SPRING BOOT DANS UN CONTAINER

OUTILS ET PRATIQUES

SPRING MEETUP PARIS

18 / 12 / 2019

DANIEL GARNIER-MOIROUX

Software Engineer @ Pivotal Labs

@Kehrlann

github.com/kehrlann/spring-boot-in-a-container

QUI UTILISE DES CONTAINERS EN PRODUCTION AUJOURD'HUI ?

CHOIX DE L'IMAGE DE BASE

DOCKERFILE, V1.0

```
FROM ubuntu:latest

RUN apt update && apt install openjdk-8-jre -y

COPY spring-petclinic/target/spring-petclinic-*.jar /app.jar

ENTRYPOINT ["java","-jar","/app.jar"]
```

UN PEU MIEUX...

```
FROM openjdk:8-jre

COPY spring-petclinic/target/spring-petclinic-*.jar /app.jar

ENTRYPOINT ["java","-jar","/app.jar"]
```

PLUS LÉGER !

```
FROM openjdk:8-jre-alpine

COPY spring-petclinic/target/spring-petclinic-*.jar /app.jar

ENTRYPOINT ["java","-jar","/app.jar"]
```

ENCORE MOINS DE SURFACE

```
FROM gcr.io/distroless/java

COPY spring-petclinic/target/spring-petclinic-*.jar /app.jar

CMD ["/app.jar"]
```

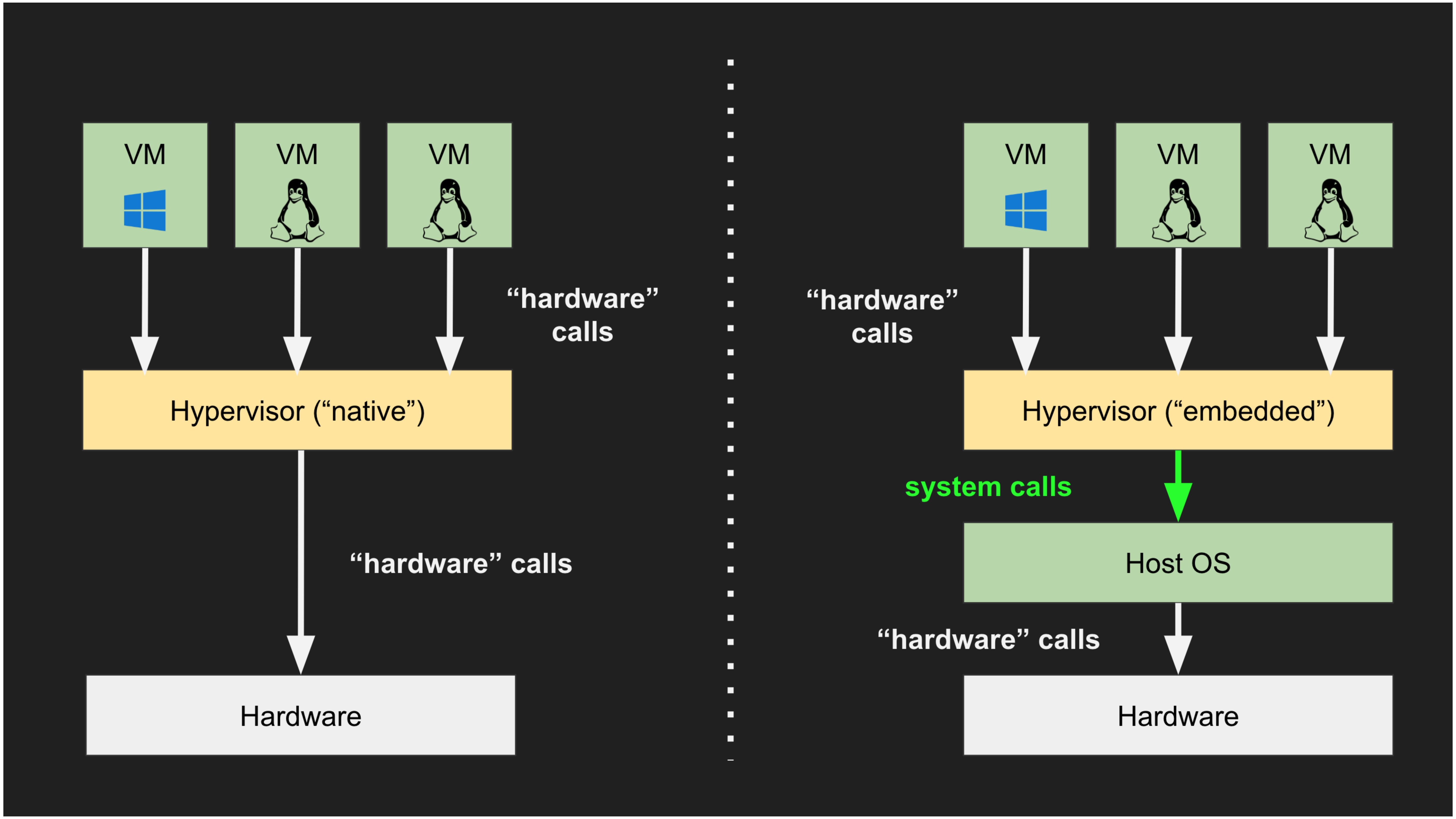
(... mais un peu plus gros en taille)

