GraalVM en prod

Moins de RAM, moins de taille applicatif, des temps de boot plus rapides
Dans la logique de maîtrise des coûts infra
Mais pas sans travaux ni impacts !

https://www.meetup.com/fr-FR/JUG-Montpellier/

Laurent Perez @laurentperez

Intro

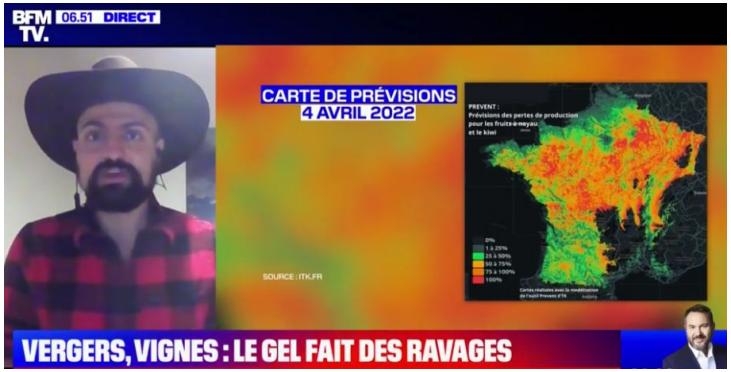
-

Meetup hébergé par

RECHERCHES & HORIZONS
#recrutement #conseil #rh

Speaker & use case

- @laurentperez , itk
- On traite beaucoup de data IoT & météo
  - Pour de la prévention de risque climatique
  - Pour le bien être végétal et animal
- On fait tourner beaucoup de code
  - Java Kotlin Python first class citizens, mais pas que



@SergeZaka

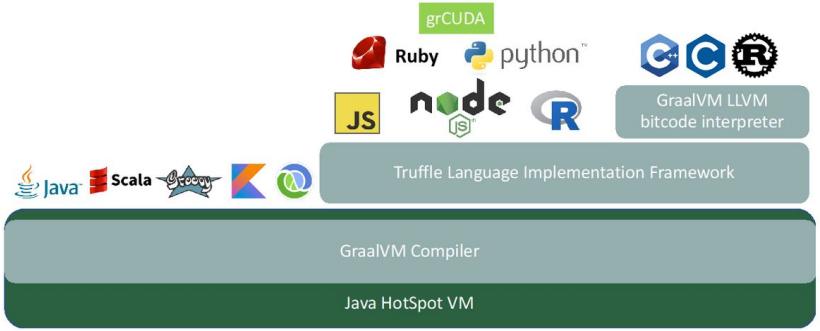
GraalVM c’est quoi ?

- Une machine virtuelle
  - Polyglotte
    - Tourne du Java, python, R, JS, wasm, CPP/Rust, offre de l’interop
- Avec un compilateur en 2 modes
  - “libgraal” AOT (= machine code connu au moment du build)
  - “jargraal” JIT (= machine code connu au moment du run sur serveur)
- Deux éditions
  - CE gratuite
  - EEnterprise : payante***, avec optimisations de compile, de GC, d’outillage
  - *** : “at no cost” under a Java SE Subscription
- Issue de Oracle Labs

Où se place GraalVM ?

Polyglotte ?

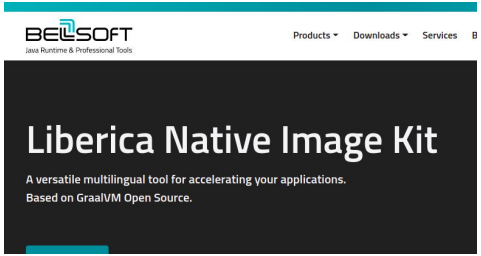
- Via “Truffle”
  - https://www.graalvm.org/22.0/graalvm-as-a-platform/language-implementation-framework/
- Java first class citizen
- Des langages sont réimplémentés (R, Python, ECMAScript, Ruby, CUDA, wasm) à divers stades de complétion
- D’autres passent par l’intermédiaire de bitcode LLVM
  - C/C++, Rust
- Tu peux écrire *ton langage* - scientifique, domain specific
  - Ex : Enso pour la dataviz
- Oracle met des billes dans le multi languages



