Jdeps, Jlink et layers Docker

Optimisation de l’empreinte d’images Docker Java

Jdeps, Jlink et layers Docker



- Benjamin Barbe
  https://benjamin-barbe.fr/
- Expert Technique Java
  @ SQLI

- shinigota
- ben.barbe10

Généralités et rappels

Généralités et rappels

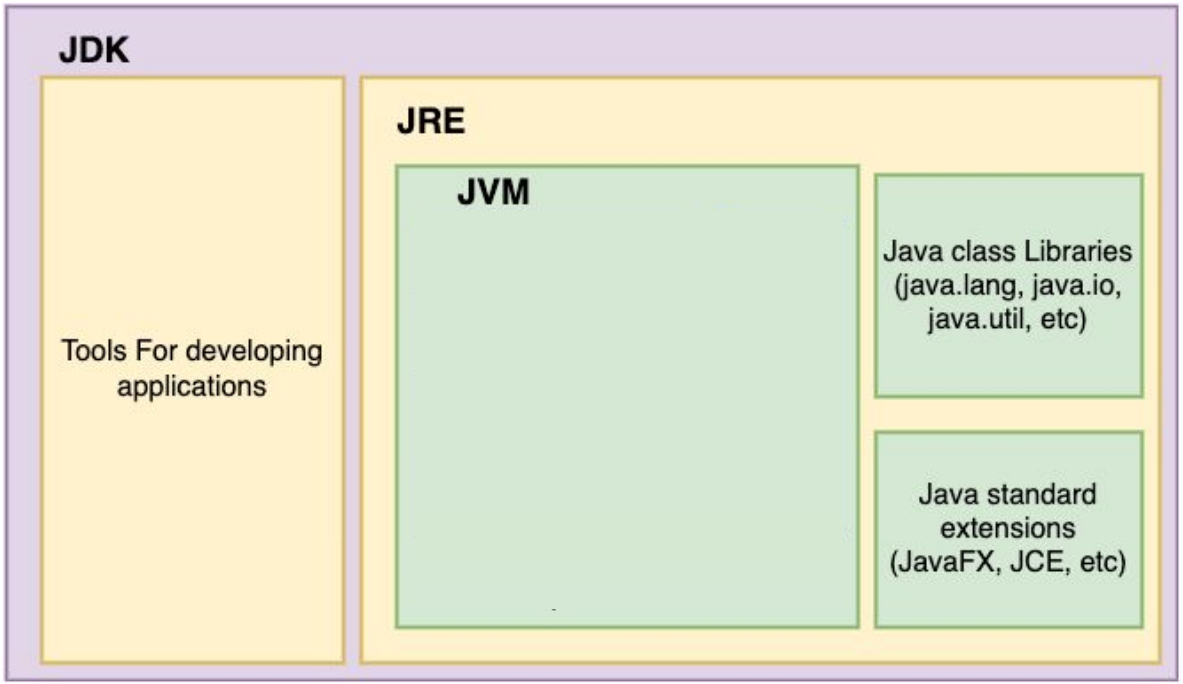
Qui connait Docker ?