IMAGE DE BASE - TAKE-AWAYS

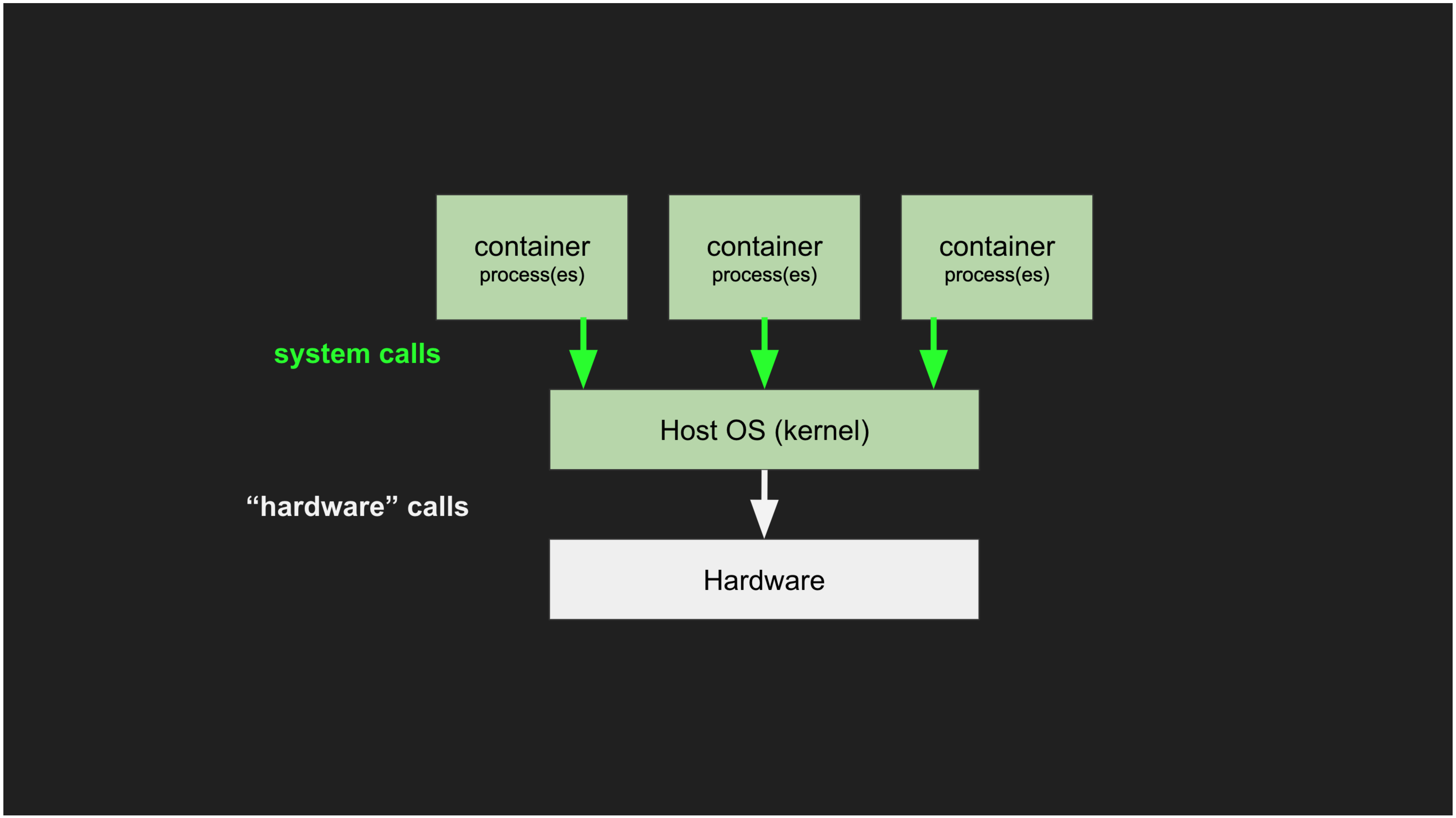
- Container != VM
- Utiliser une image spécialisée
- Penser à la taille (ex: alpine)
- Penser à la sécurité (ex: distroless)

