Kubernetes & la JVM

Alain Regnier

@altolabs

JMdoudouX

@jmdoudoux.bsky.social
@jmdoudoux

SCIAM

Java sur Kubernetes

Quand on pense « Java sur Kubernetes »

qui pense « lent à démarrer » ?

qui pense « problèmes de performances » ?

qui pense « gourmand en ressources » ?

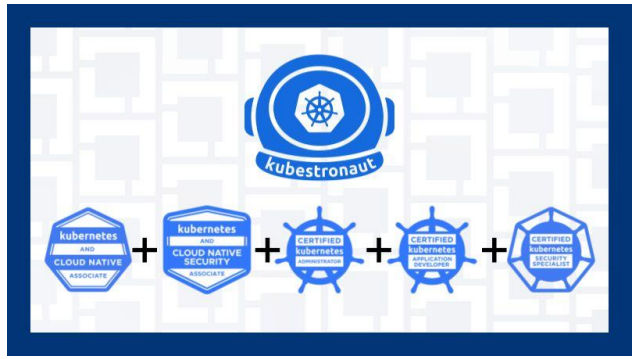
qui pense « complexité de l'environnement de développement » ?

Alain REGNIER

Directeur Technique
Pôle Kubernetes et DevOps
chez SCIAM

Experts

10 ans dans la Silicon Valley
Expert Kubernetes
GDE Cloud (Google Developer Expert)
Kubestronaut
Fondateur du meetup GDG Cloud Paris



SCIAM
Conseil, architecture, accompagnement, audit, formation...
Outils audit/supervision pour Kubernetes:
KuboScore et *KuboVisor*

Jean-Michel DOUDOUX

Directeur Technique & Formation
chez SCIAM

Java Champions

https://www.jmdoudoux.fr

@jmdoudoux.bsky.social

@jmdoudoux

Auteur d'un didacticiel depuis 2001
Diffusé sous licence GNU FDL
Développons en Java (4200 pages)

Co-fondateur du LOR'n'JUG

Roadmap

Kubernetes sur le poste de Dev

Spécificités de Kubernetes à connaître

Mécanismes internes de la JVM HotSpot

Exécution d'une App Java dans Kubernetes

Temps de démarrage et consommation de ressources

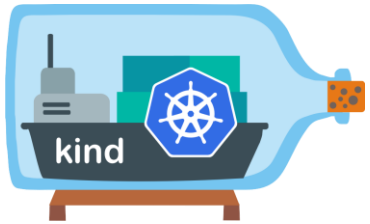
Kubernetes sur le poste de Dev

Kubernetes sur le poste de Dev

Avant : utilisation de docker ou docker-compose en local

Plusieurs solutions pour faire du kubernetes en local : minikube, k3s, kind…

Kind



outil permettant de créer des clusters Kubernetes en local

utilise des containers Docker pour les nœuds

supporte même les Ingress et Gateway avec *cloud-provider-kind*

Flox

FLOX

outil permettant de créer et isoler des environnements de travail

facilite l'utilisation de versions différentes d'outils, de configurations, de JDK…

reproductible avec approche déclarative, basé sur Nix

Kubernetes sur le poste de Dev - flox

```
// install system specific flox version


// create a work folder for environment kind-dev
$ cd flox
$ mkdir kind-dev
$ cd kind-dev


// initialize flox environment
$ flox init
⚡ Created environment 'kind-dev' (aarch64-darwin)
Next:
  $ flox search <package>    <- Search for a package
  $ flox install <package>   <- Install a package into an environment
  $ flox activate            <- Enter the environment
  $ flox edit                <- Add environment variables and shell hooks
  $ flox push                <- Use the environment from other machines or
                                share it with someone on FloxHub
```

Kubernetes sur le poste de Dev - flox

```
// search for kind in flox's list of available packages
$ flox search kind
kind                        Kubernetes IN Docker - local clusters for testing Kubernetes
(...)


// install useful tools
$ flox install kubectl kubernetes-helm jq
✓ 'kubectl', 'kubernetes-helm', 'jq' installed to environment 'kind-dev’


// check available kind versions
$ flox show kind
kind - Kubernetes IN Docker - local clusters for testing Kubernetes
Catalog: nixpkgs
Latest:  kind@0.31.0
License: Apache-2.0
Outputs: out* (* installed by default)
Systems: x86_64-linux, aarch64-darwin, aarch64-linux, x86_64-darwin
Other versions:
    kind@0.31.0
    kind@0.30.0
    kind@0.29.0
(...)
```

Kubernetes sur le poste de Dev - kind

```
// install specific kind version
$ flox install kind@0.31.0
✓ 'kind' installed to environment 'kind-dev'


// check kind version on current system
$ kind version
zsh: command not found: kind


// check kubectl version on current system
$ kubectl version
Client Version: v1.30.6
Kustomize Version: v5.0.4-0.20230601165947-6ce0bf390ce3
The connection to the server localhost:8080 was refused - did you specify the
right host or port?
```

Kubernetes sur le poste de Dev - kind

```
// activate environment
$ flox activate
✓ You are now using the environment 'kind-dev'
To stop using this environment, type 'exit'
flox [kind-dev] ➜  kind-dev


// check kind version in flox environment
$ kind version
kind v0.31.0 go1.25.7 darwin/arm64


// check kubectl version in flox environment
$ kubectl version
Client Version: v1.35.2
Kustomize Version: v5.7.1
```

Kubernetes sur le poste de Dev - kind

```
// prepare KUBECONFIG to work locally
$ export KUBECONFIG=$PWD/kubeconfig.yaml
$ echo $KUBECONFIG
/Users/alain/data/flox/kind-dev/kubeconfig.yaml


// make permanent change to KUBECONFIG to work with local version
$ flox edit
   [vars]
   KUBECONFIG = "/Users/alain/data/flox/kind-dev/kubeconfig.yaml"


$ exit
$ flox activate
$ echo $KUBECONFIG
/Users/alain/data/flox/kind-dev/kubeconfig.yaml
```

Kubernetes sur le poste de Dev - cluster

```
// create a basic Kubernetes cluster
$ kind create cluster --name smallk8s
Creating cluster "smallk8s" ...
 ✓ Ensuring node image (kindest/node:v1.35.0) 🖼
 ✓ Preparing nodes 📦
 ✓ Writing configuration 📜
 ✓ Starting control-plane 🕹️
 ✓ Installing CNI 🔌
 ✓ Installing StorageClass 💾
Set kubectl context to "kind-smallk8s"
You can now use your cluster with:
kubectl cluster-info --context kind-smallk8s
Have a nice day! 👋


// check kubeconfig.yaml update content
$ vi kubeconfig.yaml
   apiVersion: v1
   clusters:
   - cluster:
       certificate-authority-data: LS0tLS1CRUdJ
   (...)
```

Kubernetes sur le poste de Dev - cluster

```
// check available images for nodes at https://hub.docker.com/r/kindest/node/
// or use default


// prepare a more advanced cluster with 1 master and 2 workers and specific kubernetes version
$ vi devcluster.yaml
   kind: Cluster
   apiVersion: kind.x-k8s.io/v1alpha4
   networking:
     ipFamily: ipv4
   name: devk8s
   nodes:
     - role: control-plane
       image: kindest/node:v1.35.1
     - role: worker
       image: kindest/node:v1.35.1
     - role: worker
       image: kindest/node:v1.35.1
```

Kubernetes sur le poste de Dev - cluster

```
// create advanced dev cluster
$ kind create cluster --config devcluster.yaml
Creating cluster "devk8s" ...
 ✓ Ensuring node image (kindest/node:v1.35.1) 🖼
 ✓ Preparing nodes 📦 📦 📦
 ✓ Writing configuration 📜
 ✓ Starting control-plane 🕹️
 ✓ Installing CNI 🔌
 ✓ Installing StorageClass 💾
 ✓ Joining worker nodes 🚜
Set kubectl context to "kind-devk8s"
You can now use your cluster with:
kubectl cluster-info --context kind-devk8s
Not sure what to do next? 😅  Check out
https://kind.sigs.k8s.io/docs/user/quick-start/


// check created clusters
$ kind get clusters
devk8s
smallk8s
```