Contribuée par seulement Oracle ? oui & non

- RedHat a une version de GraalVM downstream, font de l'upstream
  - *Mandrel*, qui focus la compile native, ne gère pas le polyglotte
  - utilisée pour générée leurs images natives Quarkus @ Openshift/Kube
- Spring/VMware choisissent le "NIK" Native Image Kit de Bellsoft/Liberica
  - Le native AOT annoncé first class citizen pour Spring 6 & Spring Boot 3

- IBM a OpenJ9/Eclipse OMR : alternative à Truffle



Advisory board : il n’y a pas que Oracle

- https://www.graalvm.org/uploads/graalvm_project_advisory_board_meeting_february_2022.pdf
-
-

GraalVM Advisory Board February 2022 Meeting

Date

February 23rd, 2022

Participants

Alan Hayward (ARM), Aleksei Voitylov (BellSoft), Alina Yurenko (Oracle), Bruno Caballero (Microdoc), Chris Seaton (Shopify), Gilles Duboscq (Oracle), James Kleeh (Object Computing), Johan Vos (Gluon), Max Andersen (Red Hat), Michael Simons (Neo4j), Paul Hohensee (Amazon), San-Hong Li (Alibaba), Shaun Smith (Oracle), Thomas Wuerthinger (Oracle), Uma Srinivasan (Twitter)

A quoi ça sert ?

- Réduit les coûts en contexte microservices / briques légères
  - Démarrent très vite, en millisecondes (le downtime est un coût même en Cloud)
  - Prennent moins de RAM
    - Attention : à froid/tiède ! coucou les Stacktraces, la Heap, le off Heap, le GC, toujours là
  - Taille réduite de l'application : ça *peut* devenir un binaire natif, *c'est pas obligé*
    - Attention : il y a toujours un petit container, et un mini runtime Java *dans* le binaire
- Ouvre la porte du natif sans abandonner la JVM
  - en JIT graal se substitue au compilo C2 -server de HotSpot
- C'est un meilleur JIT qu' HotSpot *avec réserves

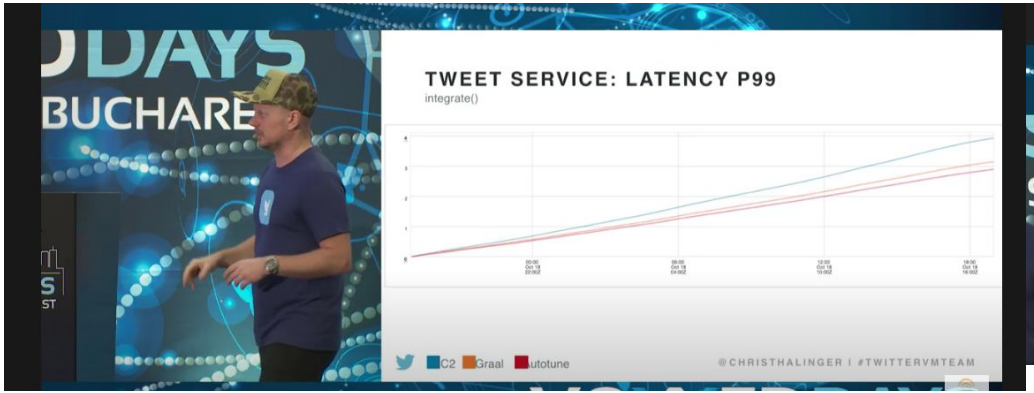
A qui ça sert ?

- Aux boites qui font du
  - Java Kotlin Scala Clojure
  - Avec des problématiques de maîtrise de coûts, rapid swarming
  - Avec intérêts pour le multilangages
- À Oracle, mais il n’y a pas qu’eux en vendeurs de Java
  - C’est leur façon de rester frais dans le cloud, le serverless, le multilangages
  - Utilisé par d’autres comme RedHat, Bellsoft/VMware
- A Twitter (leur Scala, le tweet service)
  - https://graalworkshop.github.io/2022/

Oracle and IBM are realising that WebLogic and WebSphere licensing revenue isn't going anywhere and if they want to grow revenue and achieve lock-in they have to find a new way. (reddit troll intelligent ?)

