Application Startup Issues in WebSphere Liberty Profile

It seems that after downgrading your Spring Boot application to version 2.7.18 and deploying it to WebSphere Liberty Profile (WLP), you're encountering an issue where the application fails to start within the default timeout period, as indicated by the message in `console.log`: *"the application myapp has not started in 30.xx seconds"*. This suggests that the application is either taking too long to initialize or failing to start altogether. Below, I'll guide you through a step-by-step process to diagnose and fix this issue.

Step 1: Investigate Server Logs for Clues

The first thing to do is to examine the server logs in more detail to identify the root cause of the startup failure. The `console.log` file provides the timeout message, but it might not contain the full story. Check the following log files in your WLP server's log directory (`/opt/ibm/wlp/usr/servers/myServer/logs/`):

- **`messages.log`**: This file often contains ERROR or WARNING messages that can pinpoint issues like missing dependencies, configuration errors, or exceptions during startup.
- **`trace.log`**: If detailed tracing is enabled, this file might provide more context about what's happening during deployment.

Look for: - Stack traces or exceptions (e.g., `ClassNotFoundException`, `NoSuchBeanDefinitionException`). - Messages about missing resources or incompatible libraries. - Indications that the application context failed to initialize.

If you don't see enough detail, you can increase the logging level in WLP by modifying the `server.xml` file. Add or update the `<logging>` element like this:

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

Restart the server after making this change, redeploy your application, and check the logs again for more information.

Step 2: Verify Application Startup with Logging

Since this is a Spring Boot application, the issue might be related to the application context failing to initialize. To determine how far the startup process gets, add a simple log statement to your main application class using a `@PostConstruct` method. Here's an example:

```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;
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("Application context initialized");
    }
}
```

- Rebuild your application (`mvn clean package`).
- Redeploy the WAR file to WLP's `dropins` directory.
- Check `console.log` for the message `"Application context initialized"`.

If this message appears, the application context is loading successfully, and the issue might be related to web components or servlet initialization. If it doesn't appear, the problem occurs earlier during context initialization.

Step 3: Enable Debug Logging in Spring Boot

To get more visibility into Spring Boot's startup process, enable debug logging by adding a configuration file. Create or edit `src/main/resources/application.properties` with the following:

```
debug=true
```

- Rebuild and redeploy the application.
- Check `console.log` (or other logs) for detailed debug output from Spring Boot.

This will log information about bean creation, auto-configuration, and any errors that occur during startup. Look for clues about what might be hanging or failing.

Step 4: Verify WAR File and Dependency Configuration

Since you're deploying to WLP, which provides its own Servlet container, ensure your WAR file is correctly configured for an external server:

- **WAR Packaging**: In your `pom.xml`, confirm that the packaging is set to `war`:

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

- **Tomcat as Provided**: Ensure that the embedded Tomcat is excluded from the WAR file, as WLP will provide the Servlet container. Check your `pom.xml` for:

```xml
<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>
```

- **Servlet API Compatibility**: Spring Boot 2.7.18 uses `javax.servlet:javax.servlet-api:4.0.1`, which is compatible with WLP's `javaee-8.0` feature (Servlet 4.0). To confirm there are no conflicting dependencies, run:

```
mvn dependency:tree
```

Look for any unexpected Servlet API versions (e.g., `jakarta.servlet-api`, which is used in Spring Boot 3.x and incompatible with `javaee-8.0`).

If you suspect dependency issues, unzip the WAR file and inspect `WEB-INF/lib` to ensure no unexpected Servlet-related JARs are included.

Step 5: Test Locally to Isolate the Issue

To determine if the problem is specific to WLP or the application itself, test the application locally using the embedded Tomcat:

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

If it starts successfully and you can access your endpoints (e.g., a simple `"Hello World!"` REST controller), the issue is likely related to the WLP deployment rather than the application code.

Step 6: Adjust WLP Startup Timeout (Temporary Workaround)

If the logs suggest that the application is starting but taking longer than 30 seconds, you can increase the startup timeout in WLP's `server.xml`:

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

- Redeploy the application and monitor the logs.
- If it starts after the extended timeout, this confirms a slow startup process, and you should optimize the application (e.g., reduce component scanning or initialization tasks).

However, this is a workaround—ideally, a simple application should start within 30 seconds, so continue investigating the root cause.

Step 7: Simplify and Compare with a New Project

If the issue persists, create a minimal Spring Boot 2.7.18 project to test deployment on WLP: 1. Use Spring Initializr with: - Spring Boot 2.7.18 - Java (matching your WLP version, e.g., 8 or 11) - Dependency: Spring Web 2. Add a basic REST controller:

```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!";
    }
}
```

3. Configure it for WAR deployment (extend `SpringBootServletInitializer` as shown earlier).
4. Build the WAR file (`mvn clean package`) and deploy it to WLP's `dropins` directory.

If this new project starts successfully, compare its `pom.xml`, main class, and configuration with your original project to identify differences causing the failure.

Step 8: Additional WLP Troubleshooting

If none of the above resolves the issue, consider these WLP-specific checks: - **Clean the Work Directory**: Remove cached data that might interfere with deployment: `bash    rm -rf /opt/ibm/wlp/usr/servers/myServer/workar` Then restart the server and redeploy. - **Check WLP Version**: Run `/opt/ibm/wlp/bin/productInfo version` to confirm your WLP version supports Spring Boot 2.7.x deployments with `javaee-8.0`. - **Java Version**: Ensure the Java version used by WLP (check with `java -version` in the WLP environment) matches one supported by Spring Boot 2.7.18 (Java 8, 11, or 17).

Recommended Fix

Based on the information provided, the most likely issue is a problem during application context initialization or a subtle configuration mismatch with WLP. Here's how to fix it:

1. **Start with Logging**:
   - Add the `@PostConstruct` logging (Step 2) and enable `debug=true` (Step 3).
   - Redeploy and check `console.log` and `messages.log` for errors or where the startup hangs.
2. **Validate Configuration**:
   - Confirm the WAR file setup and dependencies (Step 4).
   - Test locally (Step 5) to rule out application-level issues.
3. **Iterate and Simplify**:
   - If needed, test a minimal project (Step 7) to isolate the problem.

By following these steps, you should pinpoint the cause—whether it's a dependency conflict, a slow-starting component, or a WLP configuration issue—and resolve it accordingly. If you still encounter difficulties, share specific error messages from the logs for further assistance!