Kubernetes sur le poste de Dev - cluster

```
// check nodes on smallk8s
$ kubectl get nodes --context kind-smallk8s
NAME                       STATUS   ROLES           AGE   VERSION
smallk8s-control-plane   Ready    control-plane   14m   v1.35.0


// check nodes on devk8s
$ kubectl get nodes --context kind-devk8s
NAME                     STATUS   ROLES           AGE   VERSION
devk8s-control-plane   Ready    control-plane   87s   v1.35.1
devk8s-worker          Ready    <none>          78s   v1.35.1
devk8s-worker2         Ready    <none>          78s   v1.35.1


// create kdev alias to make work easier
$ alias kdev="kubectl --context kind-devk8s"
```

Kubernetes sur le poste de Dev - cluster

```
// check cluster Pods
$ kdev get pods -A
NAMESPACE          NAME                                          READY STATUS  RESTARTS AGE
kube-system        coredns-7d764666f9-bs65f                      1/1   Running 0        108m
kube-system        coredns-7d764666f9-wkpnp                      1/1   Running 0        108m
kube-system        etcd-devk8s-control-plane                     1/1   Running 0        108m
kube-system        kindnet-2jgq8                                 1/1   Running 0        108m
kube-system        kindnet-fp27n                                 1/1   Running 0        108m
kube-system        kindnet-mc2f6                                 1/1   Running 0        108m
kube-system        kube-apiserver-devk8s-control-plane           1/1   Running 0        108m
kube-system        kube-controller-manager-devk8s-control-plane  1/1   Running 0        108m
kube-system        kube-proxy-q5f84                              1/1   Running 0        108m
kube-system        kube-proxy-tgwml                              1/1   Running 0        108m
kube-system        kube-proxy-vn28b                              1/1   Running 0        108m
kube-system        kube-scheduler-devk8s-control-plane           1/1   Running 0        108m
local-path-storage local-path-provisioner-84669cbbb8-2bjpk       1/1   Running 0        108m
```

Kubernetes sur le poste de Dev - cluster

```
// test deployment on cluster
$ kdev run test --restart=Never --rm -it --image=busybox -- ls -al
total 52
drwxr-xr-x    1 root     root          4096 Apr 23 16:04 .
drwxr-xr-x    1 root     root          4096 Apr 23 16:04 ..
drwxr-xr-x    2 root     root         12288 Sep 26  2024 bin
drwxr-xr-x    5 root     root           380 Apr 23 16:04 dev
drwxr-xr-x    1 root     root          4096 Apr 23 16:04 etc
drwxr-xr-x    2 nobody   nobody        4096 Sep 26  2024 home
drwxr-xr-x    2 root     root          4096 Sep 26  2024 lib
lrwxrwxrwx    1 root     root             3 Sep 26  2024 lib64 -> lib
dr-xr-xr-x  340 root     root             0 Apr 23 16:04 proc
-rw-r--r--    1 root     root            37 Apr 23 16:04 product_uuid
drwx------    2 root     root          4096 Sep 26  2024 root
dr-xr-xr-x   11 root     root             0 Apr 23 16:04 sys
drwxrwxrwt    2 root     root          4096 Sep 26  2024 tmp
drwxr-xr-x    4 root     root          4096 Sep 26  2024 usr
drwxr-xr-x    1 root     root          4096 Apr 23 16:04 var
pod "test" deleted from default namespace
```

Construction d'images

Bonnes pratiques:

Réduire la taille des images autant que possible

Pas besoin d'embarquer une distribution linux complète dans son image de base

Favoriser les multi-stage builds

- image de base avec sdk et outils pour le build
- image de base légère comme source de l'image finale

D'autres solutions que « docker build » existent pour construire son image:

Kaniko

- construire une image à partir d'un Dockerfile dans un container (CI/CD, cluster Kubernetes)
  projet anciennement maintenu par Google, maintenant par Chainguard

Jib

- construire une image de container pour son application Java directement via un plugin Maven ou Gradle sans Dockerfile

Buildpacks

- construire une image de container sans Dockerfile via détection du code source

Spécificités de Kubernetes à connaître

Probes

Permettent de suivre le cycle de vie des containers

Plusieurs types (HTTP [2xx,3xx], Command, TCP), paramètres (Period, Threshold)

Liveness : Permet à Kubernetes de savoir quand l'application est en erreur et doit être redémarrée

```
spec:
  containers: (...)
    livenessProbe:
      httpGet:
        path: /healthy
```

Readiness : Permet à Kubernetes de savoir quand l'application est prête à recevoir des requêtes pour lui envoyer du trafic

Startup : Permet à Kubernetes de savoir quand l'application est prête pour démarrer les autres Probes

Par défaut, Kubernetes commence à envoyer du trafic à un Pod quand tous ses containers ont démarré… mais l'application n'est peut-être pas encore prête!

livenessProbe et *readinessProbe* ont des rôles différents… et ne doivent donc pas être identiques

Spécifier *initialDelaySeconds* pour indiquer d'attendre un certain temps avant la première Probe

Ne pas générer de logs et éviter de contacter d'autres services trop fréquemment (ex. DB)

Mécanismes de gestion de ressources

Types de ressources

cpu, memoire, ephemeral-storage...

Request

ce qu'un container est garant d'avoir en termes de ressources

utilisé par le *Scheduler* pour savoir ou déployer le Pod

les Pods sans *Request* sont les premiers candidats pour être supprimés même si ils sont en bonne santé (*BestEffort* vs *Guaranteed* QoS)

Limit

le maximum de ressources qu'un container peut utiliser avant d'être restreint

si dépassement de la *CPU Limit*: ralentissement artificiel (*throttling*)

si dépassement de la *Memory Limit*: suppression du container

Application de demo

Petite application Spring Boot

Exposant 2 services:

/chuck

fait un appel http à une API externe simple: https://api.chucknorris.io

renvoie un fait sur ***Chuck Norris*** (RIP)

/k8s

fait un appel à la **Kubernetes API** d'un cluster Kubernetes

utilise le Kubernetes Java Client officiel

renvoie la liste des Pods présents sur le cluster

Spring Boot 3 / Java 21/ Kubernetes client 26.0.0

DemoController.java

```java
@GetMapping("/chuck")
public String chuck() {
    String chuck=service.getChuckNorrisQuote();
    LOG.info("chuck of the day: {}", chuck);
    return chuck;
}

@GetMapping("/k8s")
public String k8s() {
    StringBuilder sb=new StringBuilder();
    List<DemoService.PodInfo> pods=null;
    try {
        pods=service.getAllPods();
    } catch (Exception e) {
        throw new RuntimeException(e);
    }
    sb.append("Pods found on cluster with provided KubeConfig:\n");
    for (DemoService.PodInfo info:pods) {
        sb.append(String.format("%-20.20s %-30.30s %12.12s\n",
                info.namespace(),info.name(),info.status()));
    }
    return sb.toString();
}
```

DemoService.java

```java
@Service
@RegisterReflectionForBinding(DemoService.Quote.class)
public class DemoService {

    public final String CHUCK_URL="https://api.chucknorris.io";
    @JsonIgnoreProperties(ignoreUnknown = true)
    public static record Quote (
        String id,
        String value
    ) {}


    public String getChuckNorrisQuote() {
        String result="n/a";
        RestClient restClient = RestClient.builder()
                .baseUrl(CHUCK_URL)
                .build();
        Quote quote=restClient.get()
                .uri("/jokes/random?category=dev")
                .retrieve()
                .body(Quote.class);
        result=quote.value;
        System.out.println("received quote: "+result);
        return result;
    }
```

DemoService.java (2)

```
    public static record PodInfo(String name, String namespace, String status)  {}
    public List<PodInfo> getAllPods() throws Exception {
        ApiClient client = Config.defaultClient();
        Configuration.setDefaultApiClient(client);
        CoreV1Api api = new CoreV1Api();
        V1PodList list = api.listPodForAllNamespaces()
                .execute();

        List<PodInfo> pods=list.getItems().stream()
                .map(pod -> new PodInfo(
                        pod.getMetadata().getName(),
                        pod.getMetadata().getNamespace(),
                        pod.getStatus().getPhase()
                ))
                .collect(Collectors.toList());
        System.out.println("found Pods: "+pods.size());
        return pods;
    }

}
```

