Spring Boot を WebSphere Liberty Profile にデプロイする

ここには、WebSphere Liberty Profile (WLP) 上でシンプルな Spring Boot アプリケーションを実行するための完全なガイドがあります。この例では、基本的な REST エンドポイントを持つ Spring Boot アプリケーションを作成し、それを `dropins` ディレクトリを使用して WLP にデプロイします。WLP が `javaee-8.0` フェーチャーを有効にして設定されていることを前提としています。これは、必要な Servlet サポートを提供します。

ステップバイステップガイド

1. Spring Boot アプリケーションの作成 Spring Initializr（またはお好みの方法）を使用して新しい Spring Boot プロジェクトを作成します。Maven、Java、Spring Web 依存関係を使用してシンプルな Web アプリケーションを構築します。

- **プロジェクトの生成:**
  - Spring Initializr にアクセスします。
  - 以下のように設定します：
    - **プロジェクト:** Maven
    - **言語:** Java
    - **Spring Boot バージョン:** 2.7.x（または最新の安定版）
    - **グループ:** `com.example`
    - **アーティファクト:** `demo`
    - **依存関係:** Spring Web
  - 「Generate」をクリックしてプロジェクト ZIP をダウンロードし、解凍して IDE で開きます。
- **シンプルな REST コントローラーの追加:** `src/main/java/com/example/demo` 内に `HelloController.java` という名前のファイルを作成し、以下の内容を追加します：

```
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!";
    }
}
```

これは、ルートパス（/）に REST エンドポイントを作成し、プレーンテキストとして“Hello World!”を返します。

2. WAR デプロイ用のアプリケーションの設定 デフォルトでは、Spring Boot は埋め込みサーバー（例：Tomcat）を持つ JAR ファイルとしてアプリケーションをパッケージ化します。WLP にデプロイするには、WAR ファイルとしてパッケージ化し、WLP の Servlet コンテナと一緒に動作するように設定する必要があります。

- **メインアプリケーションクラスの修正:** `src/main/java/com/example/demo/DemoApplication.java` を編集して `SpringBootServletInitializer` を拡張し、外部の Servlet コンテナ（例：WLP）でアプリケーションを実行できるようにします：

```
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);
    }
}
```

- `pom.xml` **の WAR パッケージングの更新:** `pom.xml` を開き、以下の変更を行います：
  - 以下の行を `<modelVersion>` の下に追加してパッケージングを WAR に設定します：

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

  - 嵌め込み Tomcat 依存関係を `provided` にマークして、WAR に含めないようにします（WLP が自分の Servlet コンテナを提供します）。`spring-boot-starter-web` 依存関係（Tomcat を含む）を以下のように修正します：

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

以下をその下に追加します：

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

`pom.xml` の依存関係セクションは、以下のようになります：

```
<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>
  <!-- 他の依存関係（例：spring-boot-starter-test）はそのままにしておく -->
</dependencies>
```

3. WAR ファイルのビルド Maven を使用してアプリケーションを WAR ファイルにコンパイルしてパッケージ化します。

- **ビルドコマンドの実行:** プロジェクトのルートディレクトリ（`pom.xml` がある場所）から以下を実行します：

  ```
  mvn clean package
  ```

  これにより、`target` ディレクトリに WAR ファイルが生成されます（例：`target/demo-0.0.1-SNAPSHOT.war`）。

- **WAR ファイルの名前の変更（オプション）：** より簡潔な URL を得るために、WAR ファイルを `myapp.war` に名前を変更します：

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

  これにより、コンテキストルートが `/demo-0.0.1-SNAPSHOT` ではなく `/myapp` になります。

4. WLP に WAR ファイルをデプロイ `dropins` ディレクトリを使用して WAR ファイルを WLP にデプロイします。これにより、自動デプロイが有効になります。

- **`dropins` ディレクトリの検索:** WLP サーバーの `dropins` ディレクトリを検索します。WLP が `/opt/ibm/wlp` にインストールされ、サーバー名が `myServer` の場合、パスは以下の通りです：

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

- **WAR ファイルのコピー:** WAR ファイルを `dropins` ディレクトリに移動します：

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

- **サーバーの起動（実行中でない場合）：** WLP が実行中でない場合は、以下を実行して起動します：

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

  既に実行中の場合、WAR ファイルを自動的に検出してデプロイします。

- **デプロイの確認:** サーバーログまたはコンソールに以下のようなメッセージが表示されることを確認します：

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

  - ログは `/opt/ibm/wlp/usr/servers/myServer/logs/console.log` にあります（バックグラウンドモード）または、ターミナルに表示されます（フォアグラウンドモードで `./server run myServer`）。

5. アプリケーションへのアクセス　ブラウザでデプロイされた Spring Boot アプリケーションをテストします。

- **ブラウザを開く:** 以下にアクセスします：

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

  - `9080` は WLP のデフォルトの HTTP ポートです。
  - `/myapp` は WAR ファイル名からのコンテキストルートです。
  - `/` はコントローラーの `@GetMapping("/")` に一致します。

- **期待される結果:** プレーンテキストとして以下が表示されるはずです：

```
Hello World!
```

注意点

- **コンテキストルート:** コンテキストルート（`/myapp`）は WAR ファイル名から派生します。必要に応じて WAR ファイルの名前を変更して調整します。
- **ポート番号:** WLP はデフォルトで HTTP に `9080` を使用します。サーバーが異なるポートを使用している場合は、URL を適宜更新してください。
- **Java バージョン:** WLP とビルド環境が互換性のある Java バージョン（例：Spring Boot 2.7.x の場合は Java 8 または 11）を使用していることを確認します。

- **WLP フェーチャー:** `javaee-8.0` フェーチャーには Servlet 4.0 サポートが含まれており、Spring Boot 2.7.x の要件（Servlet 3.1+）と互換性があります。
- **トラブルシューティング:**
  - ページが読み込まれない場合は、`/opt/ibm/wlp/usr/servers/myServer/logs/messages.log` を確認してエラーを確認します。
  - WAR ファイルが `dropins` にあり、サーバーが実行中であることを確認します。

まとめ

このガイドでは、以下の手順を示します：1. REST エンドポイントを持つ Spring Boot アプリケーションを作成します。2. `SpringBootServletInitializer` を拡張し、Tomcat を `provided` に設定して WAR デプロイ用に設定します。3. WAR ファイルを WLP の `dropins` ディレクトリにビルドしてデプロイします。4. `http://localhost:9080/myapp/` で“Hello World!”出力にアクセスします。

このアプローチは、WLP の自動デプロイ機能と Spring Boot の柔軟性を活用し、WebSphere Liberty Profile 上でシンプルで機能的な Web アプリケーションを提供します。