Jdeps, Jlink et Layers Docker

Comment faire des images Docker légères !

A PROPOS …



- 🌐 Benjamin Barbe
  https://bnjmn.fr/
- 💼 Expert Technique Java
  @ sqli

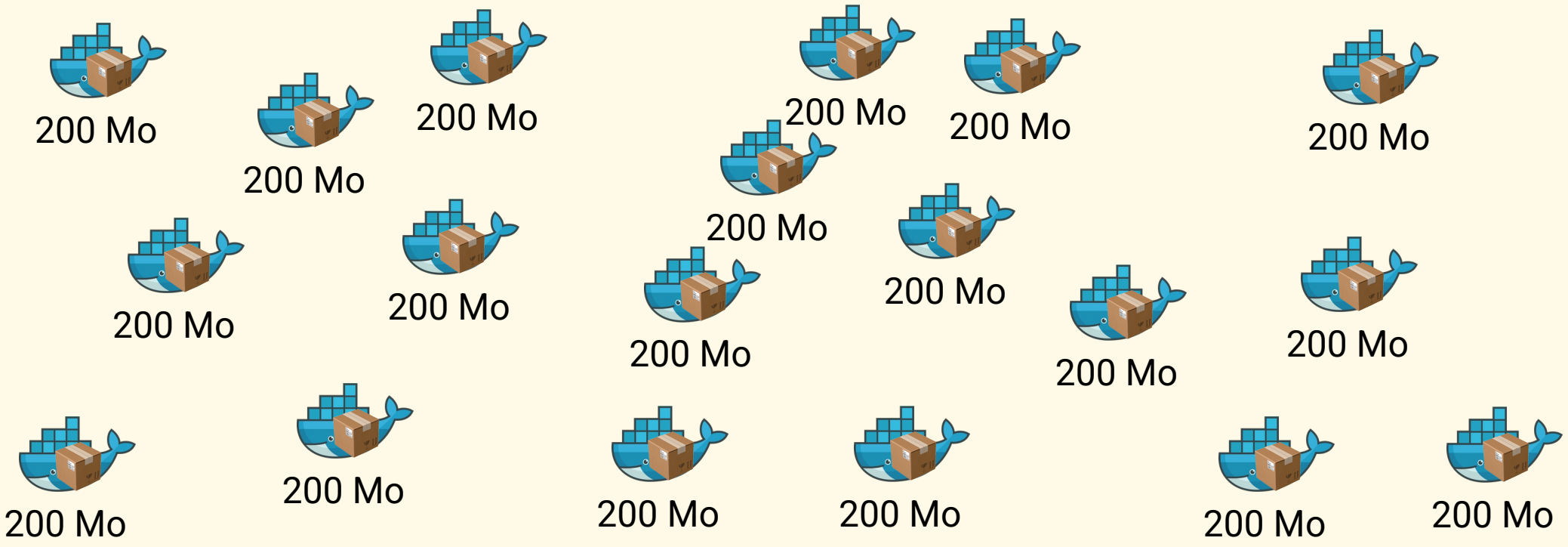
- shinigota
- ben.barbe10

Un peu de contexte …