INTERLUDE

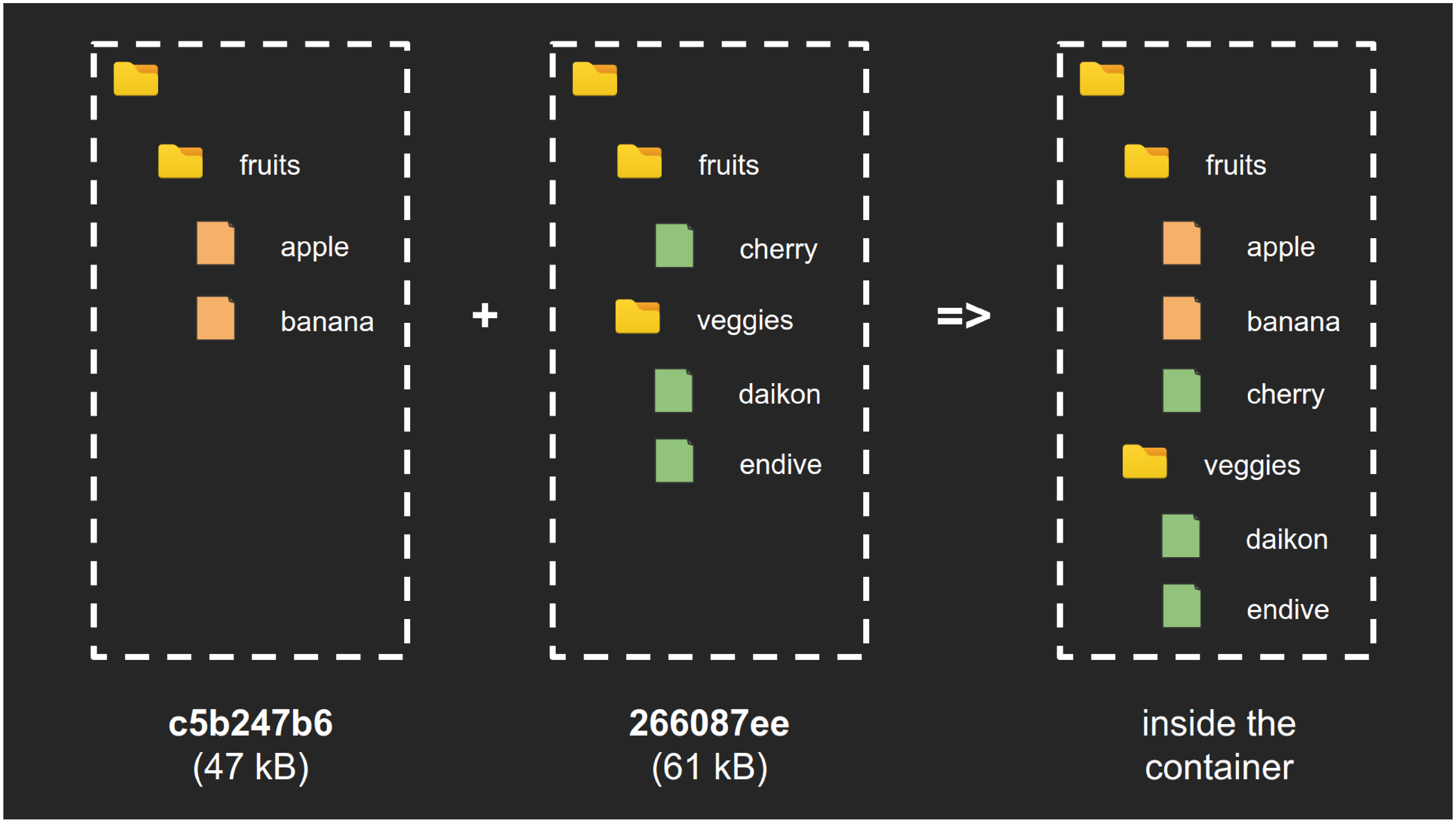
VIRTUALISATION



CONTAINERS


container
process(es)
container
process(es)
container
process(es)
system calls
Host OS (kernel)
“hardware” calls
Hardware