DemoApplication.java

```
public class DemoHintsRegistrar implements RuntimeHintsRegistrar {


@SpringBootApplication
@ImportRuntimeHints(DemoHintsRegistrar.class)
public class DemoApplication {
    private static final Logger LOG = LoggerFactory.getLogger(DemoApplication.class);

    public static void main(String[] args) {
        LOG.info("starting...");
        System.setProperty("spring.main.banner-mode", "off");
        String kubeConfig = System.getenv("KUBECONFIG");
        if (kubeConfig != null) {
            LOG.info("Found KUBECONFIG: {}", kubeConfig);
        } else {
            LOG.warn("KUBECONFIG environment variable is NOT set.");
        }

        SpringApplication.run(DemoApplication.class, args);
    }
}
```

Application de demo

```
// build jar
$ ./gradlew build


// build container image
$ docker build -t areg/demosb:0.1 .


// use “kind load docker-image” to make image available to kind cluster
$ kind load docker-image areg/demosb:0.1 -n devk8s
Image: "areg/demosb:0.1" with ID
"sha256:13fd0112c4f51a5d40010fb0910dd6516ecb8c380ffc969bfcb9d619fbb3f25d" not
yet present on node "devk8s-worker", loading...
Image: "areg/demosb:0.1" with ID
"sha256:13fd0112c4f51a5d40010fb0910dd6516ecb8c380ffc969bfcb9d619fbb3f25d" not
yet present on node "devk8s-worker2", loading...
Image: "areg/demosb:0.1" with ID
"sha256:13fd0112c4f51a5d40010fb0910dd6516ecb8c380ffc969bfcb9d619fbb3f25d" not
yet present on node "devk8s-control-plane", loading...
```

Test de l'application

```
$ curl http://localhost:8080/chuck
There is no Esc key on Chuck Norris' keyboard, because no one escapes Chuck Norris.

$ curl http://localhost:8080/chuck
Chuck Norris's first program was kill -9.

$ curl http://localhost:8080/chuck
When Chuck Norris is web surfing websites get the message "Warning: Internet Explorer has deemed this user to
be malicious or dangerous. Proceed?"

$ curl http://localhost:8080/k8s
Pods found on cluster with provided KubeConfig:
gke-managed-cim      kube-state-metrics-0                Running
gmp-system           collector-jszz5                     Running
gmp-system           collector-s4jrw                     Running
gmp-system           gmp-operator-545898dd4d-p7gp6       Running
kube-system          anetd-66hw2                         Running
kube-system          anetd-dz92r                         Running
kube-system          antrea-controller-horizontal-a      Running
kube-system          event-exporter-gke-5777bb47b8-      Running
kube-system          fluentbit-gke-gf4ks                 Running
kube-system          fluentbit-gke-zkhf9                 Running
kube-system          gke-metrics-agent-pht8d             Running
kube-system          gke-metrics-agent-s2csq             Running
kube-system          konnectivity-agent-autoscaler-      Running
(...)
```

Ressources Request / Limit

```
// prepare deployment yaml
$ vi k8s/demosb-deploy.yaml
   apiVersion: apps/v1
   kind: Deployment
   metadata:
     name: demosb
     (...)
         containers:
           - name: app
             image: areg/demosb:0.1
             ports:
               - containerPort: 8080
             resources:
               requests:
                 memory: "512Mi"
                 cpu: "50m"
               limits:
                 memory: "1Gi"
                 cpu: "200m"
```

Ressources Request / Limit

```
// check logs when cpu limit is 200m
$ kubectl logs -f pod/demosb-7f978c48f8-dj9nj
14:51:57.095 [main] INFO io.kubolabs.demosb.DemoApplication -- starting...
14:51:57.191 [main] WARN io.kubolabs.demosb.DemoApplication -- KUBECONFIG environment variable is NOT set.
2026-04-10T14:52:08.694Z  INFO 1 --- [demosb] [     main] io.kubolabs.demosb.DemoApplication       : Starting DemoApplication v0.0.1-SNAPSHOT using Java 21.0.10 with PID 1
(/app/app.jar started by springuser in /app)
2026-04-10T14:52:08.771Z  INFO 1 --- [demosb] [     main] io.kubolabs.demosb.DemoApplication       : No active profile set, falling back to 1 default profile: "default"
2026-04-10T14:52:25.197Z  INFO 1 --- [demosb] [     main] o.s.boot.tomcat.TomcatWebServer          : Tomcat initialized with port 8080 (http)
2026-04-10T14:52:25.395Z  INFO 1 --- [demosb] [     main] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
2026-04-10T14:52:25.396Z  INFO 1 --- [demosb] [     main] o.apache.catalina.core.StandardEngine    : Starting Servlet engine: [Apache Tomcat/11.0.20]
2026-04-10T14:52:26.991Z  INFO 1 --- [demosb] [     main] b.w.c.s.WebApplicationContextInitializer : Root WebApplicationContext: initialization completed in 17183 ms
2026-04-10T14:52:37.376Z  INFO 1 --- [demosb] [     main] o.s.boot.tomcat.TomcatWebServer          : Tomcat started on port 8080 (http) with context path '/'
2026-04-10T14:52:37.691Z  INFO 1 --- [demosb] [     main] io.kubolabs.demosb.DemoApplication       : Started DemoApplication in 37.396 seconds (process running for 48.08)
// ==> 37.396 seconds

// check logs when cpu limit is 1000m
$ kubectl logs -f pod/demosb-6877f79475-6pxhl
14:57:31.582 [main] INFO io.kubolabs.demosb.DemoApplication -- starting...
14:57:31.585 [main] WARN io.kubolabs.demosb.DemoApplication -- KUBECONFIG environment variable is NOT set.
2026-04-10T14:57:32.218Z  INFO 1 --- [demosb] [     main] io.kubolabs.demosb.DemoApplication       : Starting DemoApplication v0.0.1-SNAPSHOT using Java 21.0.10 with PID 1
(/app/app.jar started by springuser in /app)
2026-04-10T14:57:32.219Z  INFO 1 --- [demosb] [     main] io.kubolabs.demosb.DemoApplication       : No active profile set, falling back to 1 default profile: "default"
2026-04-10T14:57:33.112Z  INFO 1 --- [demosb] [     main] o.s.boot.tomcat.TomcatWebServer          : Tomcat initialized with port 8080 (http)
2026-04-10T14:57:33.119Z  INFO 1 --- [demosb] [     main] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
2026-04-10T14:57:33.119Z  INFO 1 --- [demosb] [     main] o.apache.catalina.core.StandardEngine    : Starting Servlet engine: [Apache Tomcat/11.0.20]
2026-04-10T14:57:33.235Z  INFO 1 --- [demosb] [     main] b.w.c.s.WebApplicationContextInitializer : Root WebApplicationContext: initialization completed in 929 ms
2026-04-10T14:57:33.792Z  INFO 1 --- [demosb] [     main] o.s.boot.tomcat.TomcatWebServer          : Tomcat started on port 8080 (http) with context path '/'
2026-04-10T14:57:33.797Z  INFO 1 --- [demosb] [     main] io.kubolabs.demosb.DemoApplication       : Started DemoApplication in 2.008 seconds (process running for 2.754)
// ==> 2.008 seconds
```

Graceful Shutdown

Permet d'éviter certains problèmes quand un container doit être arrêté (via l'API ou par le système)

- Perte de données
- Erreurs de type *Connection Refused* quand le conteneur reçoit encore des requêtes
- Transactions interrompues

Déroulement:

- Passage du Pod en *Terminating*
- Envoi d'un **SIGTERM** au process 1 de chaque container (à attraper)

ET

- Suppression du Pod de la liste des destinations du service associé
- Attente de 30 secondes (*terminationGracePeriodSeconds*)
- Envoi d'un **SIGKILL** aux containers qui tournent toujours

Possibilité d'ajouter un script à exécuter avec le *preStop* hook

Mécanismes internes de la JVM HotSpot

Le JIT (Just-In-Time Compiler)

Transforme le bytecode en code machine natif à l'exécution pour améliorer les perfs à chaud