JDK vs JRE vs JVM

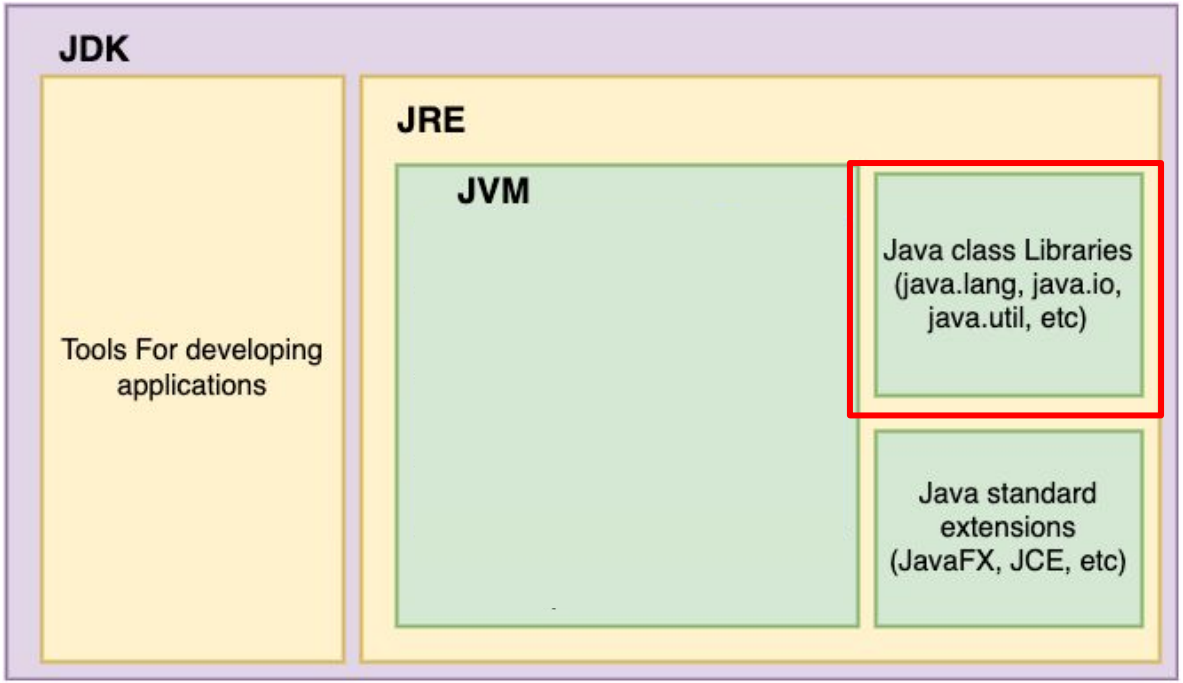
JDK vs JRE vs JVM



JDK vs JRE vs JVM

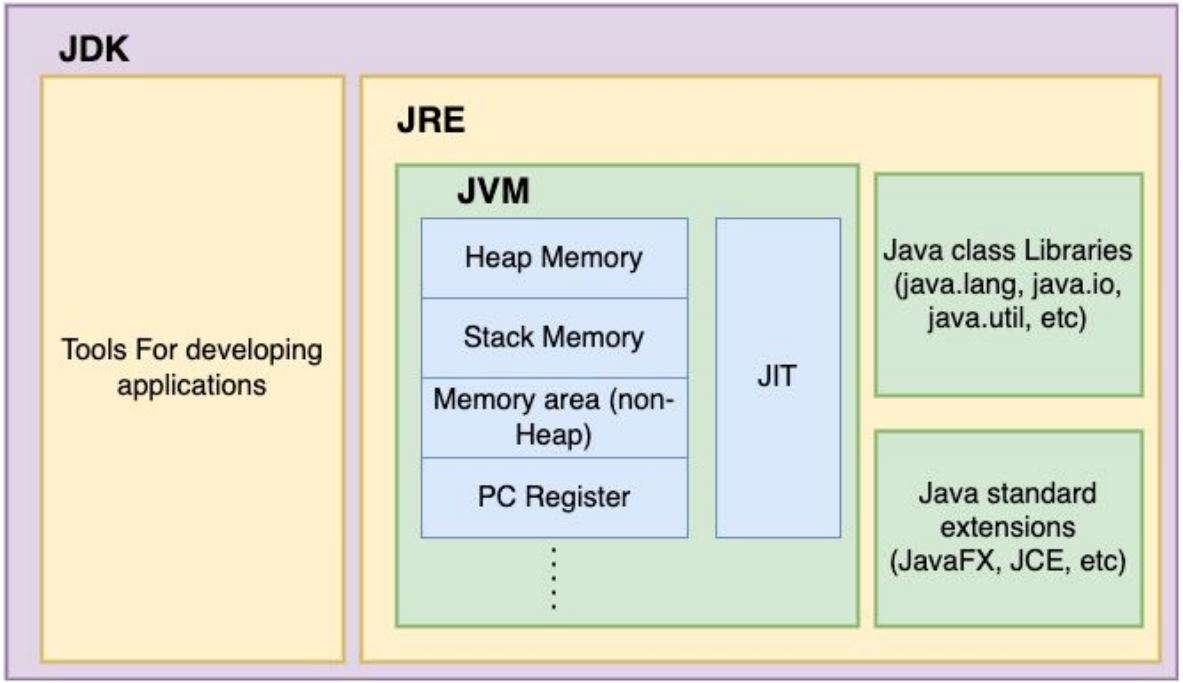


JDK vs JRE vs JVM



Très important pour la suite !

JDK vs JRE vs JVM



Les classes du JRE

Historiquement, les classes étaient packagées dans **lib/rt.jar**.

Les classes du JRE

Historiquement, les classes étaient packagées dans **lib/rt.jar**.

Depuis Java 9 et l'apparition des modules, les classes sont encapsulées dans un fichier **lib/modules**.

Les classes du JRE

Historiquement, les classes étaient packagées dans **lib/rt.jar**.

Depuis Java 9 et l'apparition des modules, les classes sont encapsulées dans un fichier **lib/modules**.

```
$ ls -lh lib/modules
-rw-r--r-- 1 bebarbe bebarbe 124M Jan 18  2023 lib/modules
```

```
$ bin/jimage list lib/modules
jimage: lib/modules

Module: java.base
    META-INF/services/java.nio.file.spi.FileSystemProvider
    com/sun/crypto/provider/AESCipher$AES128_CBC_NoPadding.class
    com/sun/crypto/provider/AESCipher$AES128_CFB_NoPadding.class
    com/sun/crypto/provider/AESCipher$AES128_ECB_NoPadding.class
    com/sun/crypto/provider/AESCipher$AES128_OFB_NoPadding.class
    com/sun/crypto/provider/AESCipher$AES192_CBC_NoPadding.class
    com/sun/crypto/provider/AESCipher$AES192_CFB_NoPadding.class
    com/sun/crypto/provider/AESCipher$AES192_ECB_NoPadding.class
    com/sun/crypto/provider/AESCipher$AES192_OFB_NoPadding.class
    com/sun/crypto/provider/AESCipher$AES256_CBC_NoPadding.class
```

Dockerfile application Java

Dockerfile application Java

```
FROM alpine:3.20.3

RUN apk add --no-cache openjdk17-jre && \
      addgroup -g 1000 -S hello && \
      adduser -u 1000 -G hello -D hello

WORKDIR /app
COPY ./target/*.jar /app/app.jar

RUN chown hello /app/app.jar

USER hello

EXPOSE 8080

CMD java -jar /app/app.jar
```

Dockerfile application Java

```
FROM alpine:3.20.3

RUN apk add --no-cache openjdk17-jre && \
      addgroup -g 1000 -S hello && \
      adduser -u 1000 -G hello -D hello

WORKDIR /app
COPY ./target/*.jar /app/app.jar

RUN chown hello /app/app.jar

USER hello

EXPOSE 8080

CMD java -jar /app/app.jar
```

Image de base Alpine

Dockerfile application Java

```
FROM alpine:3.20.3

RUN apk add --no-cache openjdk17-jre && \
      addgroup -g 1000 -S hello && \
      adduser -u 1000 -G hello -D hello

WORKDIR /app
COPY ./target/*.jar /app/app.jar

RUN chown hello /app/app.jar

USER hello

EXPOSE 8080

CMD java -jar /app/app.jar
```

Installation du JRE, sans cache

Dockerfile application Java

```
FROM alpine:3.20.3

RUN apk add --no-cache openjdk17-jre && \
      addgroup -g 1000 -S hello && \
      adduser -u 1000 -G hello -D hello

WORKDIR /app
COPY ./target/*.jar /app/app.jar

RUN chown hello /app/app.jar

USER hello

EXPOSE 8080

CMD java -jar /app/app.jar
```

Copie du jar

Dockerfile application Java

```
FROM alpine:3.20.3

RUN apk add --no-cache openjdk17-jre && \
      addgroup -g 1000 -S hello && \
      adduser -u 1000 -G hello -D hello

WORKDIR /app
COPY ./target/*.jar /app/app.jar

RUN chown hello /app/app.jar

USER hello

EXPOSE 8080

CMD java -jar /app/app.jar
```

On expose le port

Dockerfile application Java

```
FROM alpine:3.20.3

RUN apk add --no-cache openjdk17-jre && \
      addgroup -g 1000 -S hello && \
      adduser -u 1000 -G hello -D hello

WORKDIR /app
COPY ./target/*.jar /app/app.jar

RUN chown hello /app/app.jar

USER hello

EXPOSE 8080

CMD java -jar /app/app.jar
```

On lance l'application

Dockerfile application Java

```
FROM alpine:3.20.3

RUN apk add --no-cache openjdk17-jre && \
      addgroup -g 1000 -S hello && \
      adduser -u 1000 -G hello -D hello

WORKDIR /app
COPY ./target/*.jar /app/app.jar

RUN chown hello /app/app.jar

USER hello

EXPOSE 8080

CMD java -jar /app/app.jar
```

```
REPOSITORY                                                      TAG                 IMAGE ID       CREATED          SIZE
hello-world                                                     latest              e63f5ccdcbce   21 minutes ago   251MB
```

Dockerfile application Java

```
FROM alpine:3.20.3

RUN apk add --no-cache openjdk17-jre && \
      addgroup -g 1000 -S hello && \
      adduser -u 1000 -G hello -D hello

WORKDIR /app
COPY ./target/*.jar /app/app.jar

RUN chown hello /app/app.jar

USER hello

EXPOSE 8080

CMD java -jar /app/app.jar
```

- L'image devient facilement lourde
- Peu problématique pour une appli front/back
- Très problématique pour une appli modulaire/microservice

```
REPOSITORY                                                      TAG                 IMAGE ID       CREATED          SIZE
hello-world                                                     latest              e63f5ccdcbce   21 minutes ago   251MB
```

Jdeps et Jlink

Jdeps : *présentation*

Jdeps : *présentation*

- Jdeps permet de lister les modules d'un jar.