LES LAYERS: COMMENT ÇA MARCHE



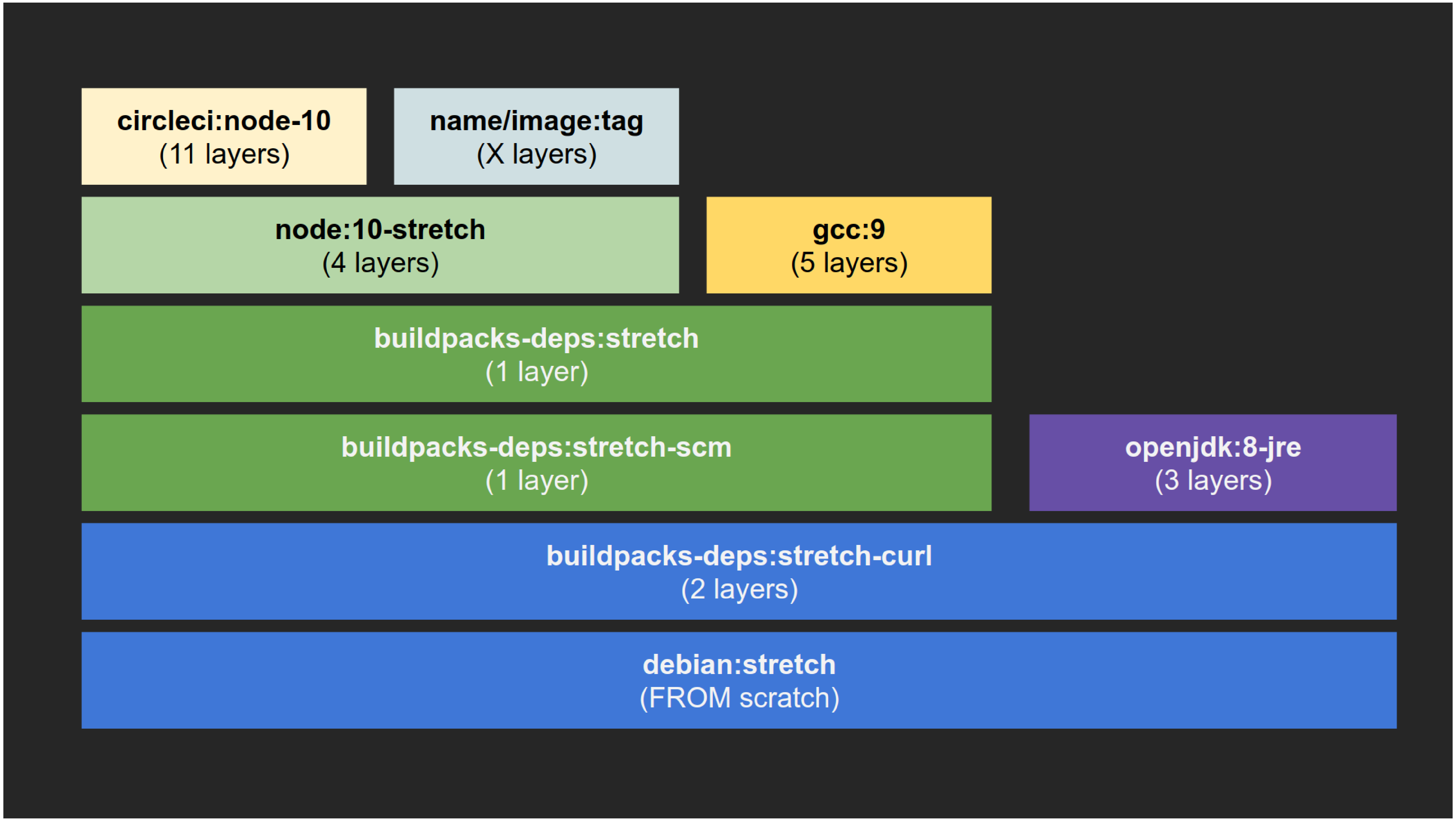
LES LAYERS

FROM réutilise toutes les layers de l'image de base.

Les layers sont créées par :

- **RUN**
- **COPY**
- **ADD**

LES LAYERS: RÉUTILISATION



REVENONS À NOS MOUTONS ...

```
FROM gcr.io/distroless/java

COPY spring-petclinic/target/spring-petclinic-*.jar /app.jar

CMD ["/app.jar"]
```

LAYERING: EXTRACTION DES DÉPENDANCES

```
DEPS_FOLDER=$PWD/spring-petclinic/target/dependency
mkdir -p "$DEPS_FOLDER"
cd "$DEPS_FOLDER"
jar -xf ../*.jar
```

LAYERING: CRÉATION DE L'IMAGE

```
FROM gcr.io/distroless/java

ARG DEPENDENCY=spring-petclinic/target/dependency

COPY ${DEPENDENCY}/BOOT-INF/lib /app/lib
COPY ${DEPENDENCY}/META-INF /app/META-INF
COPY ${DEPENDENCY}/BOOT-INF/classes /app

ENTRYPOINT [
  "java",
  "-cp",
  "app:app/lib/*",
  "org.springframework.samples.petclinic.PetClinicApplication"
]
```

CONSTRUIRE SES IMAGES - TAKE-AWAYS