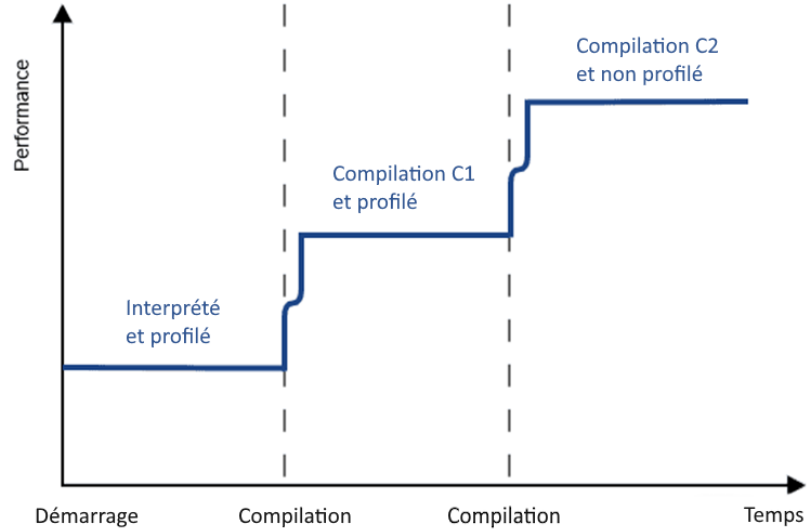
Depuis JDK7, Tiered compilation combine plusieurs fonctionnalités
pour un démarrage rapide et une optimisation maximale à chaud

Mais requière un temps de préchauffage (warmup) et de la CPU/Mémoire

1) Un interpréteur qui exécute le byte code
Avec profiling qui collecte des métriques

2) Compilateur C1 (client compiler)
code natif rapide faiblement optimisé
des hots methods (+/- 10 000 invocations)

3) Compilateur C2 (server compiler)
produit du code natif avec optimisations agressives
mais plus lentes à produire



Le JIT améliore grandement les performances

Mais consomme des ressources (CPU, mémoire) au détriment de l'application

Et cela à chaque redémarrage de la JVM

Le JIT (Just-In-Time Compiler)

La compilation est réalisée au niveau méthode

Les GC modernes (G1, ZGC, Shenandoah…) ont besoin du JIT
    Injections de barrières mémoire dans le code, réécriture de pointeurs…

Le code natif est stocké dans le Code Cache

    Taille fixe configurable avec `-XX:ReservedCodeCacheSize`

Option pour tracer l'activité avec JDK 9+ : `-Xlog:jit+compilation=info|debug`

Utilisation de JITWatch pour visualiser les logs

Ou de JFR pour capturer des événements dans un enregistrement

La gestion de la mémoire

C’est le rôle d’un ramasse-miettes (garbage collector) :

- Allouer l’espace mémoire nécessaire aux objets à leur création
- Récupérer et recycler la mémoire des objets qui ne sont plus utilisés
  - Ce sont des objets qui ne sont plus référencés

Avantages

- Le code est plus simple : pas de libération explicitement la mémoire
- Amélioration de la productivité et de la fiabilité comparée par à C ou C++

Inconvénients

- Des traitements automatiques qui consomment des ressources (CPU, mémoire)
- N’élimine pas tous les problèmes
  - Fuites de mémoire et OutOfMemoryError tout de même possibles

La gestion de la mémoire

La JVM HotSpot propose plusieurs implémentations selon la version :

GC	Disponibilité	Option pour activation
Serial GC	depuis JDK 1.0	-XX:+UseSerialGC
Parallel GC	depuis JDK 1.4.1	-XX:+UseParallelGC
CMS GC	du JDK 1.4.1 au JDK 13	-XX:+UseConcMarkSweepGC
G1 GC	depuis JDK 1.7	-XX:+UseG1GC
Shenandoah GC	depuis JDK 15	-XX:+UseShenandoahGC
Z GC	depuis JDK 15	-XX:+UseZGC
Epsilon	depuis JDK 11 (expérimental)	-XX:+UnlockExperimentalVMOptions -XX:+UseEpsilonGC

Ils évoluent et s'améliorent au fur et à mesure des versions du JDK

La gestion de la mémoire

Le ramasse-miettes par défaut est choisi par la JVM, depuis JDK 1.5

Parfois nécessaire d'en utiliser un autre

Le choix d'un GC est toujours un compromis entre trois facteurs

- Throughput : le temps pour l'exécution de l'application
- Empreinte mémoire (footprint) et CPU : utilisés pour les activités du GC
- Latence : les temps de pauses induites par les traitements du GC

Lorsque qu'un facteur est privilégié par un GC

- C'est au détriment d'un ou des deux autres

Il n'est pas possible d'avoir les 3

Il faut donc choisir selon le facteur privilégié

- Et accepter de dégrader les autres facteurs

GC	Facteur privilégié
Serial GC	Mémoire
Parallel GC	Thoughput
CMS GC	Latence
G1 GC	Équilibre Troughput / latence
Shenandoah GC	Latence
Z GC	Latence

Les ergonomics de la JVM HotSpot

Mécanismes d'auto-configuration par défaut de la JVM

- Via certaines heuristiques (depuis JDK 1.5)

Selon certaines caractéristiques physiques de l'environnement d'exécution

- CPU, RAM

Pour fournir de bonnes performances par défaut dans la majorité des cas

Choix du mode d'exécution de la JVM

- Mode server : OS 64 bits ET au moins cœurs CPU ET au moins 2Gi
  - optimisations JIT plus agressives
- Mode client sinon

Le mode est indiqué dans la version

```
C:\java>java -version
openjdk version "25" 2025-09-16
OpenJDK Runtime Environment (build 25+36-3489)
OpenJDK 64-Bit Server VM (build 25+36-3489, mixed mode, sharing)
```

Les ergonomics de la JVM HotSpot

Il est possible de forcer la configuration avec les options de démarrage

Obtenir la configuration

```
java -XX:+PrintFlagsFinal -version
```

Affiche toutes les options internes de la JVM avec leur valeur finale
y compris celles définies automatiquement par les ergonomics

{ergonomic} → valeur choisie automatiquement par la JVM (à partir du JDK 9)

```
java -XX:+PrintFlagsFinal -version | grep ergonomic
…
     uint ConcGCThreads                = 3                                   {product} {ergonomic}
    uintx GCDrainStackTargetSize       = 64                                  {product} {ergonomic}
   size_t InitialHeapSize              = 532676608                           {product} {ergonomic}
   size_t MaxHeapSize                  = 8503951360                          {product} {ergonomic}
   size_t MaxNewSize                   = 5102370816                          {product} {ergonomic}
    uintx ReservedCodeCacheSize        = 251658240                        {pd product} {ergonomic}
     bool UseCompressedClassPointers   = true                           {lp64_product} {ergonomic}
     bool UseCompressedOops            = true                           {lp64_product} {ergonomic}
     bool UseG1GC                      = true                                {product} {ergonomic}
…
```

Les ergonomics de la JVM HotSpot

A partir du JDK 11, utilisation de l'unified logging

Obtenir certaines valeurs facilement lisibles avec `java -Xlog:ergo*=info,gc* -version`

JDK 11

```
[0.007s][info   ][gc,heap] Heap region size: 2M
[0.016s][info   ][gc     ] Using G1
[0.017s][info   ][gc,heap,coops] Heap address:
0x0000000605200000, size: 8110 MB, Compressed Oops mode: Zero
based, Oop shift amount: 3
openjdk version "11.0.19" 2023-04-18
```

Amélioré dans le JDK 15

```
[0.006s][info][gc] Using G1
[0.009s][info][gc,init] Version: 15.0.2+7 (release)
[0.009s][info][gc,init] CPUs: 16 total, 16 available
[0.010s][info][gc,init] Memory: 32435M
[0.010s][info][gc,init] Large Page Support: Disabled
[0.010s][info][gc,init] NUMA Support: Disabled
[0.011s][info][gc,init] Compressed Oops: Enabled (Zero based)
[0.011s][info][gc,init] Heap Region Size: 4M
[0.011s][info][gc,init] Heap Min Capacity: 8M
[0.012s][info][gc,init] Heap Initial Capacity: 508M
[0.012s][info][gc,init] Heap Max Capacity: 8112M
[0.013s][info][gc,init] Pre-touch: Disabled
[0.014s][info][gc,init] Parallel Workers: 13
[0.014s][info][gc,init] Concurrent Workers: 3
[0.015s][info][gc,init] Concurrent Refinement Workers: 13
[0.015s][info][gc,init] Periodic GC: Disabled
[0.016s][info][gc,metaspace] CDS archive(s) mapped at:
[0x0000000800000000-0x0000000800b50000-0x0000000800b50000),
size 11862016, SharedBaseAddress: 0x0000000800000000,
ArchiveRelocationMode: 0.
openjdk version "15.0.2" 2021-01-19
```

