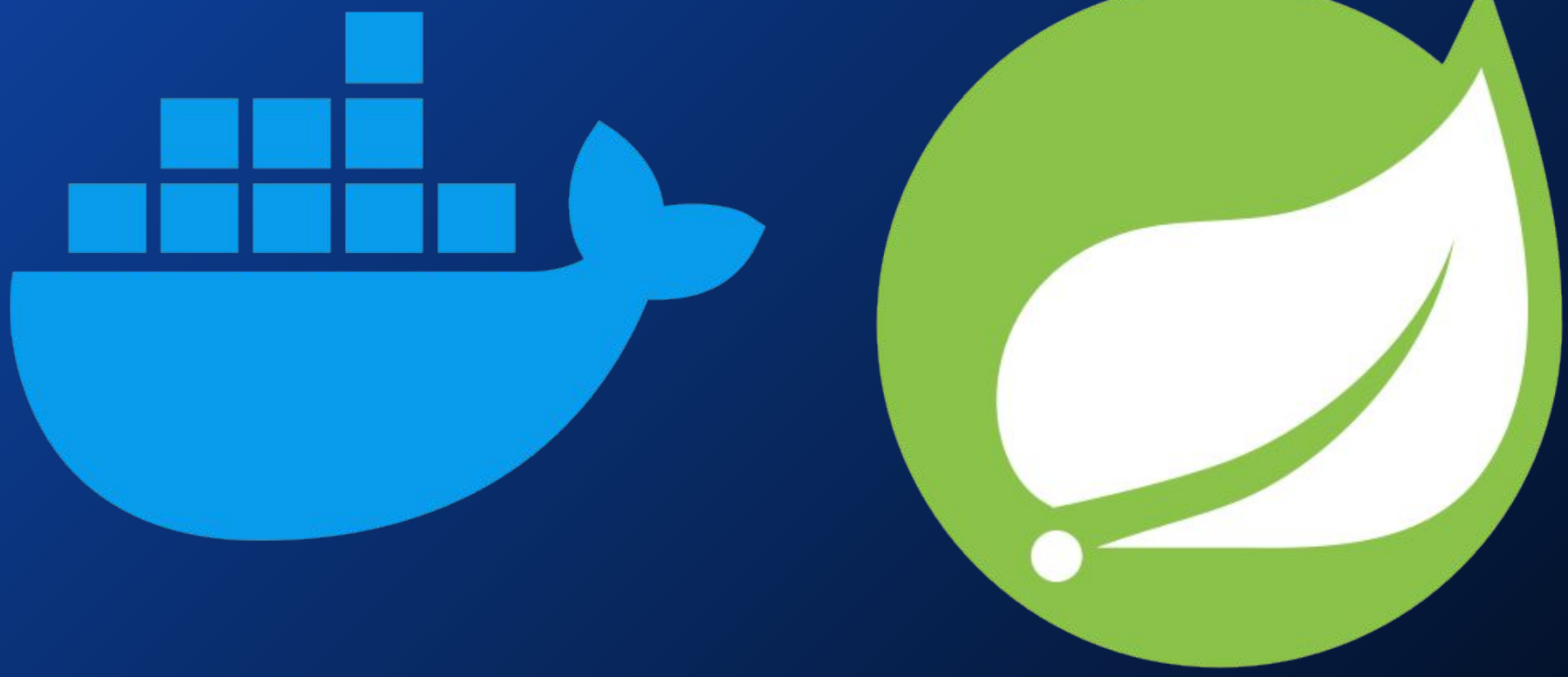
Otimizando o Docker para Spring Boot



linkedin.com/in/pietro-furlanetto-0ba95a34b

github.com/seujorgenochurras

Vamos começar com uma imagem comum

A imagem abaixo é uma imagem bem básica, ela apenas copia e executa um jar.

Dockerfile

```
FROM openjdk:22-jdk
WORKDIR /app
COPY build/libs/*.jar /app/app.jar
ENTRYPOINT ["java", "-jar", "/app/app.jar"]
```

1 - Imagem muito pesada

A imagem está usando a imagem openjdk 22 com jdk, que vem com várias ferramentas que não serão usadas, como javac, jlink e jstat.

Prefira usar imagens openjre e também use as versões alpine ou slim.

Apenas com essa implementação, o tamanho da imagem foi de 500mb para 250mb.

Dockerfile

```
FROM bellsoft/liberica-openjre-alpine:22
WORKDIR /app
COPY build/libs/*.jar /app/app.jar
ENTRYPOINT ["java", "-jar", "/app/app.jar"]
```

2 - Use o cache do Docker

Cada instrução do Dockerfile representa uma layer, as layer são todas cacheadas pelo Docker.

Isso significa que se você copia o jar inteiro, você perde todo esse cache.

Para resolver isso, use a ferramenta Layertools do Spring Boot, ela irá separar os arquivos de dependências dos arquivos da sua aplicação.

Dockerfile

```
FROM bellsoft/liberica-openjre-alpine:22 AS layertools

WORKDIR /extract

ARG JAR_FILE=build/libs/*.jar
COPY ${JAR_FILE} application.jar

RUN java -Djarmode=layertools \
    -jar application.jar extract  \
    --destination /extract/

FROM bellsoft/liberica-openjre-alpine:22 as staged
WORKDIR /app
COPY --from=layertools /extract/dependencies/ ./
COPY --from=layertools /extract/spring-boot-loader/ ./
COPY --from=layertools /extract/snapshot-dependencies/ ./
COPY --from=layertools /extract/application/ ./

ENTRYPOINT ["java",
 "org.springframework.boot.loader.launch.JarLauncher"]
```

3 - Use Java Ahead of Time compilation

O Java AOT não otimiza o Docker em si, o objetivo da compilação Ahead of Time nesse caso é dentre vários, diminuir o tempo de startup da sua aplicação.

Além disso é muito simples implementar isso no seu projeto. Basta adicionar um plugin no seu dependency manager:

build.gradle.kts

```
plugins {
    id("org.graalvm.buildtools.native") version "0.10.5"
}
```

pom.xml

```
<plugin>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-maven-plugin</artifactId>
  <executions>
    <execution>
      <id>process-aot</id>
      <goals>
        <goal>process-aot</goal>
      </goals>
    </execution>
  </executions>
</plugin>;
```

Não se esqueça de mudar o entrypoint para habilitar o AOT

Dockerfile

```
ENTRYPOINT ["java", "-Dspring.aot.enabled=true",
 "org.springframework.boot.loader.launch.JarLauncher"]
```

Bônus

Caso você esteja rodando várias instâncias da JVM em um mesmo contexto (como um micro service), é essencial o uso do CDS (class data sharing), que é uma otimização da JVM 17 que “compartilha” metadados entre classes, reutilizando classes como bibliotecas entre cada instancia.

Se você ainda acha que as builds estão lentas, recomendo procurar uma tecnologia da Google chamada Jib, que é um plugin feito para criar imagens extremamente eficientes, ela remove algumas feature como o start do Deamon do Gradle, tornando o processo de build do Jar ainda mais rápido.