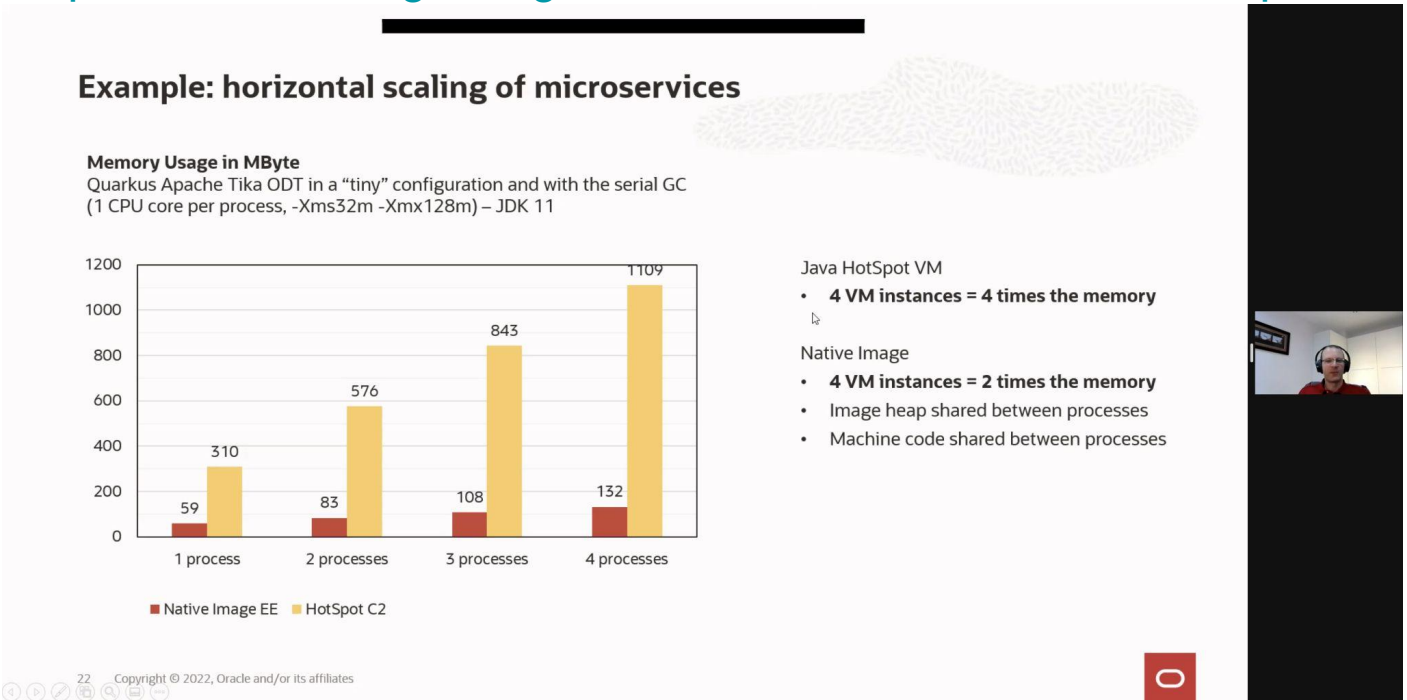
Rentable ?

- Twitter + graal : “we use 18% less machines” (on the tweet service) - 2018/9
  - Le compiler graal a encore évolué depuis, Oracle bosse sévère dessus
  - https://www.youtube.com/watch?v=jLnedMcXYEs&t=311s
  - https://jaxenter.com/graalvm-chris-thalinger-interview-163074.html
  - “Computing power will explode”. Twitter a 1000 services 100 000 instances
  - Se servent du compiler (graal) sur leur propre build de JDK optimisé Scala



Twitter + graal, Avril 2022, graal vs C2

- A la conférence International Symposium on Code Generation and Optimization
- https://twitter.com/graalcgo/status/1510258096144150531/photo/1
-



Est ce que <ma boite ici> en bénéficierait ?