UN PEU DE CONTEXTE …

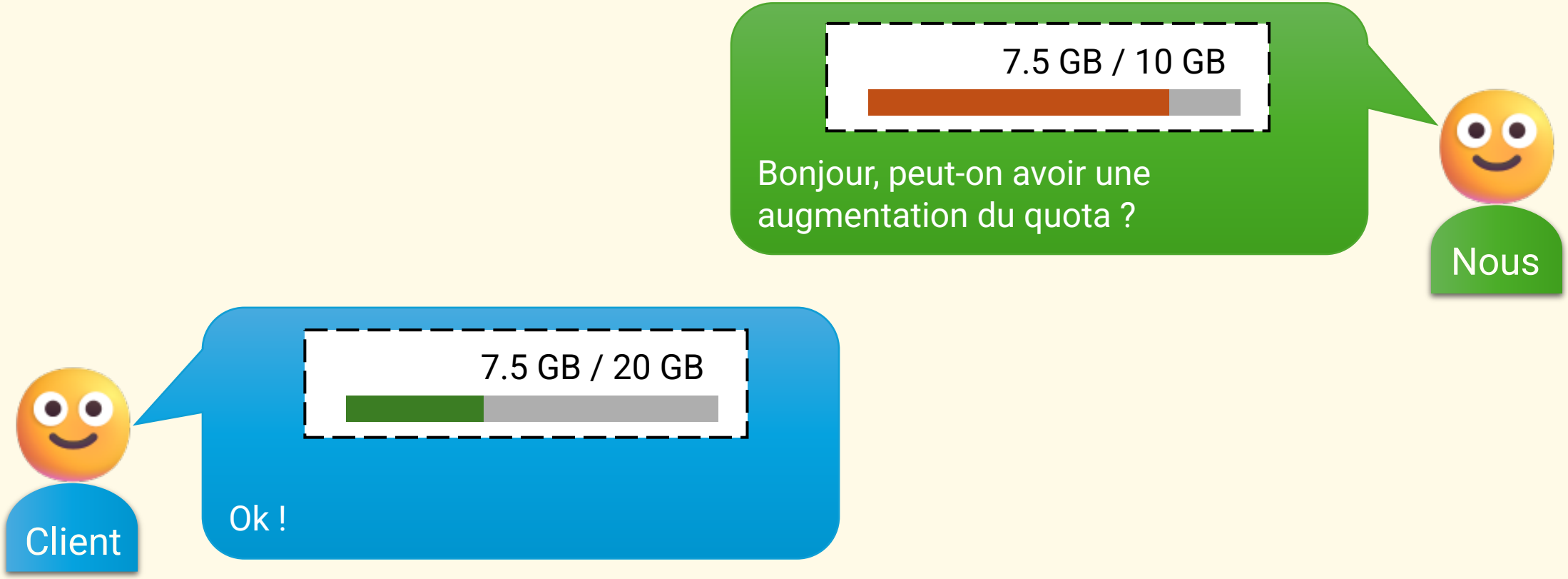
Une application **Dockerisée** Java …

En **microservices**.



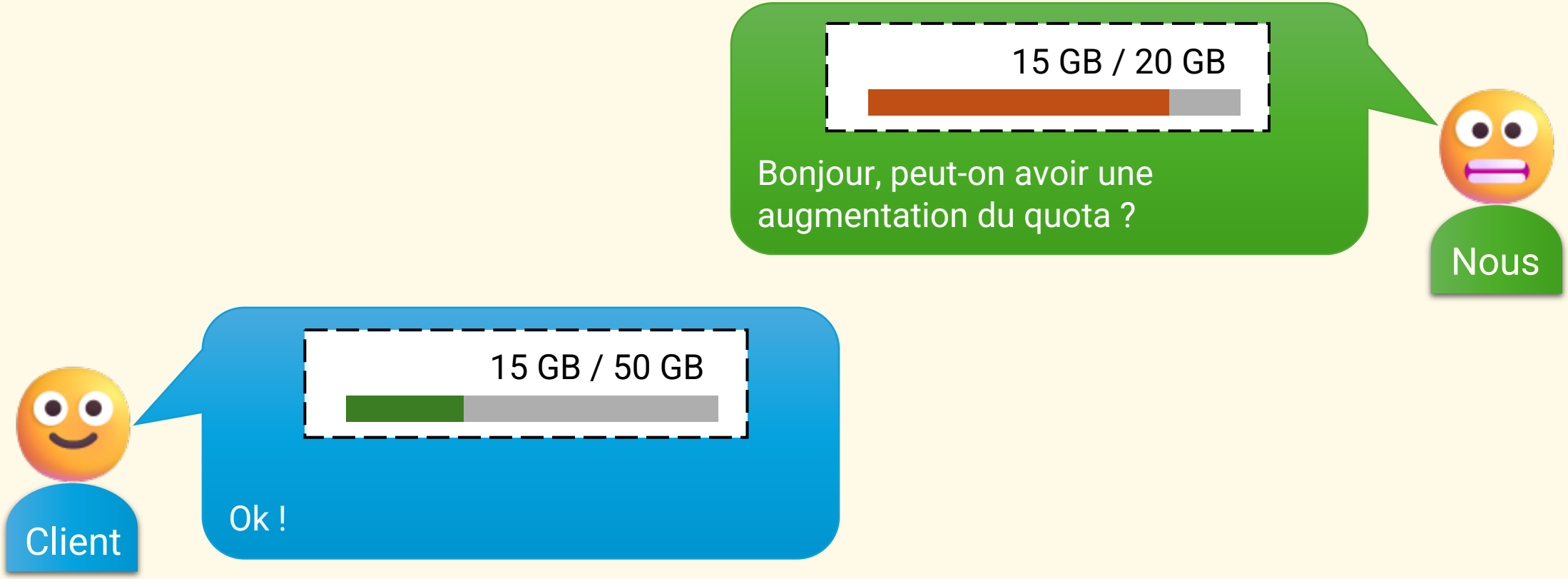
UN PEU DE CONTEXTE …

Un **registry client** et un quota **limité**.