- Choisir son image de base (taille, sécurité, ...)
- Séparer dépendances & code dans des layers séparées

BUILD PROCESS

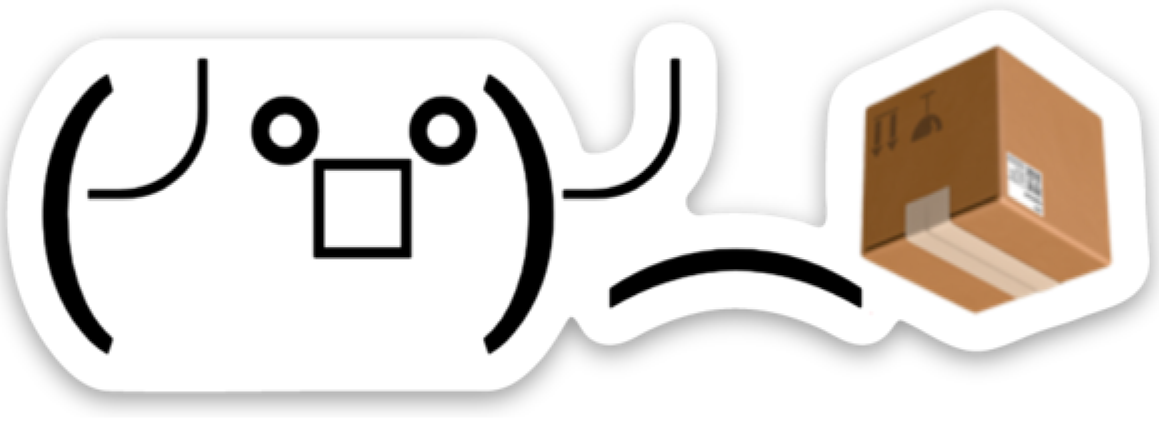
MULTI-STAGE BUILDS

```dockerfile
# BUILD SOURCE
FROM openjdk:8-jdk-alpine as build
WORKDIR /workspace/spring-petclinic/

COPY spring-petclinic/mvnw .
COPY spring-petclinic/.mvn .mvn
COPY spring-petclinic/pom.xml .
COPY spring-petclinic/src src

RUN ./mvnw package -DskipTests
RUN mkdir -p target/dependency && (cd target/dependency; jar -xf ../*.jar)

# BUILD IMAGE
FROM gcr.io/distroless/java

ARG DEPENDENCY=/workspace/spring-petclinic/target/dependency

COPY --from=build ${DEPENDENCY}/BOOT-INF/lib /app/lib
COPY --from=build ${DEPENDENCY}/META-INF /app/META-INF
COPY --from=build ${DEPENDENCY}/BOOT-INF/classes /app

ENTRYPOINT [
  "java",
  "-cp",
  "app:app/lib/*",
  "org.springframework.samples.petclinic.PetClinicApplication"
]
```