Jdeps : *présentation*

- Jdeps permet de lister les modules d'un jar.
- Pour lister les modules des dépendances, il faut extraire le jar et passer son dossier lib/ en paramètres

Jdeps : *utilisation*

```
$ jdeps \
--ignore-missing-deps \
--print-module-deps \
-q \
--recursive \
--multi-release 17 \
--class-path="/app/unpacked/BOOT-INF/lib/*" \
--module-path="/app/unpacked/BOOT-INF/lib/*" \
/app/app.jar > /app/dependencies.info
```

Jdeps : *utilisation*

```
$ jdeps \
--ignore-missing-deps \
--print-module-deps \
-q \
--recursive \
--multi-release 17 \
--class-path="/app/unpacked/BOOT-INF/lib/*" \
--module-path="/app/unpacked/BOOT-INF/lib/*" \
/app/app.jar > /app/dependencies.info
```

On donne le jar à jdeps

Jdeps : *utilisation*

```
$ jdeps \
--ignore-missing-deps \
--print-module-deps \
-q \
--recursive \
--multi-release 17 \
--class-path="/app/unpacked/BOOT-INF/lib/*" \
--module-path="/app/unpacked/BOOT-INF/lib/*" \
/app/app.jar > /app/dependencies.info
```

On donne les libs à jdeps

Jdeps : *utilisation*

```
$ jdeps \
--ignore-missing-deps \
--print-module-deps \
-q \
--recursive \
--multi-release 17 \
--class-path="/app/unpacked/BOOT-INF/lib/*" \
--module-path="/app/unpacked/BOOT-INF/lib/*" \
/app/app.jar > /app/dependencies.info
```

On redirige la sortie vers un fichier

Jdeps : *utilisation*

```
$ jdeps \
--ignore-missing-deps \
--print-module-deps \
-q \
--recursive \
--multi-release 17 \
--class-path="/app/unpacked/BOOT-INF/lib/*" \
--module-path="/app/unpacked/BOOT-INF/lib/*" \
/app/app.jar > /app/dependencies.info

$ cat /app/dependencies.info

java.base,java.compiler,java.desktop,java.instrument,java.net.http,java
.prefs,java.rmi,java.scripting,java.security.sasl,java.sql.rowset,jdk.j
fr,jdk.management,jdk.net,jdk.security.jgss,jdk.unsupported
```

Jlink : *présentation*

Jlink : *présentation*

Permets de créer un runtime (JRE) custom

Jlink : *présentation*

Permets de créer un runtime (JRE) custom, à partir de :

- Compression de ressources
- Symboles de debug
- Entrées manuels binaires

Jlink : *présentation*

Permets de créer un runtime (JRE) custom, à partir de :

- Compression de ressources
- Symboles de debug
- Entrées manuels binaires
- **Modules précis**
- …

Jlink : *utilisation*

```
$ jlink --verbose \
--compress 2 \
--strip-java-debug-attributes \
--no-header-files \
--no-man-pages \
--output /jre \
--add-modules $(cat /app/dependencies.info)
```

Jlink : *utilisation*

```
$ jlink --verbose \
--compress 2 \
--strip-java-debug-attributes \
--no-header-files \
--no-man-pages \
--output /jre \                    Répertoire de sortie
--add-modules $(cat /app/dependencies.info)
```

Jlink : *utilisation*

```
$ jlink --verbose \
--compress 2 \
--strip-java-debug-attributes \
--no-header-files \
--no-man-pages \
--output /jre \
--add-modules $(cat /app/dependencies.info)
```

Fichier de modules

Jlink : *utilisation*

```
$ jlink --verbose \
--compress 2 \
--strip-java-debug-attributes \
--no-header-files \
--no-man-pages \
--output /jre \
--add-modules $(cat /app/dependencies.info)
```

```
$ cd jre
$ du -h --max-depth 1
272K    ./legal
128K    ./conf
54M     ./lib
84K     ./bin
55M     .
$ ls -lh lib/modules
-rw-rw-r-- 1 bebarbe bebarbe 24M Aug 20 17:10 lib/modules
```

Jlink : *utilisation*

```
$ jlink --verbose \
--compress 2 \
--strip-java-debug-attributes \
--no-header-files \
--no-man-pages \
--output /jre \
--add-modules $(cat /app/dependencies.info)
```

```
$ cd jre
$ du -h --max-depth 1
272K    ./legal
128K    ./conf
54M     ./lib
84K     ./bin
55M     .
$ ls -lh lib/modules
-rw-rw-r-- 1 bebarbe bebarbe 24M Aug 20 17:10 lib/modules
```

- Le fichier de module est très allégé
- Le JRE lui même est aussi très allégé

Intégration dans un Dockerfile

Intégration dans un Dockerfile

Comment remplacer le JRE standard par notre JRE custom ?

Dockerfile multistage : *Dockerfile java standard*

```
FROM alpine:3.20.3
RUN apk add openjdk17-jre
WORKDIR /opt/app
EXPOSE 8080
COPY ./target/*.jar /opt/app/*.jar
CMD java -jar /app/app.jar
```

Dockerfile multistage : *Stage build maven*

```
FROM alpine:3.20.3 AS builder
RUN apk add openjdk17-jdk maven
WORKDIR /opt/app
COPY … # Copie des sources
RUN mvn clean package

FROM alpine:3.20.3
RUN apk add openjdk17-jre
WORKDIR /opt/app
EXPOSE 8080
COPY --from=builder /opt/app/target/*.jar /opt/app/*.jar
CMD java -jar /app/app.jar
```

Dockerfile multistage : *Stage build maven*

```
FROM alpine:3.20.3 AS builder
RUN apk add openjdk17-jdk maven
WORKDIR /opt/app
COPY … # Copie des sources
RUN mvn clean package

FROM alpine:3.20.3
RUN apk add openjdk17-jre
WORKDIR /opt/app
EXPOSE 8080
COPY --from=builder /opt/app/target/*.jar /opt/app/*.jar
CMD java -jar /app/app.jar
```

Dockerfile multistage : *Stage build maven*

```
FROM alpine:3.20.3 AS builder
RUN apk add openjdk17-jdk maven
WORKDIR /opt/app
COPY … # Copie des sources
RUN mvn clean package

FROM alpine:3.20.3
RUN apk add openjdk17-jre
WORKDIR /opt/app
EXPOSE 8080
COPY --from=builder /opt/app/target/*.jar /opt/app/*.jar
CMD java -jar /app/app.jar
```

Dockerfile multistage : *Stage build maven*

```
FROM alpine:3.20.3 AS builder
RUN apk add openjdk17-jdk maven
WORKDIR /opt/app
COPY … # Copie des sources
RUN mvn clean package

FROM alpine:3.20.3
RUN apk add openjdk17-jre
WORKDIR /opt/app
EXPOSE 8080
COPY --from=builder /opt/app/target/*.jar /opt/app/*.jar
CMD java -jar /app/app.jar
```

