Title: spring-boot-devtools Automatic Restart not working

Post Body:

I have a working Spring Boot 2.25 application built with mvn. As per this documentation I add

```
<dependencies>     <dependency>         <groupId>org.springframework.boot</groupId>         <artifactId>spring-boot-devtools</
```

From the documentation:

> As DevTools monitors classpath resources, the only way to trigger a restart is to update the classpath. The way in which you cause the classpath to be updated depends on the IDE that you are using. In Eclipse, saving a modified file causes the classpath to be updated and triggers a restart. In IntelliJ IDEA, building the project (Build -> Build Project) has the same effect.

With the application running I tried a simple

```
touch /path/to/app.jar
```

expecting the application to restart but nothing happened.

Okay, so maybe it's doing something smarter. I modified some source .java, recompiled the .jar, and `cp`'d it to replace the running .jar file and... nothing happened.

Also from the documentation

> DevTools relies on the application context's shutdown hook to close it during a restart. It does not work correctly if you have disabled the shutdown hook (SpringApplication.setRegisterShutdownHook(false)).

I am not doing this.

> DevTools needs to customize the ResourceLoader used by the ApplicationContext. If your application provides one already, it is going to be wrapped. Direct override of the getResource method on the ApplicationContext is not supported.

I am not doing this.

I am running this in a Docker container, if that matters. From the documentation:

> Developer tools are automatically disabled when running a fully packaged application. If your application is launched from java -jar or if it is started from a special classloader, then it is considered a "production application". If that does not apply to you (i.e. if you run your application from a container), consider excluding devtools or set the -Dspring.devtools.restart.enabled=false system property.

I don't understand what this means or if it is relevant.

I want to recompile a .jar and replace it in the running docker container and trigger and application restart without restarting the container. How can I do this?

EDIT: I am using `mvn` to rebuild the jar, then `docker cp` to replace it in the running container. (IntelliJ IDEA claims to rebuild the project, but the jar files are actually *not* touched, but that's another story.) I am looking for a non-IDE-specific solution.

Accepted Answer: None

Highest Rated Answer:

The Spring Boot Devtools offers for Spring Boot applications the functionality that usually is available in IDEs like IntelliJ in which you have the ability to, for example, restart an application or force a live browser reload when certain classes or resources change. This can be very useful in the development phase of your application.

It is typically used in conjunction with an IDE in such a way that it will be launched with the rest of your application by Spring Boot when detected in the classpath and if it is not disabled.

Although you can configure it to monitor further resources, it will usually look for changes in your application code, in your classes and resources.

It is important to say that, AFAIK, Devtools will monitor your own classes and resources in an *exploded* way, I mean, the restart process will not work if you overwrite your whole application jar, only if you overwrite some resources in your `classes` directory.

This functionality can be tested with Maven. Please, consider download a simple blueprint from Spring Initializr, with Spring Boot, Spring Boot Devtools and Spring Web, for example - in order to keep the application running. From a terminal, in the directory that contains the `pom.xml` file, run your application, for instance, with the help of the `spring-boot-maven-plugin` plugin included in the `pom.xml`:

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

The command will download the project dependencies, compile and run your application.

Now, perform any modification in your source code, either in your classes or in your resources and, from another terminal, in the same directory, recompile your resources:

```
mvn compile
```

If you look at the first terminal window you will see that the application is restarted to reflect the changes.

If you are using docker for your application deployment, try reproducing this behavior can be tricky.

On one hand, I do not know if it makes sense, but you can try creating a maven based image and run your code inside, just as described above. Your `Dockerfile` can look similar to this:

```
FROM maven:3.5-jdk-8 as maven  WORKDIR /app  # Copy project pom COPY ./pom.xml ./pom.xml  # Fetch (and cache) dependencies RUN
```

With this setup, you can copy with `docker cp` your resources to the `/app/target` directory and it will trigger an application restart. As an alternative, consider mounting a volume in your container instead of using `docker cp`.