Les ergonomics de la JVM HotSpot

Dimensionnement automatique de la taille du heap

La JVM choisit une taille basée sur une fraction de la mémoire physique

Varie parfois légèrement selon le JDK, mais généralement :

InitialHeapSize : 1/64 de la RAM totale

MaxHeapSize : 1/4 de la RAM totale

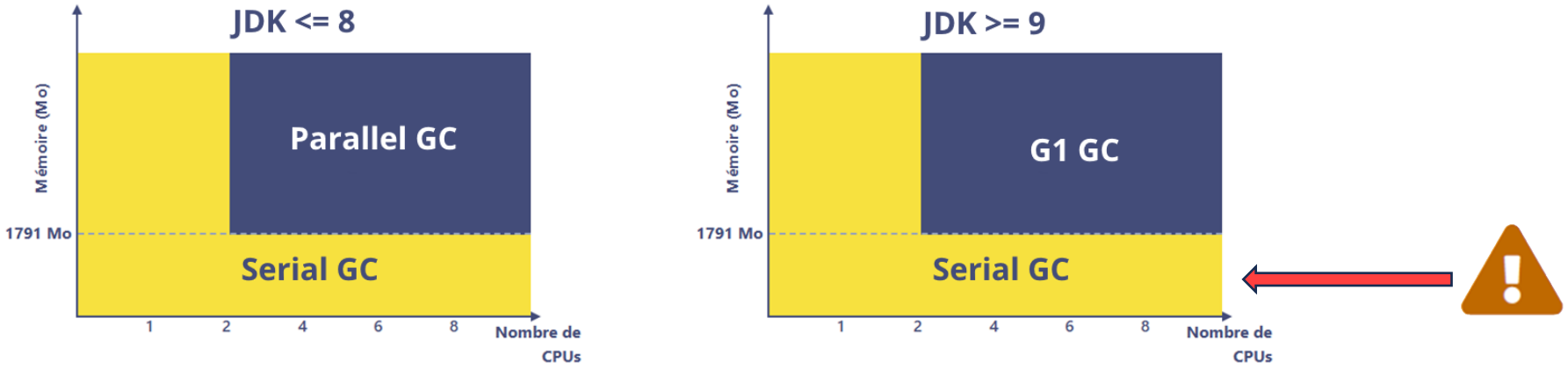
Pour connaitre les valeurs :

```
java -XX:+PrintFlagsFinal -version | grep -E "[Initial|Min|Max]HeapSize"
   size_t InitialHeapSize          = 532676608               {product} {ergonomic}
   size_t MaxHeapSize              = 8506048512              {product} {ergonomic}
   size_t MinHeapSize              = 8388608                 {product} {ergonomic}
```

Modifiable avec les options -Xms et -Xmx

Les ergonomics de la JVM HotSpot

Choix du ramasse-miettes selon la version et le mode de la JVM



Pour connaitre le GC utilisé

```
java -XX:+PrintFlagsFinal –version | grep -E "Use(G1|Z|Parallel|Serial|CMS|ConcMarkSweep|Shenandoah)GC"
     bool UseG1GC              = true              {product} {ergonomic}
     bool UseParallelGC        = false             {product} {default}
     bool UseSerialGC          = false             {product} {default}
     bool UseShenandoahGC      = false             {product} {default}
     bool UseZGC               = false             {product} {default}
```

Modification possible via l'option dédiée pour chaque GC

Les ergonomics de la JVM HotSpot

Nombre de threads et taille des pools de threads

Selon le nombre de cœurs CPU déterminé (physique ou limites du conteneur)

Détermine la valeur retournée par Runtime.getRuntime().availableProcessors()

Défini la Taille par défaut du pool ForkJoinPool commun

Modifiable avec `-Djava.util.concurrent.ForkJoinPool.common.parallelism=N`

Depuis le JDK21, défini la taille du pool de carrier threads pour les Thread Virtuels

Modifiable avec
```
-Djdk.virtualThreadScheduler.parallelism=N
-Djdk.virtualThreadScheduler.maxPoolSize=M
```

Selon l'OS, l'architecture CPU et le mode de la JVM

Défini la taille des stacks des threads

Modifiable avec –Xss, exemple : `-Xss2m`

Nombre de threads utilisé par les traitements parallèles et/ou concurrents des GC

Modifiable avec `-XX:ParallelGCThreads=N et -XX:ConcGCThreads=M`

Exécution d’une app Java dans Kubernetes

Les ergonomics et les conteneurs

Avant Java 10 :

JVM obtient ces valeurs du système hôte, ne détecte pas son exécution dans un conteneur

Ce qui peut engendrer un comportement erratique
Puisqu'elle ne prend pas en compte les limites

Solution utiliser l'option -Xmx en adéquation avec la memory limit

A partir de Java 10 avec backport en Java 8u191:

La JVM détecte les limites mémoire et CPU définies dans cgroup
Et les utilise dans les ergonomics

Nouvelles options :
-XX:InitialRAMPercentage
-XX:MaxRAMPercentage : généralement entre 75% et 90 selon la taille de la RAM
-XX:MinRAMPercentage
-XX:ActiveProcessorCount=n
UserContainerSupport activée par défaut sous Linux

JDK 18 support de cgroup v2

Les Probes

Support par Spring boot

Exposition en HTTP via l'Actuator
"/actuator/health/liveness"
et "/actuator/health/readiness"

Le endpoint health est activé et exposé par défaut

Les probes sont désactivées par défaut

```
management.endpoint.health.probes.enabled=true
management.health.livenessState.enabled=true
management.health.readinessState.enabled=true
```

Configuration dans deployment de Kubernetes

```
containers:
  - name: mon-app-spring
    image: mon-app-image:latest
    ports:
      - containerPort: 8080

    # Readiness probe
    readinessProbe:
      httpGet:
        path: /actuator/health/readiness
        port: 8080
      initialDelaySeconds: 5
      periodSeconds: 5

    # Liveness probe
    livenessProbe:
      httpGet:
        path: /actuator/health/liveness
        port: 8080
      initialDelaySeconds: 10
      periodSeconds: 10
```

Les Probes

Support par Quarkus

Avec l'extension smallrye-health
Exposition en HTTP via

```
quarkus.smallrye-health.root-path=/q/health
```

Endpoint sonde readiness : /q/health/ready

Enpoint sonde liveness : /q/health/live

Configuration dans deployment de Kubernetes

```
containers:
  - name: mon-app-quarkus
    image: mon-app-image:latest
    ports:
      - containerPort: 8080

    # Readiness probe
    readinessProbe:
      httpGet:
        path: /q/health/ready
        port: 8080
      initialDelaySeconds: 5
      periodSeconds: 5

    # Liveness probe
    livenessProbe:
      httpGet:
        path: /q/health/live
        port: 8080
      initialDelaySeconds: 10
      periodSeconds: 10
```

Gracefull shutdown

Support par Quarkus

par configuration de propriétés

```
quarkus.shutdown.delay-enabled=true
quarkus.shutdown.delay=10S              // délai avant de commencer le shutdown (ms, S ou M)
quarkus.shutdown.timeout=20S            // temps max pout terminer l'arrêt (ms, S ou M)
```

Support par Spring boot par configuration de propriétés

```
server.shutdown=graceful
spring.lifecycle.timeout-per-shutdown-phase=20s
```

Pas de délai avant de commencer le shutdown
Utiliser preStop dans le fichier deployment

```
lifecycle:
  preStop:
    exec:
      command: ["sh", "-c", "sleep 10"]
```

Le besoin en pic CPU de la JVM

La JVM a besoin de CPU pour les traitements de l'application

- Selon la charge et/ou le volume de données à traiter

Mais elle a aussi parfois besoin de CPU pour certains de ces traitements

- Au démarrage
  - Chargement de milliers de classes, création d'instances, …
- Traitements de certaines étapes du GC
  - La réduction des temps de pause requière de la CPU
- JIT

Le manque de CPU n'induit que des problèmes de performances

Vérifier à minima dans le profiling/monitoring :

- le CPU throttling
- les temps de pause du GC

OOME / OOMK

Un manque de mémoire à des conséquences funestes

OutOfMemoryError (OOME) dans la JVM

Survient lors d'un manque de mémoire dans la JVM (heap, metaspace, stacks)