Dockerfile multistage : *Stage build maven*

```
FROM alpine:3.20.3 AS builder
RUN apk add openjdk17-jdk maven
WORKDIR /opt/app
COPY … # Copie des sources
RUN mvn clean package

FROM alpine:3.20.3
RUN apk add openjdk17-jre
WORKDIR /opt/app
EXPOSE 8080
COPY --from=builder /opt/app/target/*.jar /opt/app/*.jar
CMD java -jar /app/app.jar
```

Dockerfile multistage : *Stage build maven*

```
FROM alpine:3.20.3 AS builder
RUN apk add openjdk17-jdk maven
WORKDIR /opt/app
COPY … # Copie des sources
RUN mvn clean package

FROM alpine:3.20.3
RUN apk add openjdk17-jre
WORKDIR /opt/app
EXPOSE 8080
COPY --from=builder /opt/app/target/*.jar /opt/app/*.jar
CMD java -jar /app/app.jar
```

Dockerfile multistage : *Stage build maven*

```
FROM alpine:3.20.3 AS builder
RUN apk add openjdk17-jdk maven
WORKDIR /opt/app
COPY … # Copie des sources
RUN mvn clean package

FROM alpine:3.20.3
RUN apk add openjdk17-jre
WORKDIR /opt/app
EXPOSE 8080
COPY --from=builder /opt/app/target/*.jar /opt/app/*.jar
CMD java -jar /app/app.jar
```

Intégration dans un Dockerfile : *stage builder*

```
FROM alpine:3.20.3 as builder

RUN apk add --no-cache openjdk17-jdk

# …
```

Intégration dans un Dockerfile : *stage builder*

```
FROM alpine:3.20.3 as builder                    Ajout d'un label au stage

RUN apk add --no-cache openjdk17-jdk

# …
```

Intégration dans un Dockerfile : *stage builder*

```
FROM alpine:3.20.3 as builder

RUN apk add --no-cache openjdk17-jdk        JDK pour jdeps/jlink

# …
```

Intégration dans un Dockerfile : *ajout de Jdeps*

```
# …
COPY ./target/*.jar /app/app.jar
RUN mkdir /app/unpacked && \
    unzip /app/app.jar -d /app/unpacked && \
    jdeps \
    --ignore-missing-deps \
    -q \
    --recursive \
    --multi-release 17 \
    --print-module-deps \
    --class-path="/app/unpacked/BOOT-INF/lib/*" \
    /app/app.jar > /app/dependencies.info && \
    rm -rf /app/unpacked
# …
```

Intégration dans un Dockerfile : *ajout de Jdeps*

```
# …
COPY ./target/*.jar /app/app.jar
RUN mkdir /app/unpacked && \
    unzip /app/app.jar -d /app/unpacked && \
    jdeps \
    --ignore-missing-deps \
    -q \
    --recursive \
    --multi-release 17 \
    --print-module-deps \
    --class-path="/app/unpacked/BOOT-INF/lib/*" \
    /app/app.jar > /app/dependencies.info && \
    rm -rf /app/unpacked
# …
```

Copie du jar

Intégration dans un Dockerfile : *ajout de Jdeps*

```
# …
COPY ./target/*.jar /app/app.jar
RUN mkdir /app/unpacked && \
    unzip /app/app.jar -d /app/unpacked && \ Dézippage du jar
    jdeps \
    --ignore-missing-deps \
    -q \
    --recursive \
    --multi-release 17 \
    --print-module-deps \
    --class-path="/app/unpacked/BOOT-INF/lib/*" \
    /app/app.jar > /app/dependencies.info && \
    rm -rf /app/unpacked
# …
```

Intégration dans un Dockerfile : *ajout de Jdeps*

```
# …
COPY ./target/*.jar /app/app.jar
RUN mkdir /app/unpacked && \
    unzip /app/app.jar -d /app/unpacked && \
    jdeps \
    --ignore-missing-deps \
    -q \
    --recursive \
    --multi-release 17 \
    --print-module-deps \
    --class-path="/app/unpacked/BOOT-INF/lib/*" \
    /app/app.jar > /app/dependencies.info && \
    rm -rf /app/unpacked
# …
```

Listing des modules

Intégration dans un Dockerfile : *ajout de Jdeps*

```
# …
COPY ./target/*.jar /app/app.jar
RUN mkdir /app/unpacked && \
    unzip /app/app.jar -d /app/unpacked && \
    jdeps \
    --ignore-missing-deps \
    -q \
    --recursive \
    --multi-release 17 \
    --print-module-deps \
    --class-path="/app/unpacked/BOOT-INF/lib/*" \
    /app/app.jar > /app/dependencies.info && \
    rm -rf /app/unpacked                          Suppression du jar dezippé
# …
```

Intégration dans un Dockerfile : *ajout de Jlink*

```
# …
RUN jlink --verbose \
    --compress 2 \
    --strip-java-debug-attributes \
    --no-header-files \
    --no-man-pages \
    --output /jre \
    --add-modules $(cat /app/dependencies.info)
# …
```

Intégration dans un Dockerfile : *ajout de Jlink*

```
# …
RUN jlink --verbose \
    --compress 2 \
    --strip-java-debug-attributes \
    --no-header-files \
    --no-man-pages \
    --output /jre \                              Sortie dans /jre
    --add-modules $(cat /app/dependencies.info)
# …
```

Intégration dans un Dockerfile : *ajout de Jlink*

```
# …
RUN jlink --verbose \
    --compress 2 \
    --strip-java-debug-attributes \
    --no-header-files \
    --no-man-pages \
    --output /jre \
    --add-modules $(cat /app/dependencies.info)  Utilisation du ficher jdeps
# …
```

Intégration dans un Dockerfile : *stage applicatif*

```
# …
FROM alpine:3.20.3

ENV JAVA_HOME=/jre
ENV PATH="${JAVA_HOME}/bin:${PATH}"

COPY ./target/*.jar /app/app.jar

RUN addgroup -g 1000 -S hello && \
    adduser -u 1000 -G hello -D hello && \
    chown -R hello /app

COPY --from=builder /jre $JAVA_HOME

USER hello

WORKDIR /app

EXPOSE 8080

CMD java -jar /app/app.jar
```

Intégration dans un Dockerfile : *stage applicatif*

```
# …
FROM alpine:3.20.3

ENV JAVA_HOME=/jre
ENV PATH="${JAVA_HOME}/bin:${PATH}"

COPY ./target/*.jar /app/app.jar

RUN addgroup -g 1000 -S hello && \
    adduser -u 1000 -G hello -D hello && \
    chown -R hello /app

COPY --from=builder /jre $JAVA_HOME        Copie du JRE du stage builder

USER hello

WORKDIR /app

EXPOSE 8080

CMD java -jar /app/app.jar
```

Intégration dans un Dockerfile : *résultat*

```
REPOSITORY                    TAG                 IMAGE ID            CREATED             SIZE
hello-world                   latest              505e27accaa5        2 seconds ago       87.5MB
```

