Problèmes de démarrage d'application dans WebSphere Liberty Profile

Il semble que, après avoir rétrogradé votre application Spring Boot à la version 2.7.18 et l'avoir déployée sur WebSphere Liberty Profile (WLP), vous rencontrez un problème où l'application échoue à démarrer dans le délai d'attente par défaut, comme indiqué par le message dans `console.log` : *"l'application myapp n'a pas démarré en 30.xx secondes"*. Cela suggère que l'application met trop de temps à s'initialiser ou échoue à démarrer complètement. Ci-dessous, je vais vous guider à travers un processus étape par étape pour diagnostiquer et résoudre ce problème.

Étape 1 : Examiner les Journaux du Serveur pour des Indices

La première chose à faire est d'examiner les journaux du serveur en détail pour identifier la cause racine de l'échec de démarrage. Le fichier `console.log` fournit le message de dépassement de temps, mais il ne contient peut-être pas toute l'histoire. Vérifiez les fichiers journaux suivants dans le répertoire des journaux de votre serveur WLP (`/opt/ibm/wlp/usr/servers/myServer/logs/`):

- `messages.log` : Ce fichier contient souvent des messages d'ERREUR ou d'AVERTISSEMENT qui peuvent pointer des problèmes comme des dépendances manquantes, des erreurs de configuration ou des exceptions pendant le démarrage.
- `trace.log` : Si la traçabilité détaillée est activée, ce fichier pourrait fournir plus de contexte sur ce qui se passe pendant le déploiement.

Recherchez : - Des traces de pile ou des exceptions (par exemple, `ClassNotFoundException`, `NoSuchBeanDefinitionExcep
- Des messages concernant des ressources manquantes ou des bibliothèques incompatibles. - Des indications que le contexte de l'application a échoué à s'initialiser.

Si vous ne voyez pas assez de détails, vous pouvez augmenter le niveau de journalisation dans WLP en modifiant le fichier `server.xml`. Ajoutez ou mettez à jour l'élément `<logging>` comme ceci :

```
<logging traceSpecification="*=info:com.ibm.ws.webcontainer*=all" />
```

Redémarrez le serveur après avoir apporté ce changement, redeployez votre application et vérifiez à nouveau les journaux pour plus d'informations.

Étape 2 : Vérifier le Démarrage de l'Application avec la Journalisation

Étant donné qu'il s'agit d'une application Spring Boot, le problème pourrait être lié à l'échec de l'initialisation du contexte de l'application. Pour déterminer jusqu'où le processus de démarrage progresse, ajoutez une simple déclaration de journalisation à votre classe principale de l'application en utilisant une méthode `@PostConstruct`. Voici un exemple :

```
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;
import javax.annotation.PostConstruct;

@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);
    }

    @PostConstruct
    public void init() {
        System.out.println("Contexte de l'application initialisé");
    }
}
```

- Reconstruisez votre application (`mvn clean package`).
- Redeployez le fichier WAR dans le répertoire `dropins` de WLP.
- Vérifiez `console.log` pour le message `"Contexte de l'application initialisé"`.

Si ce message apparaît, le contexte de l'application se charge avec succès, et le problème pourrait être lié aux composants web ou à l'initialisation des servlets. S'il n'apparaît pas, le problème survient plus tôt pendant l'initialisation du contexte.

Étape 3 : Activer la Journalisation de Débogage dans Spring Boot

Pour obtenir plus de visibilité dans le processus de démarrage de Spring Boot, activez la journalisation de débogage en ajoutant un fichier de configuration. Créez ou modifiez `src/main/resources/application.properties` avec ce qui suit :

```
debug=true
```

- Reconstruisez et redeployez l'application.
- Vérifiez `console.log` (ou d'autres journaux) pour une sortie de débogage détaillée de Spring Boot.

Cela journalisera des informations sur la création de beans, l'auto-configuration et toute erreur survenant pendant le démarrage. Recherchez des indices sur ce qui pourrait être en attente ou échouer.

Étape 4 : Vérifier le Fichier WAR et la Configuration des Dépendances

Étant donné que vous déployez sur WLP, qui fournit son propre conteneur Servlet, assurez-vous que votre fichier WAR est correctement configuré pour un serveur externe :

- **Emballage WAR** : Dans votre `pom.xml`, confirmez que l'emballage est défini sur `war` :

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

- **Tomcat comme Fourniture** : Assurez-vous que le Tomcat intégré est exclu du fichier WAR, car WLP fournira le conteneur Servlet. Vérifiez votre `pom.xml` pour :

```
<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>
```

- **Compatibilité de l'API Servlet** : Spring Boot 2.7.18 utilise `javax.servlet:javax.servlet-api:4.0.1`, qui est compatible avec la fonction `javaee-8.0` de WLP (Servlet 4.0). Pour confirmer qu'il n'y a pas de dépendances en conflit, exécutez :

```
mvn dependency:tree
```

Recherchez toute version inattendue de l'API Servlet (par exemple, `jakarta.servlet-api`, qui est utilisée dans Spring Boot 3.x et incompatible avec `javaee-8.0`).

