Container Builders : Which is the best image builder ?

Date : 13 avril 2023
Speakers : Christian Nader (Décathlon)
Format : Tools in action (30mn)

Il existe différentes façons de construire des images Docker. Le cas d'utilisation de l'image doit aider au choix de la solution.

Ordre du jour :

1. Création d'image de conteneurs : JDK, JRE, JLINK, Distroless (JIB), CNB, Native (GraalVM)
2. Analyse d'image statique
3. Démarrer les conteneurs
4. Benchmarks

Contexte technique : Java 17, Spring Boot 3.0.5, avec des images OCI compliant

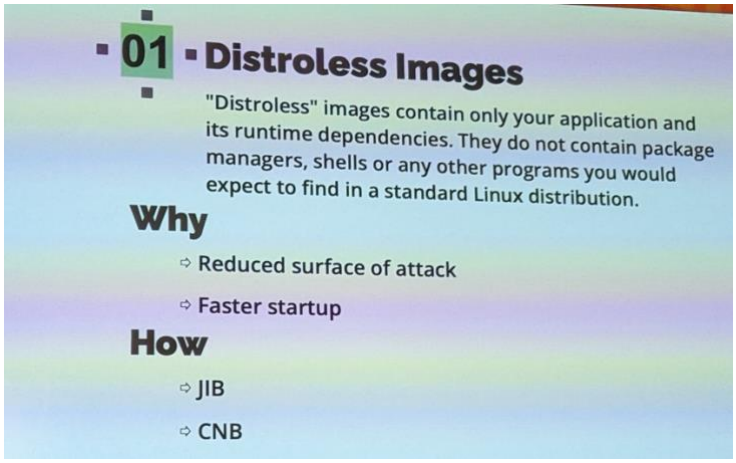
Pour ses démos, Christian ne s'est pas contenté d'un simple Hello World. Il a développé une application qui génère un labyrinthe. Stack technique utilisée : Spring Webflux, sérialisation JSON, images buffer, I/O …

1. Dockerfile mutli stage

Open JDK 17 Alpine
Image de 380 Mo
En utilisant un JRE au lieu d'un JDK on passe à 191Mo

Avec des custom JRE et JLink on peut choisir au build les bouts de JRE nécessaires à l'exécution puis utiliser une simple Alpine en image de base 113Mo

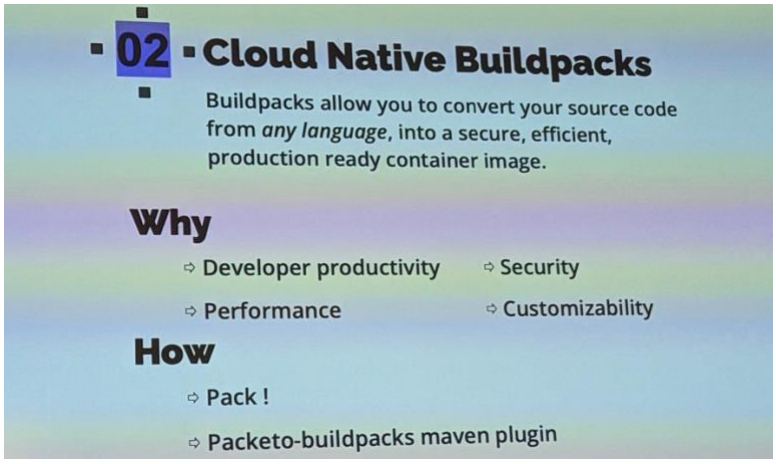
2. JIB



Ajout dans le pom.xml du plugin maven jib
Pas besoin Dockerfile ni de Docker sur la CI

Les images Distroless permettent de réduire la surface d'attaque et un démarrage plus rapide.
Taille de l'image Docker JIB : 288Mo

3. CNB Cloud Native Buildpacks



Buildpacks est géré par la CNCF
On peut utiliser Pack ! ou le plugin maven paketo-buildpacks. Le support de Docker de Spring Boot 3 s'appuie sur buildpacks.
Taille des images : 278Mo et 229Mo

