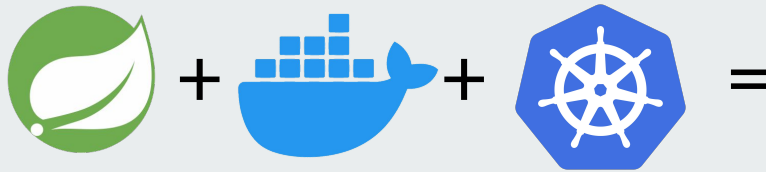

Spring Boot & Containers Do's & Don'ts



Salut 🖐️

Moi c'est Julien



Freelance @CodeKaio

Associé @Ekité

Teacher @univ-lille

👶 Speaker (DevFest
Lille - Sunny Tech)



Pourquoi ce BBL?

Exécuter une application dans un container c'est facile

Un Dockerfile 🐳, un jar 📦 et
hop 🚀

Moi qui mets mon container en prod



Les admins sys qui voient mon container en prod



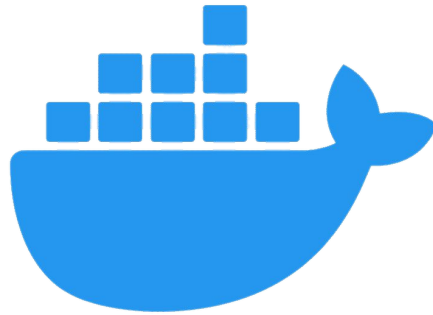
**J'écris des (mauvais) Dockerfile
depuis 2015**

**J'écris du Spring Boot depuis la
version 1.0 (2014)**

Bonnes pratiques

Spring Boot & Containers

3 parties



Spring Boot, Docker et Kubernetes - c'est facile



Dockerfile

```
FROM eclipse-temurin:17-jre
COPY target/incom-4.5.0.jar app.jar
ENTRYPOINT ["java", "-jar", "app.jar"]
```