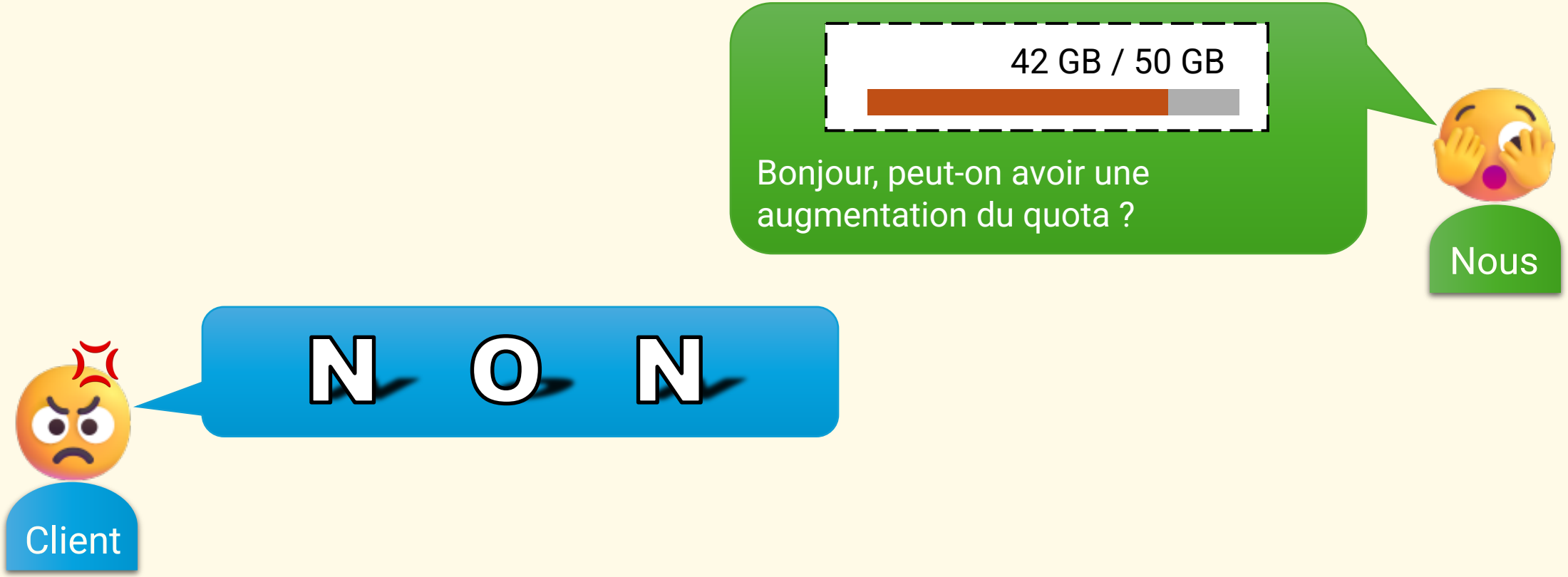
UN PEU DE CONTEXTE …

Un **registry client** et un quota **limité**.



UN PEU DE CONTEXTE …

Un **registry client** et un quota **limité**.