Intégration dans un Dockerfile : *résultat*

```
REPOSITORY                    TAG                 IMAGE ID            CREATED             SIZE
hello-world                   latest              505e27accaa5        2 seconds ago       87.5MB
```



Intégration dans un Dockerfile : *résultat*

```
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
hello-world         latest              505e27accaa5        2 seconds ago       87.5MB
```



Intégration dans un Dockerfile : *résultat*

```
REPOSITORY                          TAG                 IMAGE ID            CREATED             SIZE
hello-world                         latest              505e27accaa5        2 seconds ago       87.5MB
```



Intégration dans un Dockerfile : *résultat*

```
REPOSITORY                    TAG                 IMAGE ID            CREATED             SIZE
hello-world                   latest              505e27accaa5        2 seconds ago       87.5MB
```



Intégration dans un Dockerfile : *résultat*

```
REPOSITORY                          TAG                 IMAGE ID            CREATED             SIZE
hello-world                         latest              505e27accaa5        2 seconds ago       87.5MB
```



Layers

Layers docker : *définition*

Layers docker : *définition*

- Un layer est une instruction, dans un Dockerfile, modifiant le contenu de l'image

Layers docker : *définition*

- Un layer est une instruction, dans un Dockerfile, modifiant le contenu de l'image
- Un layer est mis en cache, en local et sur le registry. Si une image utilise un même layer, il ne sera pas rebuildé

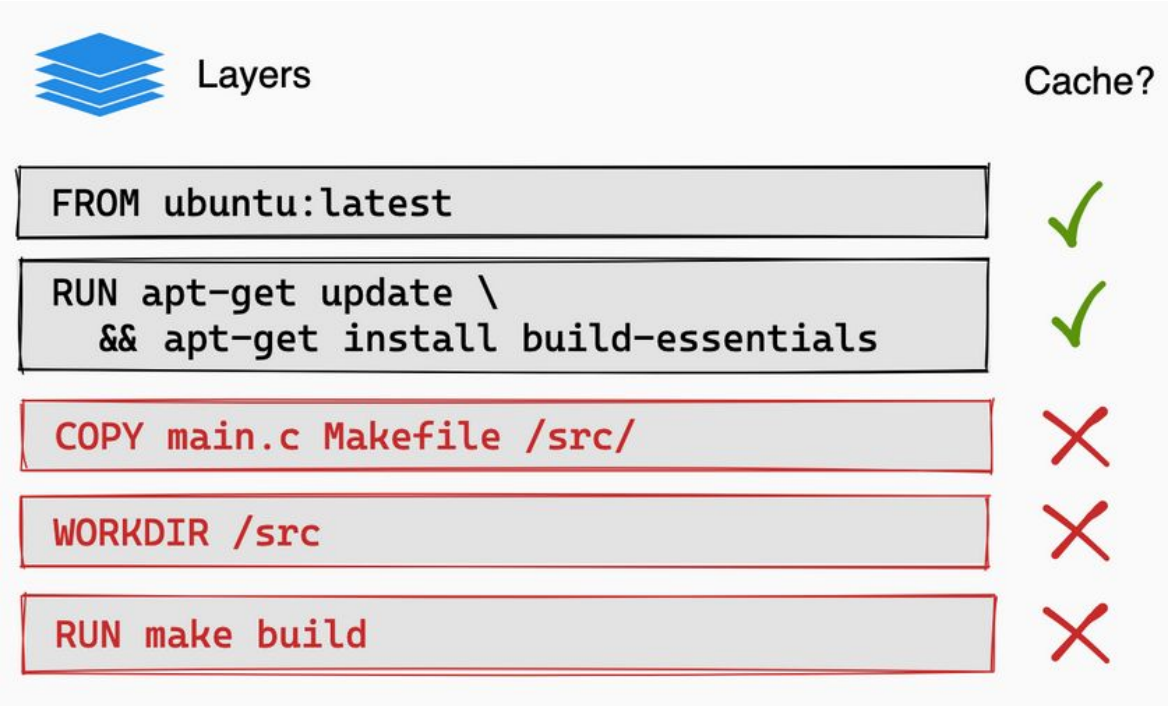
Layers docker : *définition*

- Un layer est une instruction, dans un Dockerfile, modifiant le contenu de l'image
- Un layer est mis en cache, en local et sur le registry. Si une image utilise un même layer, il ne sera pas rebuildé
- Si un layer est invalidé, il est rebuildé

Layers docker : *définition*

- Un layer est une instruction, dans un Dockerfile, modifiant le contenu de l'image
- Un layer est mis en cache, en local et sur le registry. Si une image utilise un même layer, il ne sera pas rebuildé
- Si un layer est invalidé, il est rebuildé
- Si un layer est invalidé, les layers suivants le sont aussi

Layers docker : *définition*



Layers docker : *définition*

FROM … as A	✓
RUN …	✓
COPY …	✗
RUN …	✗

FROM …	✓
RUN …	✓
COPY …	✓
COPY –from=A …	✗
ENTRYPOINT	✗

Layers docker : *définition*

FROM … as A	✓
RUN …	✓
COPY …	✗
RUN …	✗



FROM …	✓
RUN …	✓
COPY …	✓
COPY –from=A …	✗
ENTRYPOINT	✗



Layers docker : *définition*

FROM … as A	✓
RUN …	✓
COPY …	✗
RUN …	✗



FROM …	✓
RUN …	✓
COPY …	✓
COPY –from=A …	✗
ENTRYPOINT	✗



Layers docker : *définition*

FROM … as A	✓
RUN …	✓
COPY …	✗
RUN …	✗



FROM …	✓
RUN …	✓
COPY …	✓
COPY –from=A …	✗
ENTRYPOINT	✗



Layers docker : *définition*

FROM … as A	✓
RUN …	✓
COPY …	✗
RUN …	✗



FROM …	✓
RUN …	✓
COPY …	✓
COPY –from=A …	✗
ENTRYPOINT	✗



Layers docker : *définition*

FROM … as A	✓
RUN …	✓
COPY …	✗
RUN …	✗

FROM …	✓
RUN …	✓
COPY …	✓
COPY –from=A …	✗
ENTRYPOINT	✗





Layers docker : *définition*

FROM … as A	✓
RUN …	✓
COPY …	✗
RUN …	✗



FROM …	✓
RUN …	✓
COPY …	✓
COPY –from=A …	✗
ENTRYPOINT	✗



Layers docker : *définition*

- Une image est un ensemble de layers.
- Dans un registry, en réalité, on ne pousse pas une image docker entière, mais seulement ses layers.
- Des images peuvent partager des mêmes layers.

Layers Spring

Layers Spring : *définition*

Spring Boot propose ses layers pour être compatible avec Docker.

Toutes les classes et dépendances sont réparties dans plusieurs layers / dossiers.

Layers Spring : *définition*

Spring Boot propose ses layers pour être compatible avec Docker.

