10 Best Practices to Containerize Java Applications with Docker

Last edit: Aug. 24, 2022

snyk

01 Use explicit and deterministic Docker base image tags

- Avoid `FROM openjdk`
- Avoid `FROM maven:openjdk`
- Avoid `FROM maven:3-jdk11`

Instead of generic image aliases, use SHA256 hashes or specific image version tags for deterministic builds.

For example:

- `FROM maven:3.6.3-jdk-11-slim@sha256:68ce1cd457891f`

02 Only install what you need for prod in the Java container image

You do not need a JDK, the Java code, and a build tool like Maven or Gradle in your production image. Instead, use the product of your Java build.

- Copy the Jar or War.
- Use a Java Runtime Environment (JRE).

03 Find and fix security vulnerabilities in your Java Docker image

Docker base images may include security vulnerabilities in the software toolchain they bundle, including the Java Runtime Environment itself.

Scan and fix security vulnerabilities with the free Snyk Container tool which also provides base image recommendations:

```
npm install -g snyk
snyk auth
snyk container test myimage --file=Dockerfile
```

04 Use multi-stage builds to further reduce production image size

Separate your building image from your production image.

- Build your artifacts in the build stage with all possible tools and secrets you need.
- Copy the resulting artifact(s) to the most minimal production image.

05 Don't run Java apps as root

Docker defaults to running the process in the container as the root user, which is a precarious security practice. Use a low privileged user and proper filesystem permissions:

- Create a new user.
- Give the user only necessary permissions.
- Call **USER youruser.**

06 Properly handle events to safely terminate a Java application

Docker creates processes—such as PID 1—and they must inherently handle process signals to function properly. This is why you should avoid any of these variations:

- `CMD "java" "-jar" "application.jar"`
- `CMD "start-app.sh"`

Instead, use a lightweight init system, such as dumb-init, to properly initialize the Java process with signals support:

- `CMD "dumb-init" "java" "-jar" "application.jar"`

07 Gracefully tear down Java applications

Avoid an abrupt termination of a running Java application that halts live connections; either use an application server or create a shutdown hook. Try using a process signal event handler:

```
Runtime.getRuntime().addShutdownHook
(yourShutdownThread);
```

08 Keep unnecessary files out of your Java container images using `.dockerignore`

Use **.dockerignore** to ensure:

- `no debug log files appear in your container.`
- `no secrets are accidentally leaking.`
- `a small Docker base image without redundant and unnecessary files.`

09 Make sure Java is container-aware

Old JVM versions don't respect Docker memory and CPU settings. Make sure the JVM is container-aware.

- `Use Java 10+`
- `Use Java 8 update 191+`

10 Be careful with automatic Docker container generation tools

Tools like Spring Boot Docker image creation and JIB are great tools to automatically build Docker images. However, you are not aware of all security concerns. So, be careful when using these!

When using these tools, properly investigate all aspects of Java container security or create a specific Dockerfile implementing all best practices.

Author

Brian Vermeer
@BrianVerm