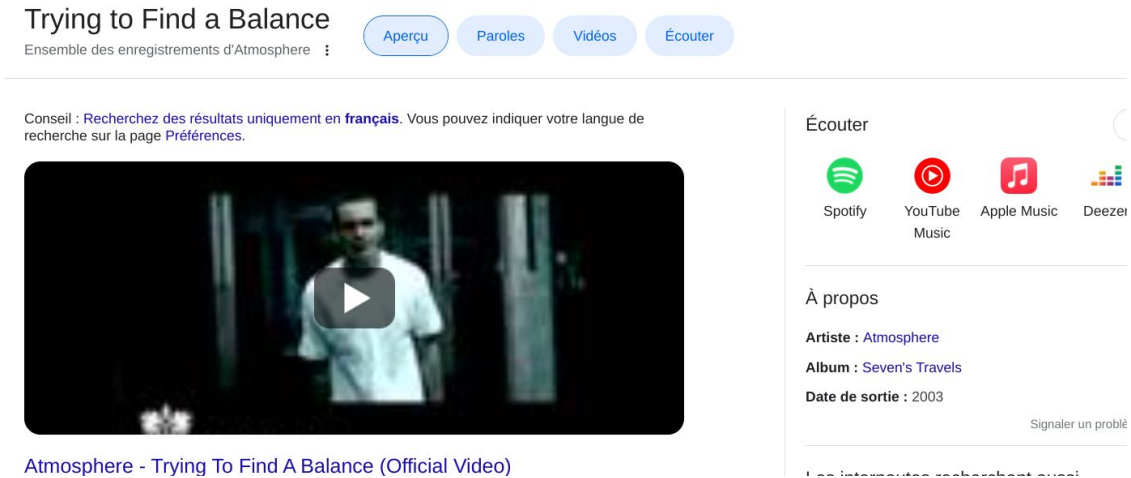
- Pour mettre davantage de briques Java dans autant ou moins de RAM
- Pour les démarrer en millisecondes : quasi zero downtime support
  - Le container pourrait aussi être plus petit sur wire, sur disque
- *Resources are scarce, expensive*
- … peut on chiffrer ce gain, les impacts ?
- Pas d’actions claires sans idées claires =>

Pas d’actions claires sans idées claires

- Sommes nous en Java *réellement microservices* ?
- Prenons une stack “modérée” : une vingtaine de microservices, avec ou sans framework
  - *Exercice* :
  - Si je mets un -Xmx64M quel % de services tourne sans Out of Heap ?
  - Si je met un docker limit swarm / kube limits à 128M ? (pour la Heap + la non Heap) ?
- Peux tu en <= 30 minutes me dire
  - Combien de RAM prend toute ta stack en pleine charge
  - Combien de temps prend ta JVM pour arriver à température de prod

Pas d’actions claires sans idées claires

- Avant de chercher à réduire l’empreinte financière de Java
- Tu veux d’abord avoir une *idée* de ton TCO Total Cost of Software/Hardware Ownership
  - Sans évoquer le TJM, la QA, les moyens de production (locaux, matériel, télétravail)
- => IT & DevOps & FinOps to the rescue !
- *Graal va te donner de la frugalité au RUN* => étude de cas sur un livrable type