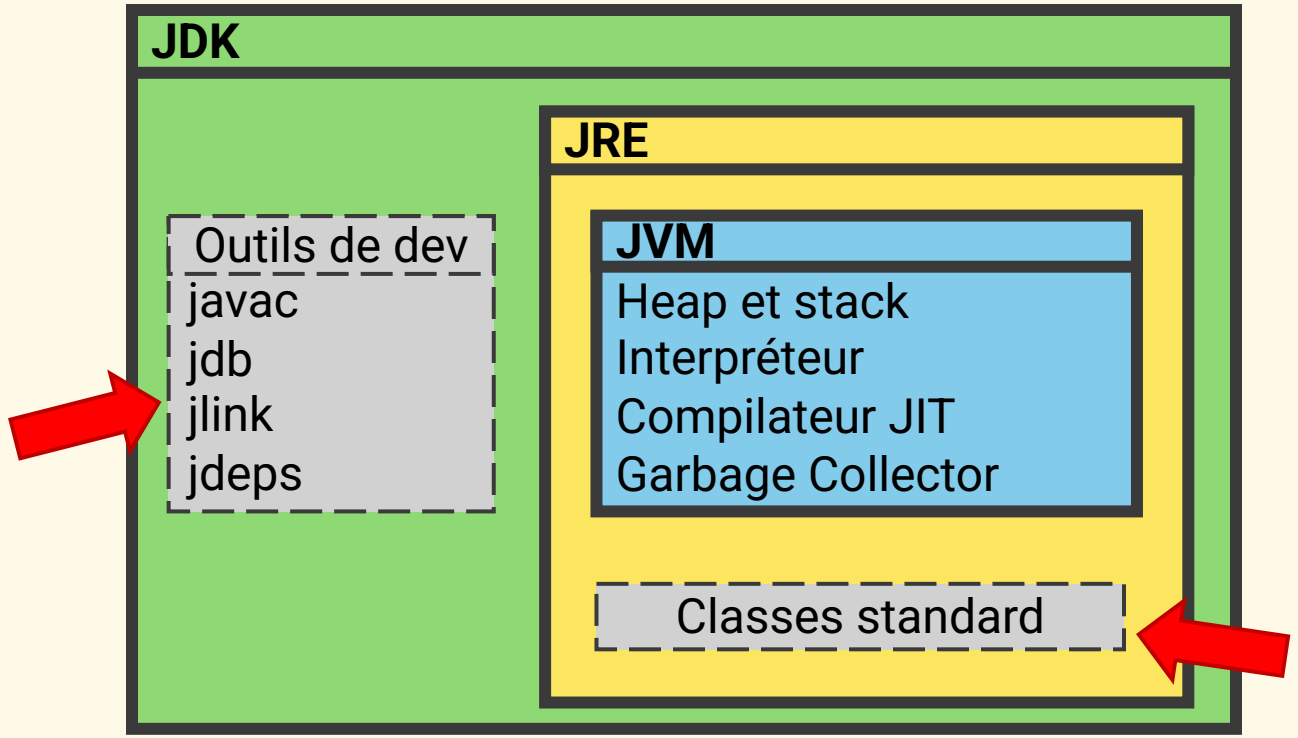
UN PEU DE CONTEXTE …

Au lieu d'**augmenter** le quota… 

Et si on **réduisait** la taille de nos images ? 

Un peu de rappels …

UN PEU DE RAPPELS …



En **production**, on utilise un **JRE**.

Les classes Java sont réparties dans des **modules**.

UN PEU DE RAPPELS …

Avant Java 9, les classes standard étaient dans **`lib/rt.jar`**.

Depuis Java 9, les classes standard sont dans **`lib/modules`**.

```
$ du -h --max-depth=0 jre-17.0.18/
136M    jre-17.0.18/
```

```
$ du -h jre-17.0.18/lib/modules
81M     jre-17.0.18/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
```

Le Dockerfile coupable

LE DOCKERFILE COUPABLE

```dockerfile
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 --chown=hello ./target/*.jar /app/app.jar

USER hello

EXPOSE 8080

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

- Alpine : **léger**
- **JRE** et non un JDK
- Cache apk vidé
- jar de l'application

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

Jdeps et Jlink

JDEPS

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

Pour lister les modules des **dépendances**, il faut **extraire le jar**.

JDEPS

```
$ 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.instr
ument,java.net.http,java.prefs,java.rmi,java.sc
ripting,java.security.sasl,java.sql.rowset,jdk.
jfr,jdk.management,jdk.net,jdk.security.jgss,jd
k.unsupported
```

- On donne le **jar**
- On donne les **libs**
- On redirige la sortie vers un fichier

JLINK

Permets de créer un runtime custom, à partir de :

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

JLINK

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

- Répertoire de sortie
- Fichier modules