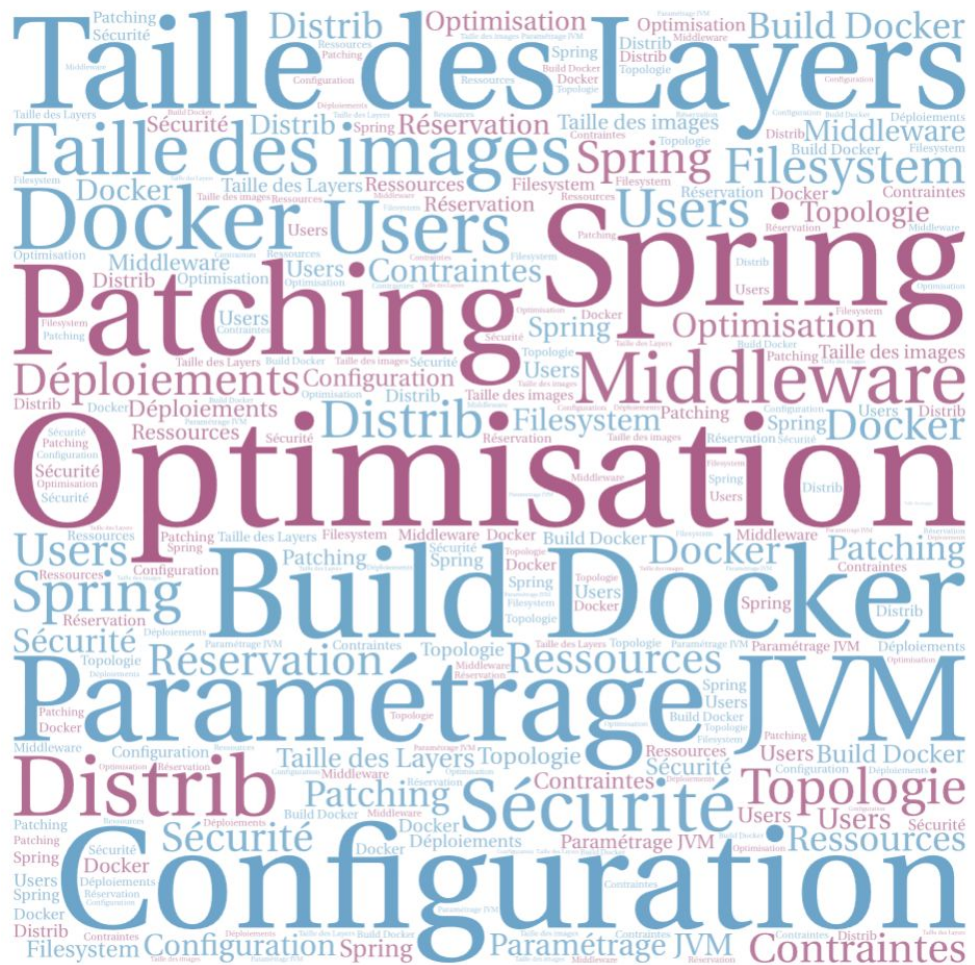


pod.yaml

```
apiVersion: v1
kind: Pod
metadata:
  name: incom
spec:
  containers:
  - name: app
    image: incom:4.5.0
    ports:
    - containerPort: 8080
```



Spring Boot, Docker et Kubernetes - C'est compliqué





Spring Boot, Docker et Kubernetes - C'est compliqué



```
FROM eclipse-temurin:17-jdk-alpine as build
WORKDIR /workspace/app

COPY mvnw .
COPY .mvn .mvn
COPY pom.xml .
COPY src src

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

FROM eclipse-temurin:17-jdk-alpine
VOLUME /tmp
ARG DEPENDENCY=/workspace/app/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/*", "hello.Application"]
```



Spring Boot, Docker et Kubernetes - C'est compliqué

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: {{ include "lol.fullname" . }}
  labels:
    {{- include "lol.labels" . | nindent 4 }}
spec:
  {{- if not .Values.autoscaling.enabled }}
  replicas: {{ .Values.replicaCount }}
  {{- end }}
  selector:
    matchLabels:
      {{- include "lol.selectorLabels" . | nindent 6 }}
  template:
    metadata:
      {{- with .Values.podAnnotations }}
      annotations:
        {{- toYaml . | nindent 8 }}
      {{- end }}
      labels:
        {{- include "lol.selectorLabels" . | nindent 8 }}
    spec:
      {{- with .Values.imagePullSecrets }}
      imagePullSecrets:
        {{- toYaml . | nindent 8 }}
      {{- end }}
      serviceAccountName: {{ include "lol.serviceAccountName" . }}
      securityContext:
        {{- toYaml .Values.podSecurityContext | nindent 8 }}
      containers:
        - name: {{ .Chart.Name }}
          securityContext:
            {{- toYaml .Values.securityContext | nindent 12 }}
          image: "{{ .Values.image.repository }}:{{ .Values.image.tag | default .Chart.AppVersion }}"
          imagePullPolicy: {{ .Values.image.pullPolicy }}
          ports:
            - name: http
              containerPort: 80
              protocol: TCP
          livenessProbe:
            httpGet:
              path: /
              port: http
          readinessProbe:
            httpGet:
              path: /
              port: http
          resources:
            {{- toYaml .Values.resources | nindent 12 }}
      {{- with .Values.nodeSelector }}
      nodeSelector:
        {{- toYaml . | nindent 8 }}
      {{- end }}
      {{- with .Values.affinity }}
      affinity:
        {{- toYaml . | nindent 8 }}
      {{- end }}
      {{- with .Values.tolerations }}
      tolerations:
        {{- toYaml . | nindent 8 }}
      {{- end }}
```

Spring Boot & JVM

Bonnes pratiques

“Your Spring Boot app, prepare you must”

– Yoda, The Empire Strikes Back (1980)



la configuration

Utiliser les profils Spring

Prévoir des profil de configuration multiples:

- par environnement : local, TU, development, staging, production...
- par typologie de déploiement : on-premises, Kubernetes sur un cloud, serverless

 composition de profils en fonction de l'endroit où on déploie

 jamais de secrets dans les properties !

[Doc Spring Profile Specific Files](#)





les healthcheck

 Exposer des endpoint HTTP pour connaître la santé de l'application

- est-elle démarrée ?
- est-elle prête à répondre à des requêtes ? (Connexion à la BDD dispo par ex.)

 `spring-boot-starter-actuator`

- utilisé par les healthcheck Docker, et par les probes Kubernetes
- peut être utilisé par les healthcheck de Load Balancing (on-premises)
- utilisé par des sondes de monitoring

spring-boot-starter-actuator 🕶️

Plein de endpoint cools pour débbuger une appli

- `/actuator/env` : récupérer la configuration
- `/actuator/httpexchanges` : récupérer les 100 dernières requêtes traitées
- `/actuator/loggers` : récupérer et modifier (🧑🔧) la configuration des logs
- `/actuator/health` & `/actuator/metrics` : récupérer la vie de l'application
- `/actuator/mappings` : liste les endpoints de l'application

...