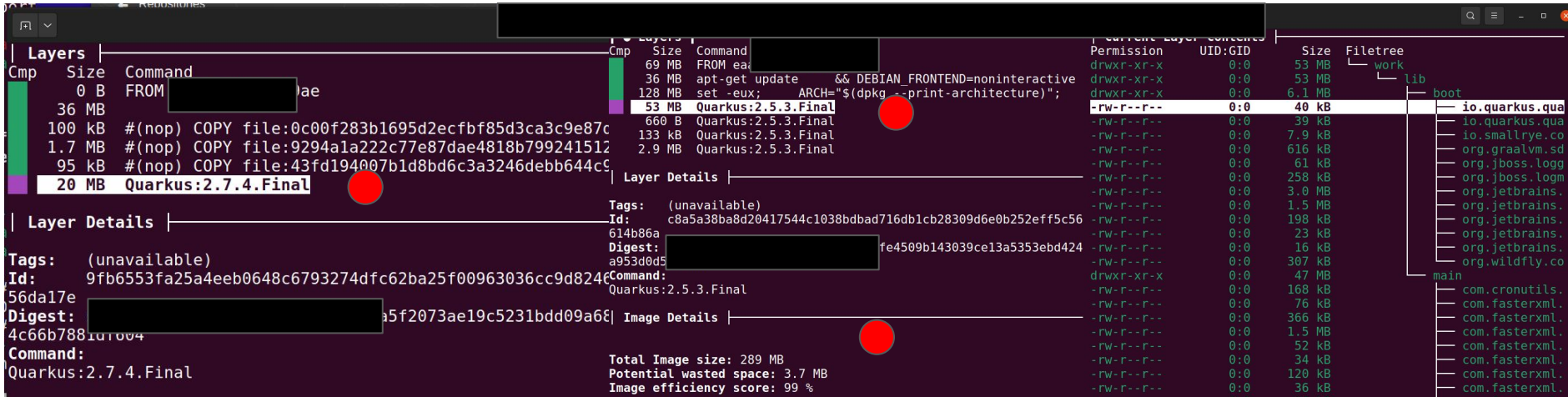


Des chiffres !

- ***Cas du livrable type actuel nommé XXX*** :
  - HTTP API REST + UI HTML Swagger + HTTP Client
  - Client/producer RabbitMQ
  - JDBC thin, pas d'ORM surprise
  - De la fault tolerance, des metrics tech et métier
  - Always available 24/7 sur multi timezones
  - Volume : milliers de requêtes/heure
    - Pas si micro… pas un monolithe non plus
    - Depuis Quarkus JVM => on le passe en Graal native image

Des chiffres !

- ***On part de*** : JVM + Debian + OpenJDK11 : **289MB** dont 55MB de jars, Quarkus treeshaké
- Vers applicatif XXX si buildé en graal natif : **90MB**, au lieu de 55MB de jars
  - De 90MB, on réduit à **20MB** si compressé - longuement… - via UPX
- Vers container si image native : **131MB** (RHEL microUBI + 90MB de XXX)
  - Réduite à **58MB** si compressée
- Vers boot : XXX native (powered by Quarkus 2.7.4.Final) started in **0.097**s. Listening on: http://0.0.0.0:8080



Des chiffres !

[redacted]

```
$ docker images
REPOSITORY                                  TAG           IMAGE ID       CREATED          SIZE
[redacted]                                  unspecified   9805[redacted] 4 hours ago      58.4MB
[redacted]                                  1.10.0        dac7[redacted] 5 hours ago      289MB
[redacted]                                  1.4           5fe3[redacted] 26 hours ago     277MB
[redacted]                                  1.5.0         058f[redacted] 9 days ago       293MB
```

- **Résultat : image de 289M à 58MB, démarre < 100ms versus 5 sec**
- Dans 200MB de RAM (*à tiède/tempéré, sans surcharge trafic de prod*)
- Quarkus, ok… et Spring Native beta ?
  - Se servent aussi d'UPX pour compresser
  - Leur image une Alpine MUSL, voire Debian, plus "souple" que RedHat
  - En Alpine très possiblement on est < 50MB (vu à Polytech Mtp)
- Pas mal... mais à quel prix, pour quels impacts ?
- … où sont les pièges ou coûts cachés ? **4 clés** =>

Division poids par 5
Division RAM par 2,5*
Division boot par 50

?

En natif, impact n°1 : la CI et le container produit

- Un runner qui compile en AOT natif au lieu d'une compile JIT sur serveur
  - Tu dois lui donner environ 6 gig de RAM car il va introspecter et shaker tout ton code
  - Avec un maximum de cores
  - Pour builder en 5-6 minutes (pour x64 sur laptop 4 cores 2.7GHz)
    - Avec UPX pour compresser, c'est encore plus lent, 15 minutes en level max
    - 10 levels de compression possibles, best around 7/10
- Créer une *petite* base image run et CI + native-image n'est pas cher
  - RedHat Quarkus et Spring Native ont des containers natifs minimalistes, circa 30MB
    - RedHat : UBI Universal Base Image
    - Spring : Paketo buildpacks (Alpine MUSL, Ubuntu, Debian)
  - *Il faut s'accorder avec votre équipe containers/infra/security*