```
$ du -h --max-depth=0 jre/
55M    jre/
```

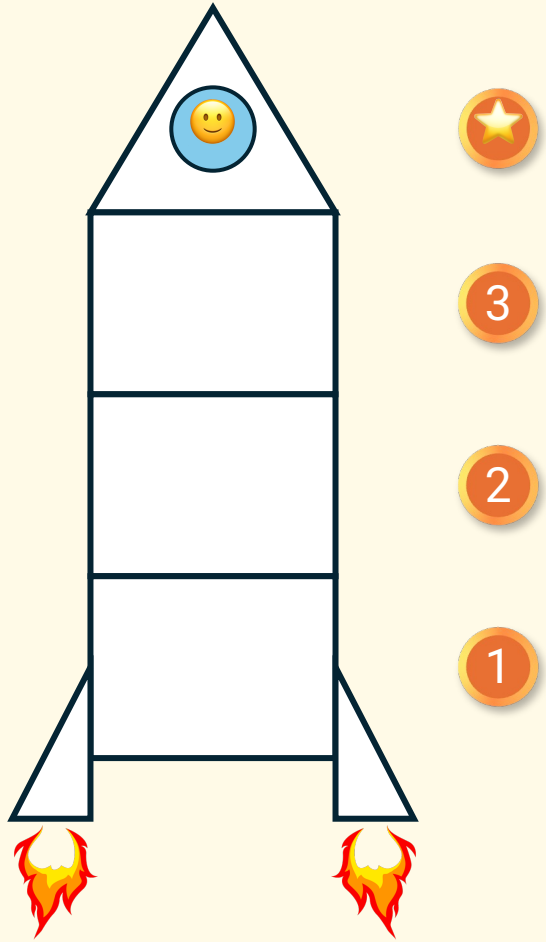
- JRE de à **136M** à **55M**

```
$ du -h jre/lib/modules
24M    jre/lib/modules
```

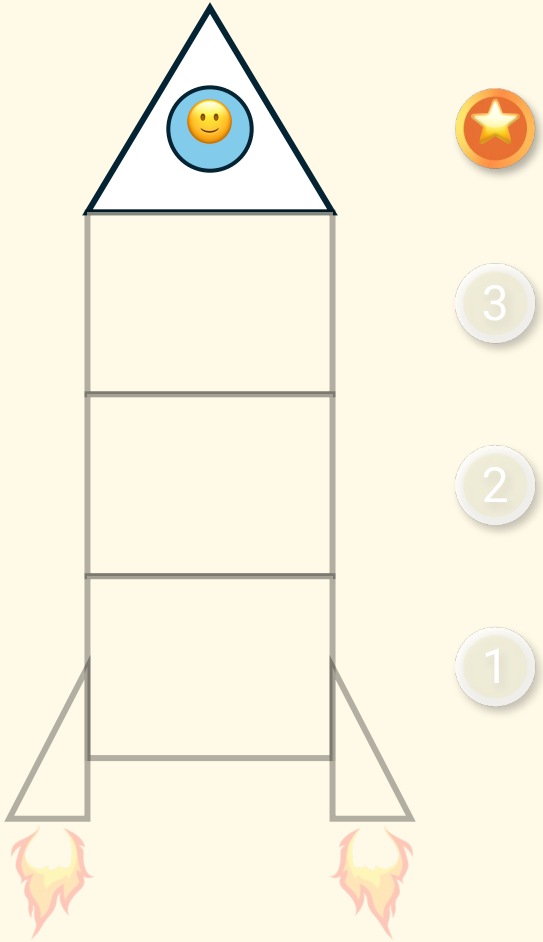
- Modules de **81M** à **24M**

Intégration dans un Dockerfile

DOCKERFILE MULTISTAGE



DOCKERFILE MULTISTAGE



DOCKERFILE MULTISTAGE

DOCKERFILE MULTISTAGE

```dockerfile
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
```

- Stage de build
- Stage applicatif
- Le jar vient du **stage de build** !

DOCKERFILE MULTISTAGE

```
FROM alpine:3.20.3 AS builder
RUN apk add –no-cache openjdk17-jdk
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
# …
```

- Stage de build
- Installation JDK 17
- Copie du jar
- Dézippage du jar
- Listing des modules
- Suppression du jar dézippé

DOCKERFILE MULTISTAGE

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

- Sortie dans **/jre**
- Utilisation du **fichier jdeps**

DOCKERFILE MULTISTAGE

