erstellen. 2. Sie für die WAR-Bereitstellung konfigurieren, indem Sie `SpringBootServletInitializer` erweitern und Tomcat auf `provided` setzen. 3. Die WAR-Datei in das `dropins`-Verzeichnis von WLP bereitstellen. 4. Die "Hello World!"-Ausgabe unter `http://localhost:9080/myapp/` aufrufen.

Diese Vorgehensweise nutzt die automatischen Bereitstellungsfähigkeiten von WLP und die Flexibilität von Spring Boot, um eine einfache, aber funktionale Webanwendung auf WebSphere Liberty Profile bereitzustellen.