Exemple : la heap est pleine et le GC n'arrive pas à récupérer

Lié à un manque ou une fuite de mémoire

=> Génération et analyse d'un heapdump

OutOfMemoryKilled (OOMK) dans Kubernetes

Survient lorsque le pod manque de mémoire

Le process tente d'allouer de la mémoire au-delà de la limite fixée

Rien dans les logs applicatifs, il faut inspecter le pod

```
kubectl describe pod mon-pod
```

```
State:        Terminated
Reason:       OOMKilled
Exit Code:    137
```

La mémoire occupée par la JVM

La mémoire occupée par la JVM ne concerne pas que le heap

Elle est composée de plusieurs autres zones :

- Le metaspace (à partir de Java 9) / la permanent generation (avant Java 9)
- Les stacks des threads
- Éventuellement de la mémoire native
  - Pour obtenir des informations sur son utilisation

```
jcmd <pid> VM.native_memory
```

  - Avec l'option de la JVM `-XX:NativeMemoryTracking=summary ou detail`

Cela rend l'évaluation de la taille mémoire requise difficile à déterminer

Ne pas définir la mémoire du conteneur/pod équivalente à celle du heap

Temps de démarrage
et consommation de ressources

Le temps de démarrage d'une JVM

Souvent problématique dans une JVM depuis toujours

Des milliers classes sont chargées à la demande via un ClassLoader avec un coût

- Recherche dans le JDK, le classpath, chargement .class, vérification,
- Création une instance de Class, liaison de la classe, exécution initialisation static

De plus la JVM est dynamique :

- Utilisation de l'API Introspection
  - Recherche annotations, chargements et manipulations dynamiques, …
- Classes créées dynamiquement via des proxys
- Utilisation d'agents qui manipule le bytecode

Problématique accentuée dans Kubernetes :

- Déplacement d'un pod sur un autre nœud, autoscaling (HPA), …

Plusieurs techniques et fonctionnalités évoluent pour améliorer le temps de démarrage

CDS (Class Data Sharing)

Mécanisme de la JVM HotSpot de partage des métadonnées de classes
    Pour réduire leur temps de chargement

Le but principal est d'accélérer le démarrage des applications Java

    Qui prend beaucoup de temps et consomme de la mémoire

    Et surtout est répété à l'identique pour chaque JVM lancée

Peut permettre de réduire l'empreinte mémoire
    Surtout quand plusieurs JVM tournent sur la même machine

Utilise une archive binaire mappée en mémoire partagée en lecture seule

Mis en open source dans le JDK 11

    Archive par défaut fournie dans le JDK (uniquement classes du JDK)

AppCDS dans le JDK 13 (classes du JDK + applicatives)

    Requière une exécution préalable pour créer l'archive à utiliser ultérieurement

AOT (Ahead-Of-Time)

L'AOT a pour but :

- Réduire le temps de démarrage
- Réduire la consommation CPU au démarrage
- Améliorer la performance des applications à courte durée de vie

Cela permet de mieux convenir aux environnements cloud / serverless

Travaux du projet Leyden d'OpenJDK

Premières fonctionnalités livrées :

- Ahead-of-Time Class Loading & Linking : JDK 24 (JEP 483)
- Ahead-of-Time Command-Line Ergonomics : JDK 25 (JEP 514)
- Ahead-of-Time Method Profiling : JDK 25 (JEP 515)
- Ahead-of-Time Object Caching with Any GC : JDK 26 (JEP 516)

Ahead-of-Time Class Loading & Linking

JDK 24 (JEP 483)

But : réduire le temps de démarrage
- En rendant les classes instantanément disponibles, dans un état chargé et lié

S'appuie sur AppCDS

Contrairement à GraalVM
- Qui propose le même type de fonctionnalités mais dans un monde fermé

La JVM est dynamique
- La liste des classes à charger n'est pas forcément fixe au démarrage
- Car les classes peuvent être créées dynamiquement
- Par exemple via des proxys dynamiques ou chargées dynamiquement par réflexion

Avec AOT, toutes les fonctionnalités de la JVM sont conservées
- Une classe non présente dans l'archive sera chargée dynamiquement

Ahead-of-Time

Mise en œuvre en 3 étapes :

1) Exécution de l'application en mode enregistrement
   Pour enregistrer les classes à charger dans un fichier de configuration
   Avec les options -XX:AOTMode=record -XX:AOTConfiguration=xxx

```
java -XX:AOTMode=record -XX:AOTConfiguration=app.aotconf -cp monapp.jar fr.sciam.monapp.MainApp
```

2) Création du cache à partir du fichier de configuration
   Avec les options -XX:AOTMode=create -XX:AOTConfiguration=xxx -XX:AOTCache=yyy

```
java -XX:AOTMode=create -XX:AOTConfiguration=app.aotconf -XX:AOTCache=main.aot -cp monapp.jar
```

3) Exécution de l'application avec le cache
   Avec l'option -XX:AOTCache=yyy

```
java -XX:AOTCache=main.aot -cp monapp.jar fr.sciam.monapp.MainApp
```

Ahead-of-Time Command-Line Ergonomics

Dans le JDK 25 (JEP 514)

Simplifier la création du cache AOT dans la majorité des cas

Une nouvelle option de la commande java, AOTCacheOutput
    qui spécifie le fichier cache AOT

Utilisée seule, sans autres options AOT*
    Oblige la JVM à diviser son invocation en deux sous-invocations
    Dans les deux modes (AOTMode=record et AOTMode=create)

```
java -XX:AOTCacheOutput=main.aot -cp monapp.jar fr.sciam.monapp.MainApp
```

Ahead-of-Time Command-Line Ergonomics

Le mode enregistrement doit être similaire à la prod
- Exemple : ne pas utiliser de classes de suite de tests
- Se concentrer sur l'exécution d'un large éventail de scénarios

Toutes les exécutions doivent utiliser la même version du JDK
- Et être sur la même architecture matérielle (par exemple, x64 ou aarch64)
- Et le même système d'exploitation

Toutes les exécutions doivent avoir des options de modules cohérentes
- Et des graphes de modules cohérents
- Options -m, --module, -p, --module-path et --add-modules doivent être identiques,
- Ne PAS utiliser --limit-modules, --patch-module et --upgrade-module-path

Toutes les exécutions ne doivent pas utiliser d'agents JVMTI
- Qui peuvent réécrire arbitrairement des classes à l'aide de ClassFileLoadHook

Pas de support de ZGC jusqu'au JDK 25 inclus
- JEP 516, JDK 26 : introduction d'un format GC-agnostique (streamable)
- Moins performant mais compatible avec tous les GC

GraalVM Native Image

Permet de :

- Compiler toute l'application en binaire natif
- Avoir un démarrage très rapide
- Réduire l'empreinte mémoire

Mais :

- Certaines fonctionnalités limitées (reflection, proxys dynamiques…)
- Build plus complexe et surtout beaucoup plus lourd
- Double la charge de tests
- Plus de JVM au runtime : pas de JIT

Le JIT peut profiler le programme et optimiser à chaud (après un LONG warmup)

Avec un binaire natif : les optimisations sont faites à l'avance,

- Il n'y a plus d'optimisation dynamique

Résultat : meilleur démarrage

- Mais parfois moins performant sur les longues exécutions

GraalVM Native Image

Close world à la compilation

- Tout le code utilisé est connu à l'avance au moment de la compilation
- Compilations longues et très coûteuses en ressources

Certaines fonctionnalités ne sont pas utilisables ou partiellement :

- Manipulation de bytecode, chargement dynamique, instrumentation
  - Résultats : incompatibilités avec certaines bibliothèques
- Le support des fonctionnalités dynamiques
  - Reflection, Invoke Dynamic, Proxy dynamiques, Dynamic class loading, …

Requière l'écriture de fichiers de configuration (reflection config, resource config…)

- Manuellement
- Ou généré via un agent avec une exécution requise
  ayant un code coverage le plus élevé possible

JVM vs natif

	JVM	GraalVM Native
Démarrage	lent	**très rapide**
Mémoire	plus élevée	**plus faible**
Compatibilité	**totale**	parfois limitée
Build time	**rapide**	Lent et très coûteux en ressources
Performance long terme	**souvent meilleure**	parfois inférieure
diagnostic et observabilité	**totale**	plus compliqué

Natif intéressant pour :

App à durée de vie courte, Serverless, CLI, environnements contraints

JVM intéressant pour :

App à durée de vie longue (JIT), fonctionnalités dynamiques

Exemple : application Java en natif

Passage en natif pour notre Application de Demo

Préparer les ***Hints*** notamment pour la gestion de la réflexion…

(permet d'aider GraalVM à gérer ce qu'il ne peut pas comprendre en inspectant le code)

manuellement

avec l'agent de tracing qui intercepte les appels

avec des listes communautaires pour certains projets/frameworks

Intégrer le plugin de build

Compiler en natif

avec gradle: *./gradlew nativeCompile*

avec maven: *mvn -Pnative native:compile*

Fixer les problèmes éventuels et recommencer…

La première fois pas mal de trial & error à prévoir.