Si vous suspectez des problèmes de dépendances, dézippez le fichier WAR et inspectez `WEB-INF/lib` pour vous assurer qu'aucun JAR inattendu lié à Servlet n'est inclus.

Étape 5 : Tester Localement pour Isoler le Problème

Pour déterminer si le problème est spécifique à WLP ou à l'application elle-même, testez l'application localement en utilisant le Tomcat intégré :

```
mvn spring-boot:run
```

Si elle démarre avec succès et que vous pouvez accéder à vos points de terminaison (par exemple, un simple contrôleur REST `"Hello World!"`), le problème est probablement lié au déploiement WLP plutôt qu'au code de l'application.

Étape 6 : Ajuster le Délai d'Attente de Démarrage de WLP (Solution de Contournement Temporaire)

Si les journaux suggèrent que l'application démarre mais prend plus de 30 secondes, vous pouvez augmenter le délai d'attente de démarrage dans `server.xml` de WLP :

```
<applicationMonitor startTimeout="60s" />
```

- Redeployez l'application et surveillez les journaux.
- Si elle démarre après le délai d'attente étendu, cela confirme un processus de démarrage lent, et vous devriez optimiser l'application (par exemple, réduire le balayage des composants ou les tâches d'initialisation).

Cependant, il s'agit d'une solution de contournement—idéalement, une application simple devrait démarrer en moins de 30 secondes, alors continuez à enquêter sur la cause racine.

Étape 7 : Simplifier et Comparer avec un Nouveau Projet

Si le problème persiste, créez un projet Spring Boot 2.7.18 minimal pour tester le déploiement sur WLP : 1. Utilisez Spring Initializr avec : - Spring Boot 2.7.18 - Java (correspondant à votre version WLP, par exemple 8 ou 11) - Dépendance : Spring Web 2. Ajoutez un contrôleur REST de base :

```
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!";
    }
}
```

3. Configurez-le pour le déploiement WAR (étendez `SpringBootServletInitializer` comme montré précédemment).
4. Construisez le fichier WAR (`mvn clean package`) et déployez-le dans le répertoire `dropins` de WLP.

Si ce nouveau projet démarre avec succès, comparez son `pom.xml`, sa classe principale et sa configuration avec votre projet original pour identifier les différences causant l'échec.

Étape 8 : Dépannage Supplémentaire de WLP

Si aucune des étapes ci-dessus ne résout le problème, envisagez ces vérifications spécifiques à WLP : - **Nettoyer le Répertoire de Travail** : Supprimez les données en cache qui pourraient interférer avec le déploiement : `bash rm -rf /opt/ibm/wlp/usr/servers/myServer/workarea` Ensuite, redémarrez le serveur et redeployez. - **Vérifier la Version de WLP** : Exécutez `/opt/ibm/wlp/bin/productInfo version` pour confirmer que votre version de WLP prend en charge les déploiements Spring Boot 2.7.x avec `javaee-8.0`. - **Version de Java** : Assurez-vous que la version de Java utilisée par WLP (vérifiez avec `java -version` dans l'environnement WLP) correspond à une version prise en charge par Spring Boot 2.7.18 (Java 8, 11 ou 17).

Correction Recommandée

Sur la base des informations fournies, le problème le plus probable est un problème pendant l'initialisation du contexte de l'application ou une incompatibilité subtile avec WLP. Voici comment le résoudre :

1. **Commencer par la Journalisation** :
    - Ajoutez la journalisation `@PostConstruct` (Étape 2) et activez `debug=true` (Étape 3).
    - Redeployez et vérifiez `console.log` et `messages.log` pour des erreurs ou là où le démarrage est en attente.
2. **Valider la Configuration** :
    - Confirmez la configuration du fichier WAR et des dépendances (Étape 4).
    - Testez localement (Étape 5) pour écarter les problèmes au niveau de l'application.
3. **Itérer et Simplifier** :
    - Si nécessaire, testez un projet minimal (Étape 7) pour isoler le problème.

En suivant ces étapes, vous devriez pouvoir identifier la cause—qu'il s'agisse d'un conflit de dépendances, d'un composant à démarrage lent ou d'un problème de configuration WLP—and le résoudre en conséquence. Si vous rencontrez toujours des difficultés, partagez des messages d'erreur spécifiques des journaux pour obtenir une assistance supplémentaire !