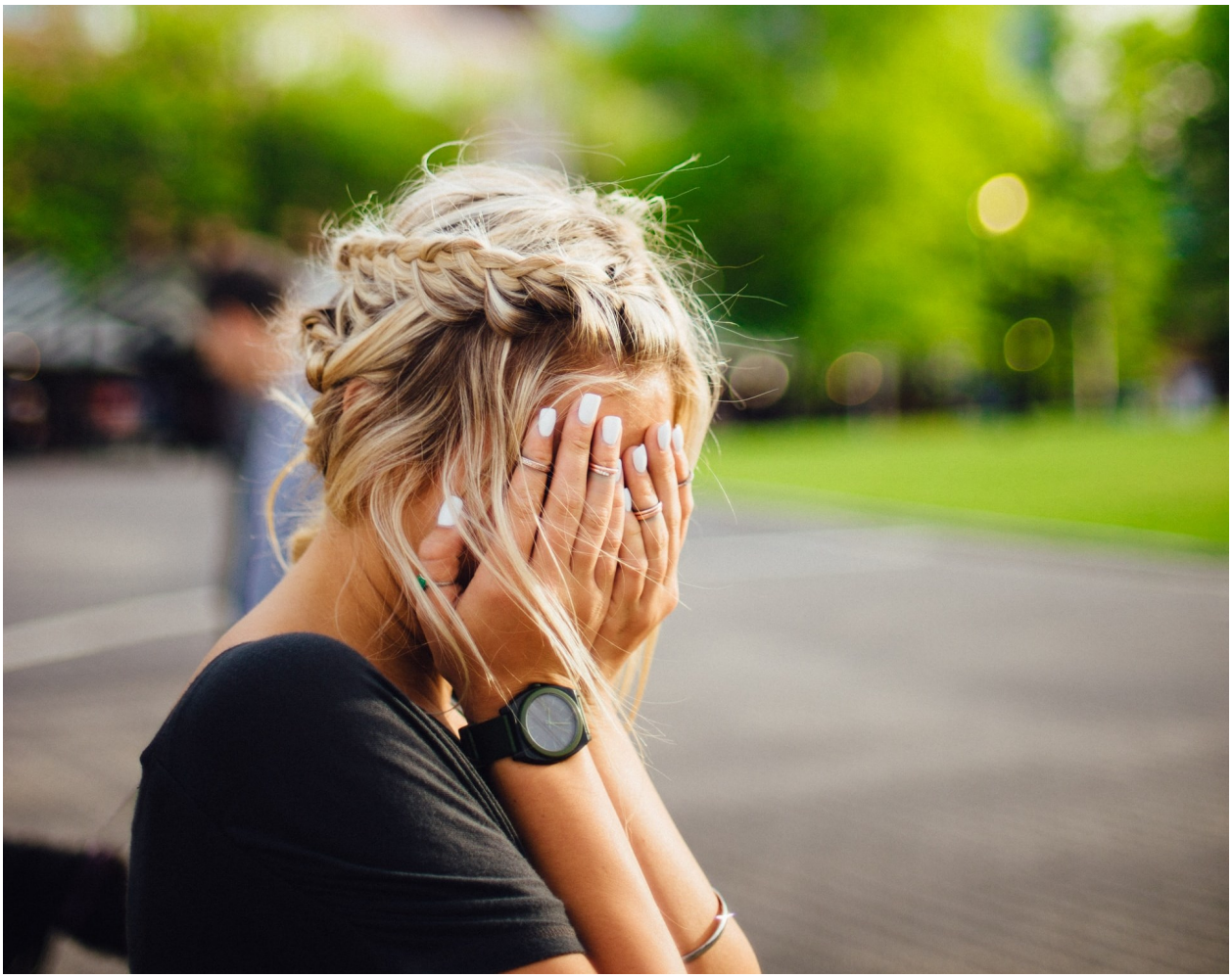
Use Case: Fixing a Java version conflict on a Linux host

June 29, 2022



Fixing version conflicts can be a very annoying process. On a computer, processes usually rely on libraries, and often on specific versions of those libraries. When new versions get released and you upgrade your system, applications that used to work may stop working. This is what became known in the late 1990s as "DLL hell". Since then, operating systems and vendors in general have become much better at limiting this type of conflict. We have containers that host all the required dependencies packaged as a neat little bundle, we have version tags for dynamic libraries, separate folder

structures, and update systems that attempt to check compatibility before upgrading things. But even with all this in place, conflicts can still occur. In this post I will go over a recent issue I had to solve regarding a conflicting Java version.

The software in this use case ran on a particular Amazon Linux instance and required a recent version of Java. It was running fine, and no update had happened to the application. However, one day after doing regular software updates for the OS, the application wouldn't start anymore. This was the error message:

java.lang.UnsupportedClassVersionError: net/minecraft/bundler/Main has been compiled by a more recent version of the Java Runtime (class file version 61.0), this version of the Java Runtime only recognizes class file versions up to 55.0

As you can see, it's obvious that this is a dependency issue, but it isn't exactly clear how to fix it. Java comes with two different version tags, so the first thing to do was figuring out which version was installed. By running this command, I could see the potential issue:

```
$ /usr/bin/java --version
openjdk 11.0.15 2022-04-19 LTS
OpenJDK Runtime Environment Corretto-11.0.15.9.1 (build 11.0.15+9-LTS)
OpenJDK 64-Bit Server VM Corretto-11.0.15.9.1 (build 11.0.15+9-LTS, mixed mode)
```

This binary is old, and it's an LTS binary meaning it's likely an even older version. It's very unlikely that the application expects this file, which means the file was likely overwritten during the Linux update command. Or more precisely, the symbol link was changed:

```
$ ls -l /usr/bin/java
lrwxrwxrwx 1 root root 70 Jun 29 20:34 /usr/bin/java ->
```

/etc/alternatives/java

I right away suspected that this system had 2 different versions of Java, one from the base OS and one from another source. This command would confirm it:

```
$ sudo yum list installed |grep java
java-11-openjdk.x86_64                1:11.0.13.0.8-1.amzn2.0.3
@amzn2extra-java-openjdk11
java-11-openjdk-headless.x86_64       1:11.0.13.0.8-1.amzn2.0.3
@amzn2extra-java-openjdk11
java-latest-openjdk.x86_64            1:18.0.1.0.10-2.rolling.el7    @epel
java-latest-openjdk-headless.x86_64   1:18.0.1.0.10-2.rolling.el7
@epel
```

We have 2 installed Java packages, one that says version 11 and the other version 18. This likely corresponds to the 55 and 61 version tags from the error message. So now I know what needs to be fixed. The binary needs to link to the proper version of Java. To find out where the second Java installation lives, I can list the files included in that package and find its folder:

```
sudo repoquery -l java-latest-openjdk
```

Now I can correct the symbolic link:

```
sudo rm -f /usr/bin/java
sudo ln -s /usr/lib/jvm/java-18-openjdk-18.0.1.0.10-
2.rolling.el7.x86_64/bin/java /usr/bin/java
```

And finally, confirm the Java version is the correct one:

```
$ /usr/bin/java --version
openjdk 18.0.1 2022-04-19
OpenJDK Runtime Environment 21.9 (build 18.0.1+10)
OpenJDK 64-Bit Server VM 21.9 (build 18.0.1+10, mixed mode,
```

sharing)

That's all. As you can see, fixing version conflicts can be tricky. This is why using modern technologies like containers and serverless are so useful. But when you need to deal with these issues, it's a good thing to know how to resolve them.