MULTI-STAGE BUILDS, WITH CACHING

```
# BUILD SOURCE
FROM openjdk:8-jdk-alpine as build
WORKDIR /workspace/spring-petclinic/

COPY spring-petclinic/mvnw .
COPY spring-petclinic/.mvn .mvn
COPY spring-petclinic/pom.xml .
COPY spring-petclinic/src src

RUN --mount=type=cache,target=/root/.m2 ./mvnw clean package -DskipTests
RUN mkdir -p target/dependency && (cd target/dependency; jar -xf ../*.jar)

# BUILD IMAGE
FROM gcr.io/distroless/java

ARG DEPENDENCY=/workspace/spring-petclinic/target/dependency

COPY --from=build ${DEPENDENCY}/BOOT-INF/lib /app/lib
COPY --from=build ${DEPENDENCY}/META-INF /app/META-INF
COPY --from=build ${DEPENDENCY}/BOOT-INF/classes /app

ENTRYPOINT [
  "java",
  "-cp",
  "app:app/lib/*",
  "org.springframework.samples.petclinic.PetClinicApplication"
]
```

REALLY ?



GOOGLE JIB

Dans votre `pom.xml`, il suffit d'ajouter:

```xml
<build>
  [...]
  <plugins>
    <plugin>
      <groupId>com.google.cloud.tools</groupId>
      <artifactId>jib-maven-plugin</artifactId>
      <version>1.3.0</version>
      <configuration>
        <to>
          <image>kehrlann/pet-clinic:jib</image>
        </to>
      </configuration>
    </plugin>
  </plugins>
</build>
```