Toutes les classes et dépendances sont réparties dans plusieurs layers / dossiers.

```
$ java -Djarmode=layertools -jar app.jar extract
$ ls

dependencies/
spring-boot-loader/
snapshot-dependencies/
application/
```

Layers Spring : *intégration*

```
### builder stage ###
FROM alpine:3.20.3 AS builder
#...
RUN cd /app && \
      java -Djarmode=layertools -jar app.jar extract
### app stage ###
FROM alpine:3.20.3
#...
COPY --from=builder /jre $JAVA_HOME
COPY --from=builder /app/dependencies/ /app/
COPY --from=builder /app/snapshot-dependencies/ /app/
COPY --from=builder /app/spring-boot-loader/ /app/
COPY --from=builder /app/application/ /app/

USER hello

WORKDIR /app

EXPOSE 8080

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

Layers Spring : *intégration*

Stage builder, jdeps, jlink, …

```
### builder stage ###
FROM alpine:3.20.3 AS builder
#...
RUN cd /app && \
      java -Djarmode=layertools -jar app.jar extract
### app stage ###
FROM alpine:3.20.3
#...
COPY --from=builder /jre $JAVA_HOME
COPY --from=builder /app/dependencies/ /app/
COPY --from=builder /app/snapshot-dependencies/ /app/
COPY --from=builder /app/spring-boot-loader/ /app/
COPY --from=builder /app/application/ /app/

USER hello

WORKDIR /app

EXPOSE 8080

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

Layers Spring : *intégration*

```
### builder stage ###
FROM alpine:3.20.3 AS builder
#...
RUN cd /app && \
      java -Djarmode=layertools -jar app.jar extract
### app stage ###
FROM alpine:3.20.3
#...
COPY --from=builder /jre $JAVA_HOME
COPY --from=builder /app/dependencies/ /app/
COPY --from=builder /app/snapshot-dependencies/ /app/
COPY --from=builder /app/spring-boot-loader/ /app/
COPY --from=builder /app/application/ /app/

USER hello

WORKDIR /app

EXPOSE 8080

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

Layers Spring : *intégration*

```
### builder stage ###
FROM alpine:3.20.3 AS builder
#...
RUN cd /app && \
      java -Djarmode=layertools -jar app.jar extract
### app stage ###
FROM alpine:3.20.3
#...
COPY --from=builder /jre $JAVA_HOME
COPY --from=builder /app/dependencies/ /app/
COPY --from=builder /app/snapshot-dependencies/ /app/
COPY --from=builder /app/spring-boot-loader/ /app/
COPY --from=builder /app/application/ /app/

USER hello

WORKDIR /app

EXPOSE 8080

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

Layers Spring : *intégration*

```
### builder stage ###
FROM alpine:3.20.3 AS builder
#...
RUN cd /app && \
      java -Djarmode=layertools -jar app.jar extract
### app stage ###
FROM alpine:3.20.3
#...
COPY --from=builder /jre $JAVA_HOME
COPY --from=builder /app/dependencies/ /app/
COPY --from=builder /app/snapshot-dependencies/ /app/
COPY --from=builder /app/spring-boot-loader/ /app/
COPY --from=builder /app/application/ /app/

USER hello

WORKDIR /app

EXPOSE 8080

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

jar remplacé par layers Spring

Layers Spring : *intégration*

```
### builder stage ###
FROM alpine:3.20.3 AS builder
#...
RUN cd /app && \
      java -Djarmode=layertools -jar app.jar extract
### app stage ###
FROM alpine:3.20.3
#...
COPY --from=builder /jre $JAVA_HOME
COPY --from=builder /app/dependencies/ /app/
COPY --from=builder /app/snapshot-dependencies/ /app/
COPY --from=builder /app/spring-boot-loader/ /app/
COPY --from=builder /app/application/ /app/

USER hello

WORKDIR /app

EXPOSE 8080

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

launcher Spring

Layers Spring : *résultat*

COPY --from=layers-builder /app/dependencies/ ./	✓
COPY --from=layers-builder /app/snapshot-dependencies/ ./	✓
COPY --from=layers-builder /app/spring-boot-loader/ ./	✓
COPY --from=layers-builder /app/application/ ./	✗

Layers Spring : *résultat*

COPY --from=layers-builder /app/dependencies/ ./	✓
COPY --from=layers-builder /app/snapshot-dependencies/ ./	✓
COPY --from=layers-builder /app/spring-boot-loader/ ./	✓
COPY --from=layers-builder /app/application/ ./	✗

Les dépendances ne changent pas, le code de l'application change.

Seulement le layer applicatif est rebuildé.

Layers Spring : *résultat*

Lors du build, les layers de dépendances sont bien cachées. Uniquement le layer applicatif et les suivants sont rebuildés.

```
$ docker build -t hello-world -f Dockerfile-jlink-jdeps-layers .

[...]
=> CACHED [stage-1 3/8] COPY --from=builder /jre /jre
=> CACHED [stage-1 4/8] COPY --from=builder /app/dependencies/ /app/
=> CACHED [stage-1 5/8] COPY --from=builder /app/snapshot-dependencies/ /app/
=> CACHED [stage-1 6/8] COPY --from=builder /app/spring-boot-loader/ /app/
=> [stage-1 7/8] COPY --from=builder /app/application/ /app/
[...]
```

Layers Spring : *résultat*

Lors du build, les layers de dépendances sont bien cachées. Uniquement le layer applicatif et les suivants sont rebuildés.

```
$ docker build -t hello-world -f Dockerfile-jlink-jdeps-layers .

[...]
=> CACHED [stage-1 3/8] COPY --from=builder /jre /jre
=> CACHED [stage-1 4/8] COPY --from=builder /app/dependencies/ /app/
=> CACHED [stage-1 5/8] COPY --from=builder /app/snapshot-dependencies/ /app/
=> CACHED [stage-1 6/8] COPY --from=builder /app/spring-boot-loader/ /app/
=> [stage-1 7/8] COPY --from=builder /app/application/ /app/
[...]
```

Layers Spring : *résultat*

Lors du build, les layers de dépendances sont bien cachées. Uniquement le layer applicatif et les suivants sont rebuildés.

```
$ docker build -t hello-world -f Dockerfile-jlink-jdeps-layers .

[...]
=> CACHED [stage-1 3/8] COPY --from=builder /jre /jre
=> CACHED [stage-1 4/8] COPY --from=builder /app/dependencies/ /app/
=> CACHED [stage-1 5/8] COPY --from=builder /app/snapshot-dependencies/ /app/
=> CACHED [stage-1 6/8] COPY --from=builder /app/spring-boot-loader/ /app/
=> [stage-1 7/8] COPY --from=builder /app/application/ /app/
[...]
```