Much better, and taking into account the fact that overwriting your application jar will probably not work, you can try to copy both your classes and library dependencies, and run your application in a exploded way. Consider the following `Dockerfile`:

```
FROM maven:3.5-jdk-8 as maven  WORKDIR /app  # Copy your project pom COPY ./pom.xml ./pom.xml  # Fetch (and cache) dependencie
```

The important line in the `Dockerfile` is this:

```
mvn clean compile dependency:copy-dependencies -Dspring-boot.repackage.skip=true
```

It will instruct maven to compile your resources and copy the required libraries. Although redundant for the typical Maven phase in which the `spring-boot-maven-plugin` `repackage` goal runs, the flag `spring-boot.repackage.skip=true` will instruct this plugin to not repackage the application.

With this `Dockerfile`, build you image (let's tag it `devtools-demo`, for example):

```
docker build -t devtools-demo .
```

And run it:

```
docker run devtools-demo:latest
```

With this setup, if now you change your classes and/or resources, and run `mvn` locally:

```
mvn compile
```

you should be able to force the restart mechanism in your container with the following `docker cp` command:

```
docker cp classes <container name>:/app/classes
```

Please, again, consider mounting a volume in your container instead of using `docker cp`.

I tested the setup and it worked properly.

The important think to keep in mind is to replace your exploded resources, not the whole application jar.

As another option, you can take an approach similar to the one indicated in your comments and run your Devtools in remote mode:

```
FROM maven:3.5-jdk-8 as maven  WORKDIR /app  # Copy project pom COPY ./pom.xml ./pom.xml  # Fetch (and cache) dependencies RUN
```

For the Spring Boot Devtools remote mode to work properly, you need several things (some of them pointed out by Opri as well in his/her answer).

First, you need to configure the `spring-boot-maven-plugin` to include the devtools in your application jar (it will be excluded otherwise, by default):

```
<plugin>   <groupId>org.springframework.boot</groupId>   <artifactId>spring-boot-maven-plugin</artifactId>   <configuration>
```

Then, you need to setup a value for the configuration property `spring.devtools.remote.secret`. This property has to do with the way remote debugging works in Spring Boot Devtools.

The remote debugging functionality consists of two parts, a client and a server. Basically, the client is a copy of your server code, and it uses the value of the `spring.devtools.remote.secret` configuration property to authenticate itself against the server.

This client code should be run from an IDE, and you attach your IDE debugging process to a local server exposed from that client.

Every change performed in the client monitored resources, remember, the same as in your server, is pushed to the remote server and it will trigger a restart if necessary.

As you can see, this functionality is again more appropriate from a development point of view.

If you need to actually restart your application by overwriting your jar application file, maybe a better approach will be to configure your docker container to run a shell script in your `ENTRYPOINT` or `CMD`. This shell script will monitor a copy of your jar, in a certain directory. If that resource changes, as a consequence of your

`docker cp`, this shell script will stop the current running application version - this application is supposed to run from a different location to avoid problems when updating the jar -, replace the current jar with the new one, and then start the new application version. Not the same, but please, consider read this related SO answer.

In any case, when you run an application in a container, you are trying to provide a consistent and platform independent way of deployment for it. From this perspective, instead of monitoring changes in your docker container, a more convenient approach may be to generate and to deploy a new version of your container image with those new changes. This process can be automated greatly using tools like Jenkins, Travis, etcetera. These tools allow you to define CI/CD pipelines that, in response to a code commit, for example, can generate on the fly a docker image with your code and, it configured accordingly, deploy later this image to services like some docker flavor or Kubernetes, on premises or in the cloud. Some of them, especially Kubernetes, but swarm an even docker compose as well, will allow you to perform rolling updates without or with minimal application service interruption.

To conclude, probably it will not fit your needs, but be aware that you can use `spring-boot-starter-actuator` directly or with Spring Boot Admin, for instance, to restart your application.

Finally, as already indicated in the Spring Boot Devtools documentation, you can try a different option, not based on restart but in application reload, in hot swapping. This functionality is offered by commercial products like JRebel although there are some open sources alternatives as well, mainly dcevm and the HotswapAgent. This related article provides some insight in how these last two products work. This Github project provides complementary information about how to run it in docker containers.