```
# …
FROM alpine:3.20.3

RUN addgroup -g 1000 -S hello && \
        adduser -u 1000 -G hello -D hello

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

COPY --from=builder /jre $JAVA_HOME

COPY --chown=hello ./target/*.jar /app/app.jar

USER hello

WORKDIR /app

EXPOSE 8080

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

- JRE **apk** remplacé par **runtime custom**

DOCKERFILE MULTISTAGE

Avant ❌

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

Après ✅

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

DOCKERFILE MULTISTAGE

Avant ❌

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

Après ✅

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



DOCKERFILE MULTISTAGE

Avant ❌

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

Après ✅

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



DOCKERFILE MULTISTAGE

Avant ❌

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

Après ✅

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



DOCKERFILE MULTISTAGE

Avant ❌

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

Après ✅

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



DOCKERFILE MULTISTAGE

Avant ❌

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

Après ✅

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



Layers

LAYERS

Une **commande** du Dockerfile.

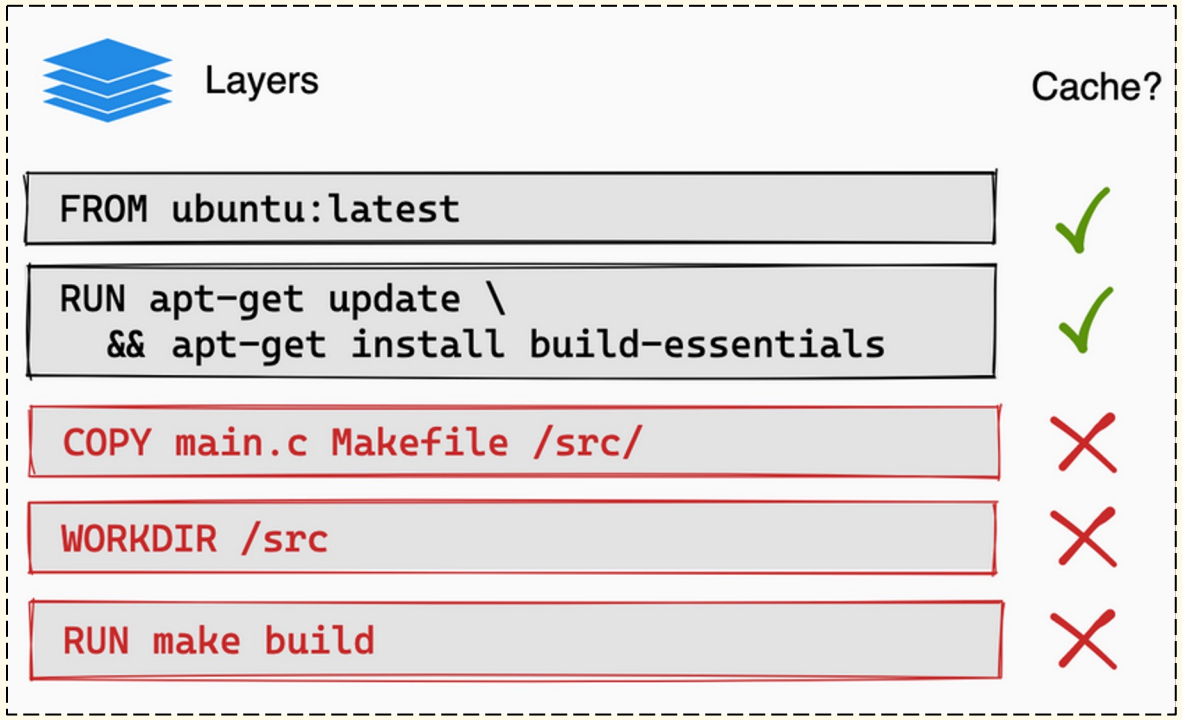
Composent une image Docker.

Un **changement de fichiers**.

Peuvent être **partagés** d'une image à une autre.

Permettent d'**accélérer les builds**.

LAYERS


Layers
Cache?
FROM ubuntu:latest
RUN apt-get update \
&& apt-get install build-essentials
COPY main.c Makefile /src/
WORKDIR /src
RUN make build

LAYERS

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

LAYERS

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

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





LAYERS

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

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





LAYERS

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

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





LAYERS

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

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





LAYERS

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

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





LAYERS

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

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





Layers Spring

LAYERS SPRING

Spring Boot propose ses layers.