Layers Spring : *résultat*

```
$ docker history b91f8acb7300
```

Layers Spring : *résultat*

```
$ docker history b91f8acb7300
```

```
MAGE          CREATED          CREATED BY                                      SIZE
91f8acb7300   2 minutes ago    ENTRYPOINT ["java" "org.springframework.boot…   0B
missing>      2 minutes ago    EXPOSE map[8080/tcp:{}]                         0B
missing>      2 minutes ago    WORKDIR /app                                    0B
missing>      2 minutes ago    USER hello                                      0B
missing>      2 minutes ago    COPY /app/application/ /app/ # buildkit         19.7kB
missing>      12 minutes ago   COPY /app/spring-boot-loader/ /app/ # buildk…   389kB
missing>      12 minutes ago   COPY /app/snapshot-dependencies/ /app/ # bui…   0B
missing>      12 minutes ago   COPY /app/dependencies/ /app/ # buildkit        25.6MB
missing>      12 minutes ago   COPY /jre /jre # buildkit                       53.7MB
```

Layers Spring : *résultat*

```
$ docker history b91f8acb7300
```

```
MAGE          CREATED          CREATED BY                                      SIZE
91f8acb7300   2 minutes ago    ENTRYPOINT ["java" "org.springframework.boot…   0B
missing>      2 minutes ago    EXPOSE map[8080/tcp:{}]                         0B
missing>      2 minutes ago    WORKDIR /app                                    0B
missing>      2 minutes ago    USER hello                                      0B
missing>      2 minutes ago    COPY /app/application/ /app/ # buildkit         19.7kB
missing>      12 minutes ago   COPY /app/spring-boot-loader/ /app/ # buildk…   389kB
missing>      12 minutes ago   COPY /app/snapshot-dependencies/ /app/ # bui…   0B
missing>      12 minutes ago   COPY /app/dependencies/ /app/ # buildkit        25.6MB
missing>      12 minutes ago   COPY /jre /jre # buildkit                       53.7MB
```

Plus de 75mo de layers déjà buildés ont été réutilisés

Layers Spring : *résultat*

```
$ docker history b91f8acb7300
```

```
MAGE          CREATED          CREATED BY                                      SIZE
91f8acb7300   2 minutes ago    ENTRYPOINT ["java" "org.springframework.boot…   0B
missing>      2 minutes ago    EXPOSE map[8080/tcp:{}]                         0B
missing>      2 minutes ago    WORKDIR /app                                    0B
missing>      2 minutes ago    USER hello                                      0B
missing>      2 minutes ago    COPY /app/application/ /app/ # buildkit         19.7kB
missing>      12 minutes ago   COPY /app/spring-boot-loader/ /app/ # buildk…   389kB
missing>      12 minutes ago   COPY /app/snapshot-dependencies/ /app/ # bui…   0B
missing>      12 minutes ago   COPY /app/dependencies/ /app/ # buildkit        25.6MB
missing>      12 minutes ago   COPY /jre /jre # buildkit                       53.7MB
```

Uniquement un layer de 20ko a été rebuildé

Layers Spring : *résultat*

Lors du push, uniquement le layer contenant les sources est envoyé sur le registry

```
5f70bf18a086: Layer already exists
8c1c2ca9e5bd: Pushed
acac22187ad8: Layer already exists
f0cc278c04f2: Layer already exists
1dae0366c68d: Layer already exists
c68145c5d6bd: Layer already exists
06f3a5fefc89: Layer already exists
78561cef0761: Layer already exists
```

Layers Spring : *résultat*

Lors du push, uniquement le layer contenant les sources est envoyé sur le registry

19.7kB

```
5f70bf18a086: Layer already exists
8c1c2ca9e5bd: Pushed
acac22187ad8: Layer already exists
f0cc278c04f2: Layer already exists
1dae0366c68d: Layer already exists
c68145c5d6bd: Layer already exists
06f3a5fefc89: Layer already exists
78561cef0761: Layer already exists
```

Layers Spring : *résultat*

Lors du push, uniquement le layer contenant les sources est envoyé sur le registry

19.7kB

```
5f70bf18a086: Layer already exists
8c1c2ca9e5bd: Pushed
acac22187ad8: Layer already exists
f0cc278c04f2: Layer already exists
0366c68d: Layer already exists
145c5d6bd: Layer already exists
3a5fefc89: Layer already exists
61cef0761: Layer already exists
```



Layers Spring : *résultat*

- L'empreinte (le poids total) de l'image reste la même
- Mais derrière, des layers sont réutilisés

```
REPOSITORY                                  TAG                 IMAGE ID       CREATED          SIZE
hello-world                                 latest              505e27accaa5   2 seconds ago    87.5MB
```

Garbage Collector

Garbage Collector : *impact de l'utilisation des layers*

Garbage Collector : *impact de l'utilisation des layers*

Sur la machine de build, Docker utilise du stockage pour son cache de build :

- Les layers y sont stockés (qu'ils soient sur des stages intermédiaires ou finaux)
- Le build est KO si le cache est plein

Garbage Collector : *impact de l'utilisation des layers*

Sur la machine de build, Docker utilise du stockage pour son cache de build :

- Les layers y sont stockés (qu'ils soient sur des stages intermédiaires ou finaux)
- Le build est KO si le cache est plein

Solution : le Garbage Collector.

Garbage Collector : *mise en place*

`/etc/docker/daemon.json`

```
{
  ...
  "builder":
    {
      "gc":
        {
          "enabled": true,
          "defaultKeepStorage": "20GB",
          "policy":
            [
              { "keepStorage": "20GB", "filter": ["unused-for=168h"] },
              { "keepStorage": "30GB", "filter": ["unused-for=72h"] },
              { "keepStorage": "40GB", "filter": ["unused-for=24h"] },
              { "keepStorage": "50GB", "filter": ["unused-for=12h"] },
              { "keepStorage": "60GB", "all": true },
            ],
        },
    },
}
```

Garbage Collector : *mise en place*

`/etc/docker/daemon.json`

```
{
  ...
  "builder":
    {
      "gc":
        {
          "enabled": true,
          "defaultKeepStorage": "20GB",
          "policy":
            [
              { "keepStorage": "20GB", "filter": ["unused-for=168h"] },
              { "keepStorage": "30GB", "filter": ["unused-for=72h"] },
              { "keepStorage": "40GB", "filter": ["unused-for=24h"] },
              { "keepStorage": "50GB", "filter": ["unused-for=12h"] },
              { "keepStorage": "60GB", "all": true },
            ],
        },
    },
}
```

Seuils de purges, basés sur seuil en GB et durée d'inutilisation

Réflexion

Réflexion

Si le JRE custom créé par Jdeps et Jlink ne contient que les modules utilisés par l'application, quid de ceux appelés par réflexion ?