En natif, impact n°2 : le run / le support N3

- Un livrable java devenu binaire natif qui explose et segfault en prod
  - Tu peux devoir sortir GDB pour debugger le crash
  - RedHat gentils ont upstreamé JFlightRecorder dans GraalVM mi 2021 :o
    - *Néanmoins* tu n'as pas de JVM *classique*, le binaire embarque un *substrat de VM : SubstrateVM …* sans JVMTI
    - … tu n'as pas toute la puissance des agents comme VisualVM, JProfiler, et JFR reste Preview - à ce stade, non supporté SLA
  - Tu n'as pas JMX : finies les métriques old school et le hot change
  - Tu n'as pas le GC G1, tu n'as que SerialGC ou Epsilon no-gc
    - G1 c'est quand tu as des grosses heap, peu microservices
  - Si tu payes GraalEE… tu as G1 et des System.gc() ou heap dumps faciles, Oracle sont pas idiots

En natif, impact n°3 : les perfs *au runtime*

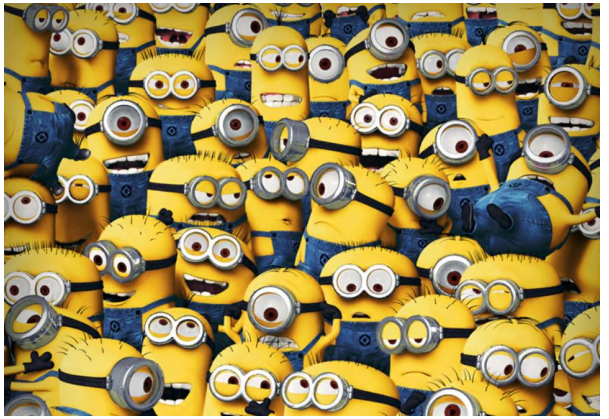
Vmax: 250km/h

Vmax: >330km/h

- Tu n’as plus de JIT - tu l’as en Graal JVM, pas en *natif*
  - Alors tu n’as plus le mécanisme d’apprentissage et de chauffe de la JVM
  - Un JIT est runtime, il sait
    - Apprendre des réentrées ou sorties de loops, optimiser : C1 rapide se profile et nourrit C2 lent
      - Ex: 100 appels de méthode foo() => C1, 100_000 => C2
    - Spéculer et (dé)optimiser
- L’AOT au build ne peut pas deviner les usages que les clients feront du code
  - C’est pas un JIT (ha!)
  - Tu as “perf” : https://www.loicmathieu.fr/wordpress/informatique/profiler-une-image-native-graalvm-avec-perf/
  - Si tu as GraalEE tu peux prendre une *photo* profiling de ces usages *en pleine charge*
  - Tu redonnes ensuite la photo au moment du rebuild natif dans ta CI
    - PGO : Profile Guided Optimization
    - https://docs.oracle.com/en/graalvm/enterprise/20/docs/reference-manual/native-image/PGO/ && https://fr.wikipedia.org/wiki/Optimisation_dirig%C3%A9e_par_les_profils

En natif, impact n°4 : l’effet rebond, l’effet frugal

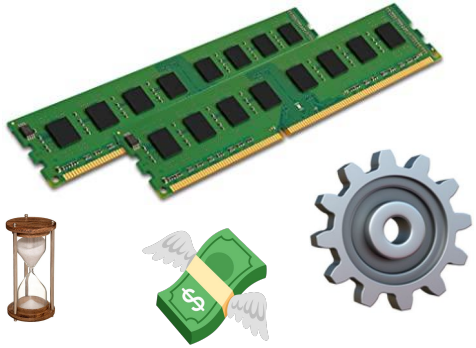
- Maintenant qu’on a -enfin- du Java léger
- Attention à la tentation de
  - Démultiplier les microservices
    - Surtout sur une techno avec Oracle majoritaire
    - En EE, Java SE subscription : $25/cpu/month
    - “No technical support”, licence Early Adopter
  - Mettre du Graal là où il n’y en a pas *besoin*
    - Certes hors natif son JIT est > HotSpot
- Action du levier financier, à la baisse
  - Diminuer *l’empreinte financière* d’une stack technique
  - Pour le bénéfice des actionnaires, et/ou des salariés ?