Le build en natif dure longtemps!

Plus simple qu'il y'a quelques années…

DemoService.java

```java
@Service
@RegisterReflectionForBinding(DemoService.Quote.class)
public class DemoService {

    public final String CHUCK_URL="https://api.chucknorris.io";
    @JsonIgnoreProperties(ignoreUnknown = true)
    public static record Quote (
        String id,
        String value
    ) {}


    public String getChuckNorrisQuote() {
        String result="n/a";
        RestClient restClient = RestClient.builder()
                .baseUrl(CHUCK_URL)
                .build();
        Quote quote=restClient.get()
                .uri("/jokes/random?category=dev")
                .retrieve()
                .body(Quote.class);
        result=quote.value;
        System.out.println("received quote: "+result);
        return result;
    }
```

DemoService.java (2)

```
    public static record PodInfo(String name, String namespace, String status)  {}
    public List<PodInfo> getAllPods() throws Exception {
        ApiClient client = Config.defaultClient();
        Configuration.setDefaultApiClient(client);
        CoreV1Api api = new CoreV1Api();
        V1PodList list = api.listPodForAllNamespaces()
                .execute();

        List<PodInfo> pods=list.getItems().stream()
                .map(pod -> new PodInfo(
                        pod.getMetadata().getName(),
                        pod.getMetadata().getNamespace(),
                        pod.getStatus().getPhase()
                ))
                .collect(Collectors.toList());
        System.out.println("found Pods: "+pods.size());
        return pods;
    }

}
```

DemoHintsRegistrar.java

```java
public class DemoHintsRegistrar implements RuntimeHintsRegistrar {
    private static final Logger LOG = LoggerFactory.getLogger(DemoHintsRegistrar.class);

    @Override
    public void registerHints(RuntimeHints hints, ClassLoader classLoader) {
        // 1. Create a scanner that doesn't look for specific annotations (just everything)
        ClassPathScanningCandidateComponentProvider scanner=new
                ClassPathScanningCandidateComponentProvider(false);
        scanner.addIncludeFilter((metadataReader, metadataReaderFactory) -> true);

        // 2. Scan the Kubernetes model package
        scanner.findCandidateComponents("io.kubernetes.client.openapi.models").forEach(bean -> {
            try {
                Class<?> clazz = Class.forName(bean.getBeanClassName());
                // 3. Register the class with all the "unsafe" categories needed for K8s models
                hints.reflection().registerType(clazz, MemberCategory.values());
            } catch (ClassNotFoundException e) {
                // Ignore classes that can't be loaded
            }
        });

        // 3. Scan the Kubernetes custom package
        scanner.findCandidateComponents("io.kubernetes.client.custom").forEach(bean -> {
        // (...)
```

DemoApplication.java

```
public class DemoHintsRegistrar implements RuntimeHintsRegistrar {


@SpringBootApplication
@ImportRuntimeHints(DemoHintsRegistrar.class)
public class DemoApplication {
    private static final Logger LOG = LoggerFactory.getLogger(DemoApplication.class);

    public static void main(String[] args) {
        LOG.info("starting...");
        System.setProperty("spring.main.banner-mode", "off");
        String kubeConfig = System.getenv("KUBECONFIG");
        if (kubeConfig != null) {
            LOG.info("Found KUBECONFIG: {}", kubeConfig);
        } else {
            LOG.warn("KUBECONFIG environment variable is NOT set.");
        }

        SpringApplication.run(DemoApplication.class, args);
    }
}
```

build.gradle

```
plugins {
    id 'java'
    id 'org.springframework.boot' version '3.5.13'
    id 'io.spring.dependency-management' version '1.1.7'
    id 'org.graalvm.buildtools.native' version '0.10.1'
}

group = 'io.kubolabs'
version = '0.0.1-SNAPSHOT'
description = 'demosb'

java {
    toolchain {
        languageVersion = JavaLanguageVersion.of(21)
    }
}

repositories {
    mavenCentral()
}

dependencies {
    implementation 'org.springframework.boot:spring-boot-starter-web'
    implementation 'io.kubernetes:client-java:26.0.0'
    implementation 'org.reflections:reflections:0.10.2'
    // for native
    testImplementation 'org.springframework.boot:spring-boot-starter-test'
}
```

build.gradle (2)

```
graalvmNative {
    metadataRepository {
        enabled = true
        // This will pull the community hints for the K8s client and other libraries like Jackson or Netty
    }
    binaries {
        main {
            // 1. Directly read the environment variables set by Flox
            def floxEnv = System.getenv("FLOX_ENV_CACHE")

            if (floxEnv !=null) {
                println("Flox detected...")
                def zlibPath = System.getenv("ZLIB_FULL_PATH")

                // 2. If the variable exists, apply the specialized Nix/Flox paths
                if (zlibPath != null && !zlibPath.isEmpty()) {
                    // Use the -H:CCompilerOption wrapper to safely pass the -I flag
                    buildArgs.add("-H:CCompilerOption=-I${zlibPath}/include")
                    buildArgs.add("-H:CLibraryPath=${zlibPath}/lib")
                    println "Flox detected: Using zlib from ${zlibPath}"
                } else {
                    // If not in Flox, GraalVM will look in standard /usr/lib and /usr/include
                    println "WARNING!!! Flox detected, but ZLIB_FULL_PATH incorrectly SET! using defaults"
                }
            }

            def osName = System.getProperty("os.name").toLowerCase()
            def isMac = osName.contains("mac")
            if (isMac) {
                println "Build environment: macOS detected. Upping memory limit to 12GB."
                buildArgs.add("-J-Xmx12g")
            } else {
                println "Build environment: Linux detected. Using 5GB memory limit to avoid Exit Code 137 OOM."
                buildArgs.add("-J-Xmx5g")

            }
```

Build en natif (linux VM 2cpu/8GB - flox)

```
// build native
$ ./gradlew nativeCompile
> Configure project :
Flox detected...
Flox detected: Using zlib from /nix/store/2kdz3m7ic8w226pcvkz1dlg169v91p6a-zlib-1.3.2
Build environment: Linux/Other detected. Using 5GB memory limit to avoid Exit Code 137 OOM.
> Task :processAot

(...)

Produced artifacts:
 /home/areg/flox/graalvm21/demosb/build/native/nativeCompile/demosb (executable)
========================================================================================
Finished generating 'demosb' in 22m 27s.
[native-image-plugin] Native Image written to:
/home/areg/flox/graalvm21/demosb/build/native/nativeCompile
BUILD SUCCESSFUL in 23m 3s
```

Build en natif (mac M3 Pro 36GB)

```
// build native
$ ./gradlew nativeCompile
> Configure project :
Build environment: macOS detected. Upping memory limit to 12GB.
> Task :processAot

(...)

Produced artifacts:
 /Users/alain/data/dev/java/demosb/build/native/nativeCompile/demosb (executable)
========================================================================================
Finished generating 'demosb' in 3m 46s.
[native-image-plugin] Native Image written to:
/Users/alain/data/dev/java/demosb/build/native/nativeCompile
BUILD SUCCESSFUL in 4m 1s
```

Build en natif avec multi stage Dockerfile

```
// build native with multi stage Dockerfile
$ vi Dockerfile.native
    # --- Stage 1: Build ---
    FROM container-registry.oracle.com/graalvm/native-image:21 AS builder

    WORKDIR /build
    COPY . .
    # Run the Gradle native compile
    RUN ./gradlew nativeCompile

    # --- Stage 2: Runtime ---
    # 'cc-debian12' includes glibc and libstdc++ required for native apps
    FROM gcr.io/distroless/cc-debian12

    WORKDIR /app
    # Copy the binary from the builder stage
    COPY --from=builder /build/build/native/nativeCompile/demosb /app/demosb

    EXPOSE 8080
    ENTRYPOINT ["/app/demosb"]
```

```
// build
$ docker build -t areg/demosb-native-linux:0.1 -f Dockerfile.native .
```