Réflexion : *test blanc*

```
HttpClient client = HttpClient.newHttpClient();

HttpRequest request = HttpRequest.newBuilder()
        .uri(new URI("https://postman-echo.com/get"))
        .GET()
        .build();

HttpResponse<String> response = client.send(request,
HttpResponse.BodyHandlers.ofString());

System.out.println("response = " + response.body());
```

Import explicite `import java.net.http.HttpClient;`

Réflexion : *test blanc*

jdeps **détecte le module**

```
java.base,java.net.http
```

Réflexion : *test blanc*

jdeps **détecte le module** et lors du lancement le **JRE jlink fonctionne**

```
java.base,java.net.http
```

```
response = {
  "args": {},
  "headers": {
    "host": "postman-echo.com",
    "x-request-start": "t1733479815.018",
    "connection": "close",
    "x-forwarded-proto": "https",
    "x-forwarded-port": "443",
    "x-amzn-trace-id": "Root=1-6752cd87-12aedaff052000e13ab6c01b",
    "user-agent": "Java-http-client/17.0.6"
  },
  "url": "https://postman-echo.com/get"
}
```

Réflexion : *version réflexion*

```
Object httpClient = Class.forName("java.net.http.HttpClient")
        .getDeclaredMethod("newHttpClient")
        .invoke(null);

Class<?> uriClazz = Class.forName("java.net.URI");
Constructor<?> uriConstructor = uriClazz.getConstructor(String.class);
Object uri = uriConstructor.newInstance("https://postman-echo.com/get");

Object request = Class.forName("java.net.http.HttpRequest")
        .getDeclaredMethod("newBuilder")
        .invoke(null);

request = request.getClass()
        .getDeclaredMethod("uri", Class.forName("java.net.URI"))
        .invoke(request, uri);

// ...
```

Pas d'import explicite de java.net.http

Réflexion : *version réflexion*

jdeps **ne détecte pas le module**

```
java.base
```

Réflexion : *version réflexion*

jdeps **ne détecte pas le module** et le **JRE jlink ne fonctionne pas**

```
java.base
```

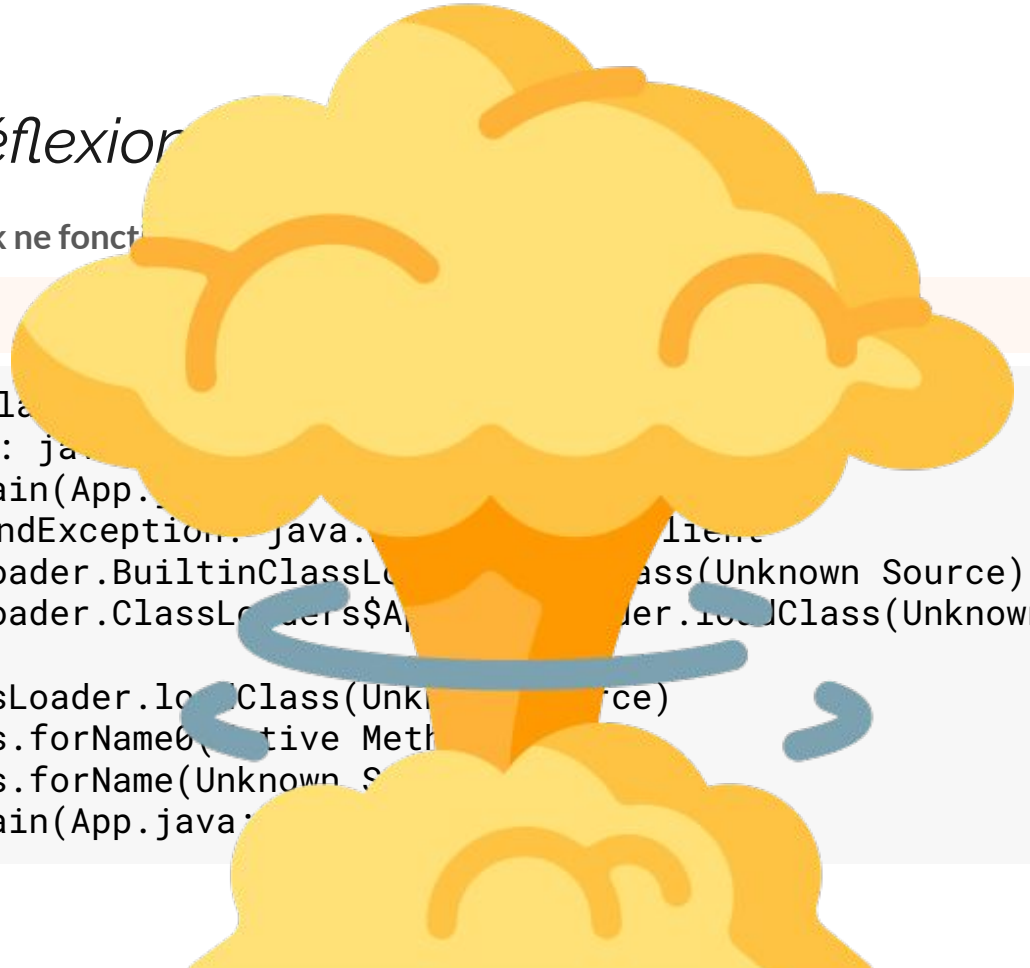
```
Exception in thread "main" java.lang.RuntimeException:
java.lang.ClassNotFoundException: java.net.http.HttpClient
    at com.benjamin_barbe.App.main(App.java:50)
Caused by: java.lang.ClassNotFoundException: java.net.http.HttpClient
    at java.base/jdk.internal.loader.BuiltinClassLoader.loadClass(Unknown Source)
    at java.base/jdk.internal.loader.ClassLoaders$AppClassLoader.loadClass(Unknown
Source)
    at java.base/java.lang.ClassLoader.loadClass(Unknown Source)
    at java.base/java.lang.Class.forName0(Native Method)
    at java.base/java.lang.Class.forName(Unknown Source)
    at com.benjamin_barbe.App.main(App.java:23)
```

Réflexion : *version réflexion*

jdeps **ne détecte pas le module** et le **JRE jlink ne fonct**

```
java.base
```

```
Exception in thread "main" java.la
java.lang.ClassNotFoundException: ja
    at com.benjamin_barbe.App.main(App.
Caused by: java.lang.ClassNotFoundExceptio
    at java.base/jdk.internal.loader.BuiltinClassLo
    at java.base/jdk.internal.loader.ClassLo
Source)
    at java.base/java.lang.ClassLoader.lo
    at java.base/java.lang.Class.forName0(
    at java.base/java.lang.Class.forName(Unknown S
    at com.benjamin_barbe.App.main(App.java:
```



Conclusion

Conclusion

Pour optimiser nos images docker, plusieurs axes :

Conclusion

Pour optimiser nos images docker, plusieurs axes :

- **Jdeps et Jlink** pour utiliser un JRE custom : **réduire la taille des images**
- **Utiliser les layers : privilégier la réutilisation**

Conclusion

Pour optimiser nos images docker, plusieurs axes :

- **Jdeps et Jlink** pour utiliser un JRE custom : **réduire la taille des images**
- **Utiliser les layers : privilégier la réutilisation**

Impacts :

Conclusion

Pour optimiser nos images docker, plusieurs axes :

- **Jdeps et Jlink** pour utiliser un JRE custom : **réduire la taille des images**
- **Utiliser les layers** : **privilégier la réutilisation**

Impacts :

- Le stockage utilisé sur la machine de build sera considérablement augmenté
- Il faut utiliser le **garbage collector** !
- Ne fonctionne pas avec de la **réflexivité** !

Questions ?



Merci ! 😁

Google Slides

GitHub

Sondage