4. Native CNB

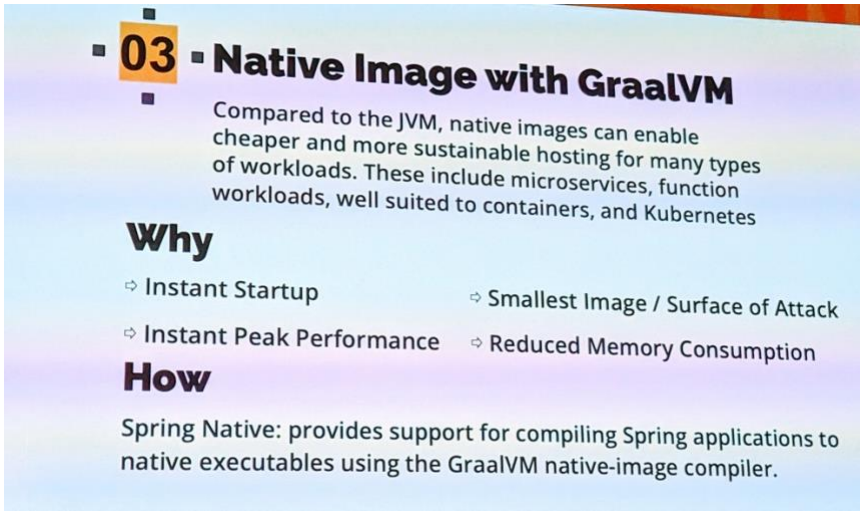
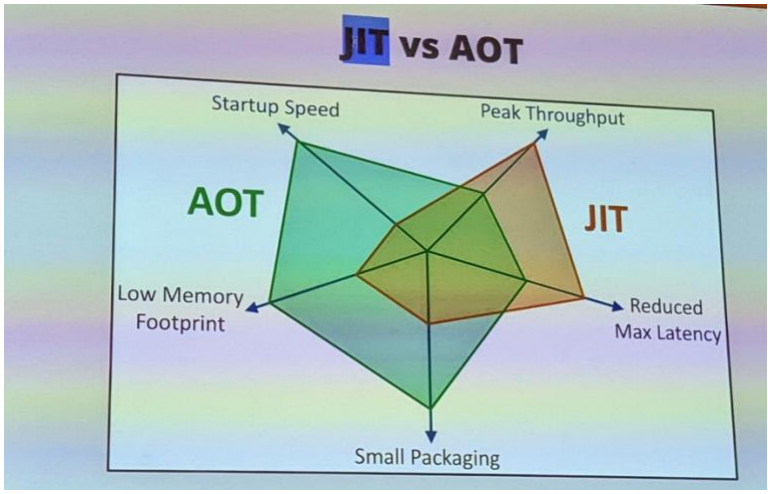


Image native de GraalVM
Why :

- Démarrage immédiat
- Images plus petites, réduction de la surface d'attaque
- Réduction de l'empreinte mémoire

JIT vs AOT

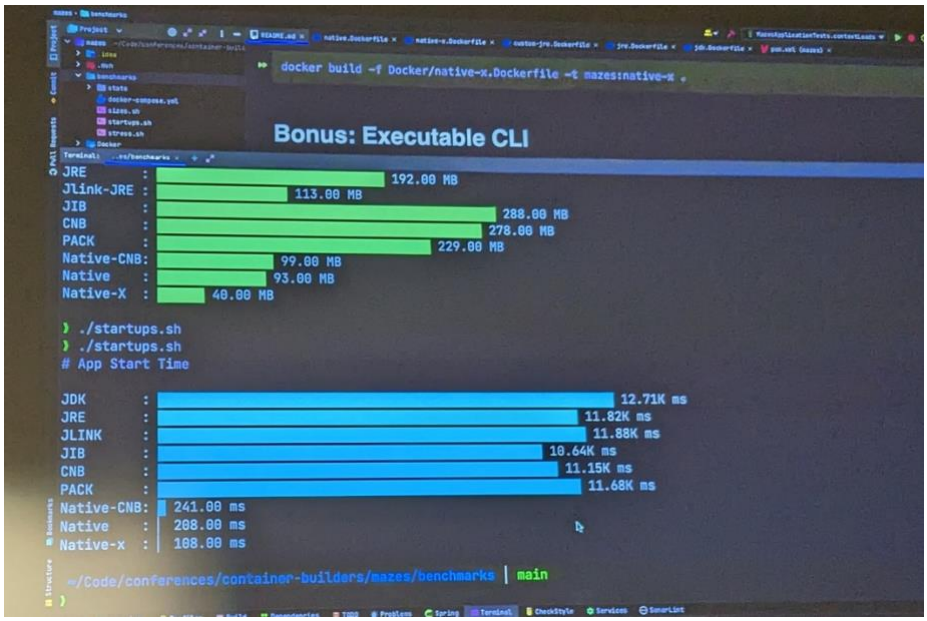


Création de l'image native avec Spring Boot 3 et Spring native.
Prérequis : environnement de build avec 8go de RAM. Après 8mn de build, on a une image de 98Mo.