Le jar est transformé en plusieurs **dossiers.**

```
$ java -Djarmode=layertools -jar app.jar extract
$ ls
application/
dependencies/
snapshot-dependencies/
spring-boot-loader/
$ java org.springframework.boot.loader.launch.JarLauncher
```

LAYERS SPRING

```dockerfile
### 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 /app/dependencies/ /app/
COPY --from=builder /app/snapshot-dependencies/ /app/
COPY --from=builder /app/spring-boot-loader/ /app/
COPY --from=builder /app/application/ /app/
#...
ENTRYPOINT ["java", "org.springframework.boot.loader.launch.JarLauncher"]
```

- Stage builder : **extraction** des layers du jar
- Jar remplacé par **layers Spring**

LAYERS SPRING

Nouvelle release, j'ajoute une feature dans mon code.

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/ ./	✗

Uniquement le code change. ✅

Uniquement le layer applicatif est rebuildé. ✅

LAYERS SPRING

```
$ 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/
[...]
```

Les layers sont bien **cachés**. ✅

Uniquement le **layer applicatif** est rebuildé. ✅

LAYERS SPRING

```
$ docker history b91f8acb7300
```

```
IMAGE          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
```

75 mo de layers **réutilisés**.

20 ko de layers **rebuildés**.

LAYERS SPRING

Lors du push ou du pull …

19.7kB

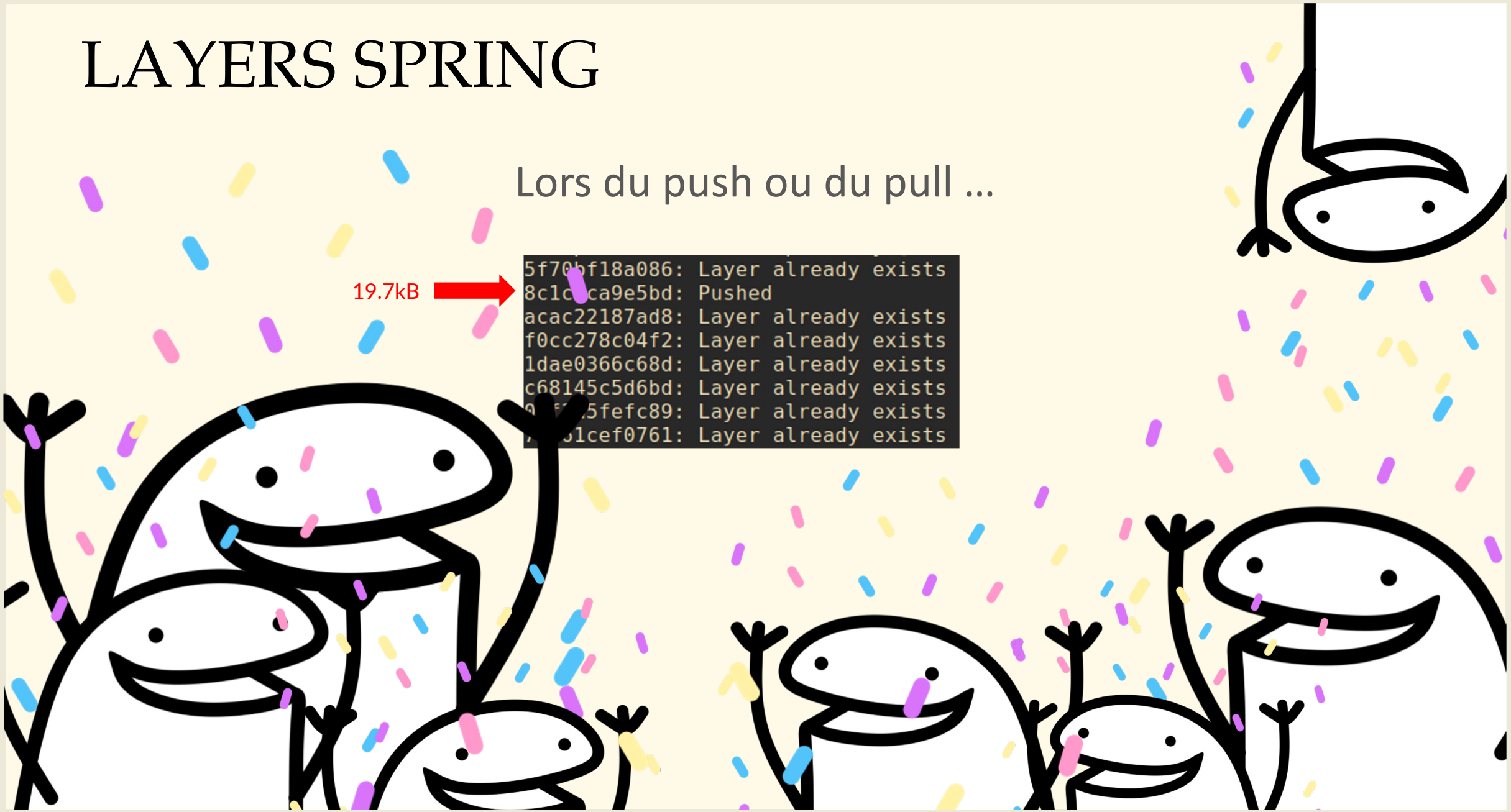
```
5f70bf18a086: Layer already exists
8c1c2ca9e5bd: Pushed
acac22187ad8: Layer already exists
f0cc278c04f2: Layer already exists
1dae0366c68d: Layer already exists
c68145c5d6bd: Layer already exists
06f3a5fefc89: Layer already exists
78561cef0761: Layer already exists
```