Comparaison Standard vs Native: Pod Events

```
// Test on a GKE cluster 1.35 (2 workers 1cpu/4GB) with both images in a GCR registry
// Events for pod with Standard image
  Type    Reason     Age   From               Message
  ----    ------     ----  ----               -------
  Normal  Scheduled  68s   default-scheduler  Successfully assigned default/demosb-7585df87b8-rgftw to gke-testcluster-default-pool-03a11ce2-3lct
  Normal  Pulling    67s   kubelet            spec.containers{app}: Pulling image "gcr.io/kubo-sandbox1/demosb/demosb:0.1"
  Normal  Pulled     54s   kubelet            spec.containers{app}: Successfully pulled image "gcr.io/kubo-sandbox1/demosb/demosb:0.1" in 12.581s (12.581s including waiting). Image size: 167569523 bytes.
  Normal  Created    54s   kubelet            spec.containers{app}: Container created
  Normal  Started    53s   kubelet            spec.containers{app}: Container started


// Events for pod with Native image
  Type    Reason     Age    From               Message
  ----    ------     ----   ----               -------
  Normal  Scheduled  2m34s  default-scheduler  Successfully assigned default/demosb-native-d4d7cf4ff-tbvk8 to gke-testcluster-default-pool-03a11ce2-3lct
  Normal  Pulling    2m33s  kubelet            spec.containers{app}: Pulling image "gcr.io/kubo-sandbox1/demosb/demosb-native-linux:0.1"
  Normal  Pulled     2m28s  kubelet            spec.containers{app}: Successfully pulled image "gcr.io/kubo-sandbox1/demosb/demosb-native-linux:0.1" in 4.92s (4.92s including waiting). Image size: 60418000 bytes.
  Normal  Created    2m28s  kubelet            spec.containers{app}: Container created
  Normal  Started    2m28s  kubelet            spec.containers{app}: Container started
```

Comparaison Standard vs Native: démarrage

```
// startup logs for pod with standard image (CPU limit: 200m)
$ kubectl logs demosb-7585df87b8-rgftw
12:10:15.825 [main] INFO io.kubolabs.demosb.DemoApplication -- starting...
12:10:15.922 [main] WARN io.kubolabs.demosb.DemoApplication -- KUBECONFIG environment variable is NOT set.
2026-04-23T12:10:31.923Z  INFO 1 --- [demosb] [           main] io.kubolabs.demosb.DemoApplication       : Starting DemoApplication v0.0.1-SNAPSHOT using Java 21.0.10 with PID 1 (/app/app.jar started by springuser in /app)
2026-04-23T12:10:31.926Z  INFO 1 --- [demosb] [           main] io.kubolabs.demosb.DemoApplication       : No active profile set, falling back to 1 default profile: "default"
2026-04-23T12:10:57.022Z  INFO 1 --- [demosb] [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port 8080 (http)
2026-04-23T12:10:57.321Z  INFO 1 --- [demosb] [           main] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
2026-04-23T12:10:57.322Z  INFO 1 --- [demosb] [           main] o.apache.catalina.core.StandardEngine    : Starting Servlet engine: [Apache Tomcat/10.1.53]
2026-04-23T12:10:59.328Z  INFO 1 --- [demosb] [           main] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
2026-04-23T12:10:59.330Z  INFO 1 --- [demosb] [           main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 26397 ms
2026-04-23T12:11:13.330Z  INFO 1 --- [demosb] [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port 8080 (http) with context path '/'
2026-04-23T12:11:13.722Z  INFO 1 --- [demosb] [           main] io.kubolabs.demosb.DemoApplication       : Started DemoApplication in 54.604 seconds (process running for 69.6)

// ==> 54.604 seconds

// startup logs for pod with native image (CPU limit: 200m)
$ kubectl logs demosb-native-d4d7cf4ff-tbvk8
(...)
2026-04-23T12:10:43.061Z  INFO 1 --- [demosb] [           main] io.kubolabs.demosb.DemoApplication       : Starting AOT-processed DemoApplication using Java 21.0.11 with PID 1 (/app/demosb started by root in /app)
2026-04-23T12:10:43.061Z  INFO 1 --- [demosb] [           main] io.kubolabs.demosb.DemoApplication       : No active profile set, falling back to 1 default profile: "default"
2026-04-23T12:10:43.161Z  INFO 1 --- [demosb] [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port 8080 (http)
2026-04-23T12:10:43.162Z  INFO 1 --- [demosb] [           main] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
2026-04-23T12:10:43.162Z  INFO 1 --- [demosb] [           main] o.apache.catalina.core.StandardEngine    : Starting Servlet engine: [Apache Tomcat/10.1.53]
2026-04-23T12:10:43.226Z  INFO 1 --- [demosb] [           main] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
2026-04-23T12:10:43.226Z  INFO 1 --- [demosb] [           main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 165 ms
2026-04-23T12:10:43.426Z  INFO 1 --- [demosb] [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port 8080 (http) with context path '/'
2026-04-23T12:10:43.428Z  INFO 1 --- [demosb] [           main] io.kubolabs.demosb.DemoApplication       : Started DemoApplication in 0.462 seconds (process running for 0.475)
// ==> 0.462 seconds
```

CRaC (Coordinated Restore at Checkpoint)

But principal est de supprimer le temps de warmup
(chargement des classes, initialisation, JIT, caches, etc.)

Comme une JVM est une VM

Sauvegarder l'état d'une JVM en cours d'exécution

Pour la restaurer ultérieurement quasi instantanément

Projet en incubation d'OpenJDK
Pas encore standard dans toutes les distributions LTS

Les contraintes :

Disponible uniquement dans certaines distributions comme Azul ou Bellsoft
ou builds CRaC expérimentales

Ne fonctionne actuellement que sous Linux : repose sur CRIU du noyau

aux droits du user requis (CAP_CHECKPOINT_RESTORE)

CRaC (Coordinated Restore at Checkpoint)

Nécessite une gestion stricte des ressources externes

Les ressources systèmes I/O (fichiers, sockets, ...) ne sont pas capturées

Elles doivent être fermées au checkpoint et récréées au restore

CRaC propose une API pour enregistrer :

des hooks avant checkpoint

des hooks après restore

Dépendance crac.org à ajouter

Génération au démarrage UUIDs, clés aléatoires

réinitialiser les PRNG dans afterRestore()

```
import org.crac.*;

public class MoService implements Resource {
   public MonService() {
      Core.getGlobalContext().register(this);
   }

   @Override
   public void beforeCheckpoint(Context<? extends Resource>
context) {
      System.out.println("Traitements avant le snapshot");
   }

   @Override
   public void afterRestore(Context<? extends Resource>
context) {
      System.out.println("Traitements après restauration");
   }
}
```

CRaC (Coordinated Restore at Checkpoint)

La mise en œuvre

Démarrer la JVM avec -XX:CRaCCheckpointTo

```
java -XX:CRaCCheckpointTo=$HOME/crac-image/ -jar mon_app.jar
```

Création d'un point de restauration qui arrête la JVM avec jcmd

```
jcmd mon-app.jar JDK.checkpoint
```

Redémarrer la JVM au point de restauration avec -XX:CRaCCheckpointFrom

```
java -XX:CRaCRestoreFrom=$HOME/crac-image/
```

Ne pas réduire le nombre de ressources (CPU, RAM) à la restauration
Car les ergonomics ne sont pas recalculés

Conclusion

Conclusion

Kubernetes et Java sont deux technologies largement utilisées

- Souvent ensemble

Pour s'éviter de gros écueils, il est important :

- De connaitre les spécificités de Kubernetes
- De connaitre les spécificités de la JVM
- De configurer les deux en adéquation

On a vu les principaux points à prendre en compte

Il y a aussi parfois des particularités liées à l'application exécutée
qui peuvent nécessiter une config et/ou un tuning spécifique avec benchmark

Tout cela requiert de la coordination et de la communication entre Dev et Ops

Merci pour votre attention