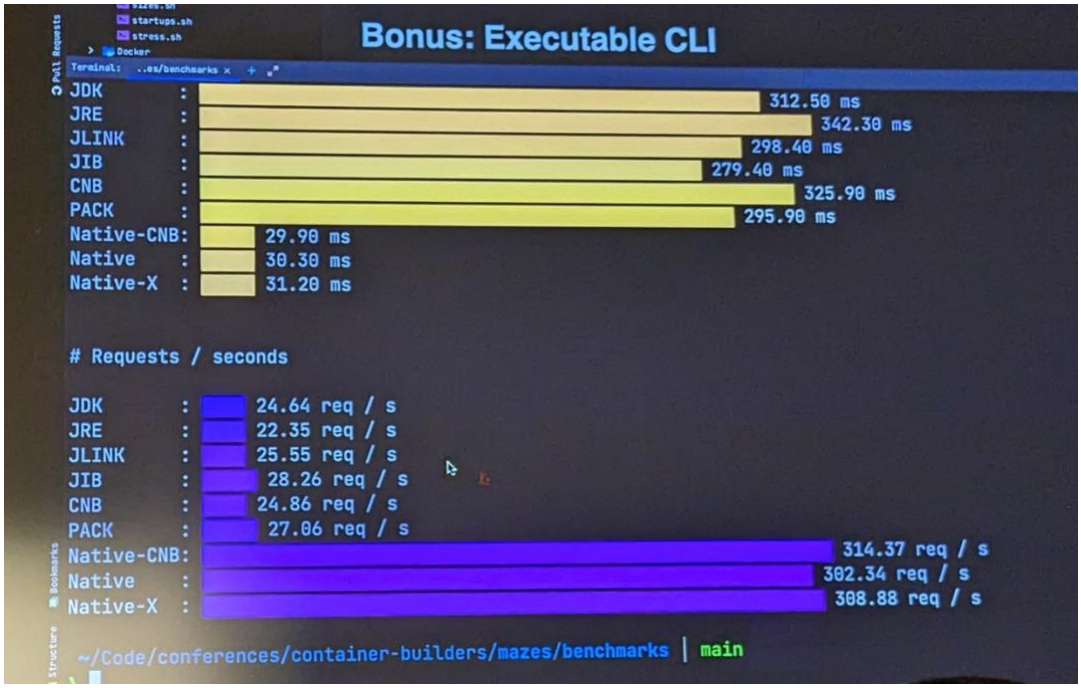
Dockerfile utilisant la commande *mvn -Pnative clean package* et une image distroless au runtime.
Image de 93Mo. Possibilité de compresser le binaire avec UPX : **image de 39Mo**.

Benchmarks

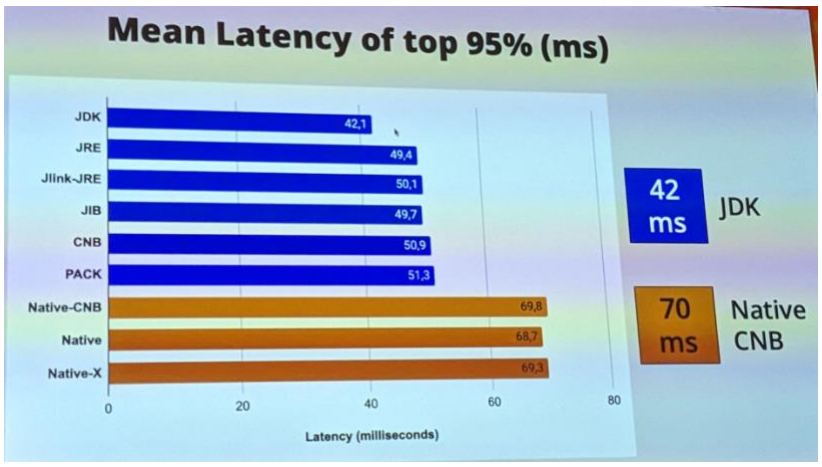
Utilisation d'un docker-compose qui démarre tous les conteneurs
Empreinte mémoire, temps de démarrage et benchmarks



En troughput : 10x plus de requêtes par seconde


Bonus: Executable CLI
startups.sh
stress.sh
Docker
Terminal: ..es/benchmarks
JDK : 312.50 ms
JRE : 342.30 ms
JLINK : 298.40 ms
JIB : 279.40 ms
CNB : 325.90 ms
PACK : 295.90 ms
Native-CNB: 29.90 ms
Native : 30.30 ms
Native-X : 31.20 ms
Requests / seconds
JDK : 24.64 req / s
JRE : 22.35 req / s
JLINK : 25.55 req / s
JIB : 28.26 req / s
CNB : 24.86 req / s
PACK : 27.06 req / s
Native-CNB: 314.37 req / s
Native : 302.34 req / s
Native-X : 308.88 req / s
~/Code/conferences/container-builders/mazes/benchmarks | main


Packaging Size (MB)
.JAR 21
Native-exe 50
JDK 380
JRE 192
Jlink-JRE 113
JIB 288
CNB 278
PACK 229
Native-CNB 99
Native 93
Native-X 40
0
100
200
300
400
Size (MB)
40 Mb
Native-x
380 Mb
JDK


Mean Latency of top 95% (ms)
JDK 42,1
JRE 49,4
Jlink-JRE 50,1
JIB 49,7
CNB 50,9
PACK 51,3
Native-CNB 69,8
Native 68,7
Native-X 69,3
0
20
40
60
80
Latency (milliseconds)
42 ms
JDK
70 ms
Native CNB

Quel est le meilleur builder ?
Comme précisé en introduction, cela dépend du use case :

Besoin de démarrer très souvent Desservir instantanément des requêtes	Native
La performance de pointe est reine Longue durée de fonctionnement	JDK
Fourre-tout	JKINK, JIB, CNB