Puis, run:

```
$ mvn compile jib:build
```

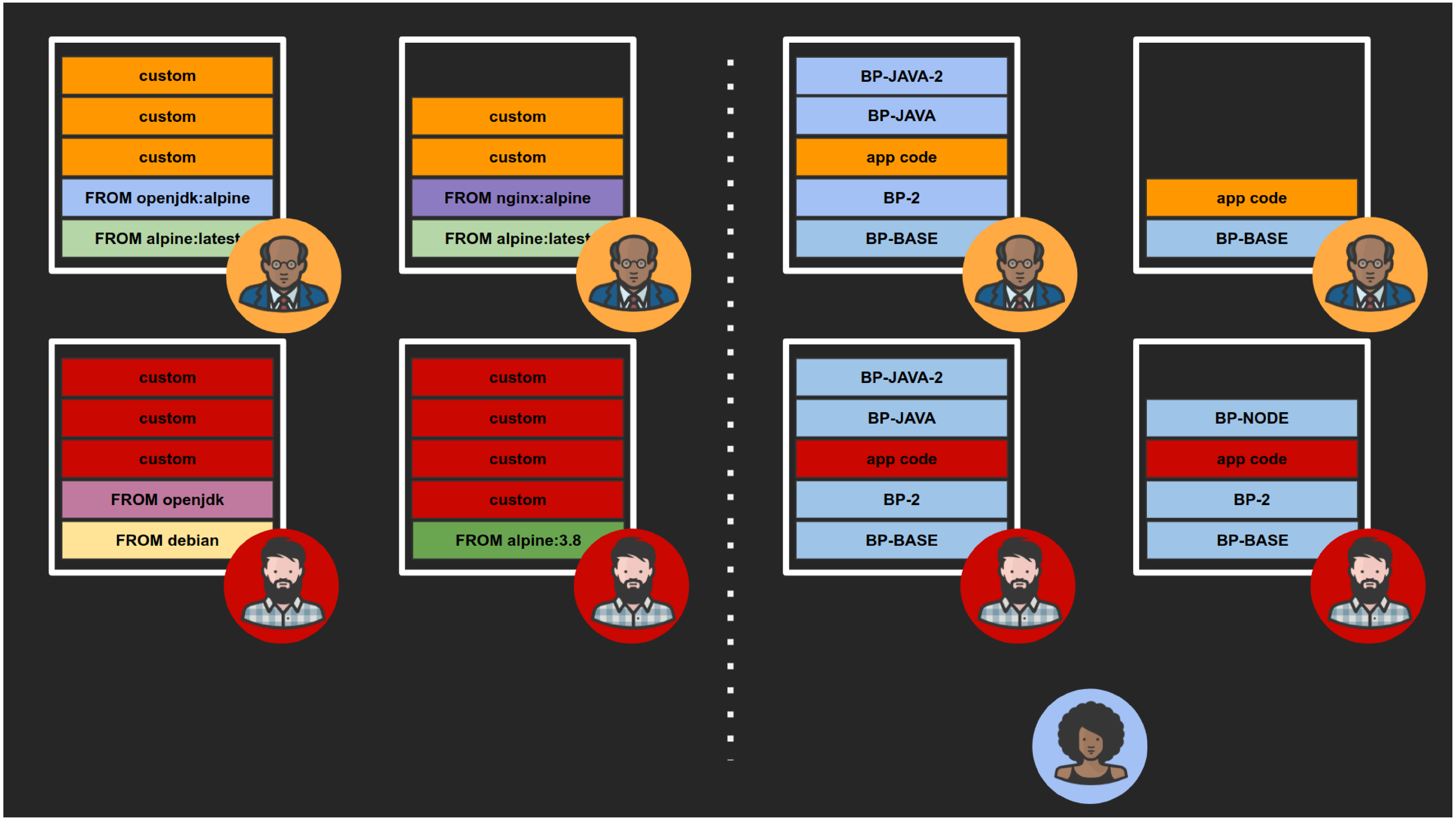
Voire:

```
$ mvn compile jib:dockerBuild
```

ALSO, CLOUD-NATIVE BUILDPACKS

Buildpacks.io

```
$ pack build kehrlann/pet-clinic:pack-cf-cn-buildpack --builder=cloudfoundry/cnb
```


custom
custom
custom
FROM openjdk:alpine
FROM alpine:latest
custom
custom
FROM nginx:alpine
FROM alpine:latest
BP-JAVA-2
BP-JAVA
app code
BP-2
BP-BASE
app code
BP-BASE
custom
custom
custom
FROM openjdk
FROM debian
custom
custom
custom
custom
FROM alpine:3.8
BP-JAVA-2
BP-JAVA
app code
BP-2
BP-BASE
BP-NODE
app code
BP-2
BP-BASE

KPACK

https://github.com/pivotal/kpack/

⚠️ experimental

```yaml
apiVersion: build.pivotal.io/v1alpha1
kind: Image
metadata:
  name: my-image-k8s-name
spec:
  tag: kehrlann/pet-clinic:kpack
  builder:
    name: default-builder
    kind: ClusterBuilder
  source:
    git:
      url: "https://github.com/spring-projects/spring-petclinic.git"
      revision: master
```

BUILD PROCESS - TAKE-AWAYS

- Le process de build, c'est important
- Mais si on peut éviter d'y penser, c'est mieux
- Standardiser la création d'images, c'est top

A VOTRE TOUR !

https://spring.io/guides/topicals/spring-boot-docker/

https://github.com/Kehrlann/spring-boot-in-a-container/

Faites moi signe: @Kehrlann