LAYERS SPRING

Lors du push ou du pull …

19.7kB

```
5f70bf18a086: Layer already exists
8c1c  ca9e5bd: Pushed
acac22187ad8: Layer already exists
f0cc278c04f2: Layer already exists
1dae0366c68d: Layer already exists
c68145c5d6bd: Layer already exists
0  5fefc89: Layer already exists
  1cef0761: Layer already exists
```



LAYERS SPRING

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

Le poids total de l'image reste le même. ⚠️

Des layers sont **réutilisés**. ♻️

Garbage Collector

GARBAGE COLLECTOR

Cache de build plein = builds KO ! ❌

La solution, le **Garbage Collector**. 🗑️

Présent **par défaut**, mais **personnalisable**. ✍️

GARBAGE COLLECTOR

/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 },
            ],
        },
    },
}
```

- À chaque **seuil en GB**, on supprime **les layers les moins utilisés**
- Plus on approche de la limite, plus on supprimer des layers utilisés

Réflexion

RÉFLEXION

Exemple, appel d'une API GET.

```
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());
```

- Classes **`HttpClient`**, **`HttpRequest`** et **`HttpResponse`**
- Module **`java.net`**

RÉFLEXION

La version API 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);
```

`java.net` est utilisé de manière implicite.

RÉFLEXION

Jdeps ne détecte que **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

Jdeps ne détecte que **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

La solution, fournir le module **manuellement**.

```
FROM alpine:3.20.3 AS builder
#...
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 && \
        echo ",java.net" >> /app/dependencies.info && \
        rm -rf /app/unpacked
# …
```

Conclusion

CONCLUSION

Pour optimiser les images Docker :

 **Jdeps et Jlink** pour utiliser un JRE custom : **réduire la taille des images**.

♻️ Les **layers** : privilégier la **réutilisation**.

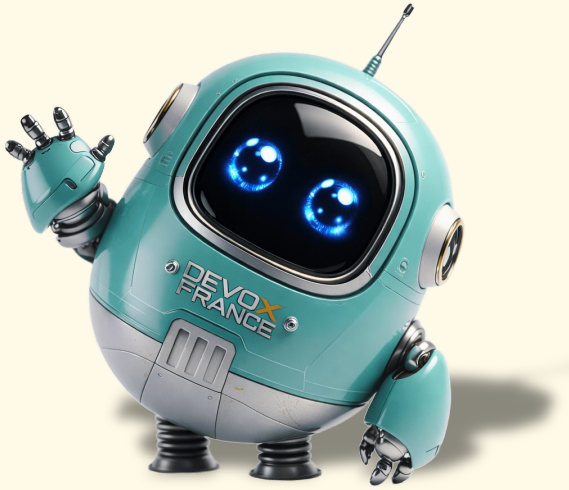
Attention !

 En cas de builds régulier, **tuner le Garbage Collector**.

 L'API réflexion, **c'est le mal** !

Merci !

Slides



? Votre avis