Pas forcés de faire natif frugal, JVM est toujours dispo

- Un GraalVM CE JVM dérive d'OpenJDK HotSpot
  - Quasi drop in replacement de JIT… sans G1
  - Il se substitue au compilo C2 via l'interface JVMCI
  - -XX:-UseJVMCICompiler si tu veux *ne pas utiliser graal* et *retrouver* Hotspot
- L'EE donne un meilleur JIT - un encore meilleur si tu payes
  - *For example, the compiler in GraalVM Enterprise includes 62 optimization phases, of which 27 are patented*
  - *https://medium.com/graalvm/enhanced-automated-vectorization-in-graalvm-76cd99925b6d.*
- CE ou EE ouvre les portes du polyglotte - la boîte de Pandore
  - Oui tu peux créer un objet dans un langage, et le retrouver dans l'autre
    - Idée pas neuve, mais le monde d'aujourd'hui c'est des boîtes multilanguages
    - Tu peux embed du Java8 dans du Java11, python/JS/R <> Kotlin/Scala
  - Une contrée à explorer avec du doliprane …

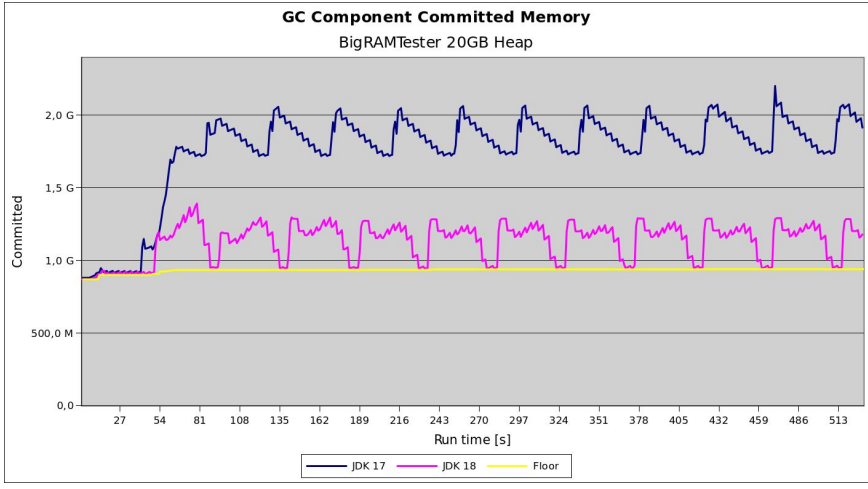
Vu ces impacts et chiffres, y aller ?



- *Oui mais*
  regarde des deux côtés avant de traverser la prod
- Assèche d'abord tes jars, ton container
  - Regarde d'abord ce qui est sous tes yeux
  - Puis regarde le macro vue d'avion / ton TCO
- Parle à tes infras & ops & FinOps & Support
  - Surtout si tu as JMX ou une run image custom (ex : banque auditée)
- Elis un service candidat : un existant migrable, où nouveau à venir
  - Et chiffre le : TJM les impacts
- C'est en majorité du Oracle, GraalEE a des brevets
  - Boite valorisée à 220B USD, plus que PepsiCo ; accepte leur modèle - si tu payes
  - Do not troll : they did not kill Java
  - Laisse au placard les fantômes de Mysql, Hudson, Solaris, LibreOffice, Android vs Google

Hors GraalVM, des efforts de RAM sont faits : JDK18

- autour de 30-40% de réduction de RAM *du GC G1*
- C’est que la RAM du GC
  - Un camion poubelle prend de la place, il fait au mieux
- C’est que pour G1
  - et pas Serial ; certaines app small heap default en Serial
- G1 n’est pas dans Graal CE gratuit
- Pose toi des questions si ton “micro” service requiert G1



Littérature

- Graal Heap Snapshotting & benchmarks versus V8
  - Oracle 2019
  - https://dl.acm.org/doi/10.1145/3360610
- “Direct Heap Snapshotting in the Java HotSpot VM: a Prototype Ludvig Janiuk”
  - Thèse avec une bonne section sur comment Graal gère la Heap en AOT
  - https://www.diva-portal.org/smash/get/diva2:1508220/FULLTEXT01.pdf
- GraalEE vs CE vs Hotspot, who runs kotlin faster ?
  - https://medium.com/javarevisited/graal-vs-c2-who-runs-kotlin-faster-82f03f1b11dd
- Le code source, toujours
  - https://github.com/oracle/graal
  - https://github.com/graalvm/mandrel
- Merci
- => Q&A