```
management.endpoints.web.exposure.include=*
```



les métriques

💡 Exposer des métriques techniques et fonctionnelles

🎉 `spring-boot-starter-actuator` embarque `micrometer`

ajout d'une dépendance pour être compatible Prometheus ou OpenMetrics

- `/actuator/prometheus` : liste les métriques au format Prometheus

les métriques



pom.xml

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
<dependency>
  <groupId>io.micrometer</groupId>
  <artifactId>micrometer-registry-prometheus</artifactId>
</dependency>
```

les métriques




actuator/prometheus

```
# HELP system_cpu_count The number of processors available to the Java virtual machine
# TYPE system_cpu_count gauge
system_cpu_count 8.0
# HELP jvm_threads_peak_threads The peak live thread count since the Java virtual machine started or peak was reset
# TYPE jvm_threads_peak_threads gauge
jvm_threads_peak_threads 21.0
# HELP system_load_average_1m The sum of the number of runnable entities queued to available processors and the
number of runnable entities running on the available processors averaged over a period of time
# TYPE system_load_average_1m gauge
system_load_average_1m 0.919921875
# HELP jvm_buffer_total_capacity_bytes An estimate of the total capacity of the buffers in this pool
# TYPE jvm_buffer_total_capacity_bytes gauge
jvm_buffer_total_capacity_bytes(id="mapped - 'non-volatile memory'",) 0.0
jvm_buffer_total_capacity_bytes(id="mapped",) 0.0
jvm_buffer_total_capacity_bytes(id="direct",) 16384.0
# HELP jvm_gc_max_data_size_bytes Max size of long-lived heap memory pool
# TYPE jvm_gc_max_data_size_bytes gauge
jvm_gc_max_data_size_bytes 1.6768827392E10
# HELP executor_queued_tasks The approximate number of tasks that are queued for execution
# TYPE executor_queued_tasks gauge
executor_queued_tasks(name="applicationTaskExecutor",) 0.0
# HELP jvm_memory_max_bytes The maximum amount of memory in bytes that can be used for memory management
# TYPE jvm_memory_max_bytes gauge
jvm_memory_max_bytes(area="heap", id="G1 Survivor Space",) -1.0
jvm_memory_max_bytes(area="heap", id="G1 Old Gen",) 1.6768827392E10
jvm_memory_max_bytes(area="nonheap", id="Metaspace",) -1.0
jvm_memory_max_bytes(area="nonheap", id="CodeCache",) 5.0331648E7
jvm_memory_max_bytes(area="heap", id="G1 Eden Space",) -1.0
jvm_memory_max_bytes(area="nonheap", id="Compressed Class Space",) 1.073741824E9
# HELP http_server_requests_active_seconds_max
# TYPE http_server_requests_active_seconds_max gauge
http_server_requests_active_seconds_max(exception="none", method="GET", outcome="SUCCESS", status="200", uri="UNKNOWN",
) 0.00874405
# HELP http_server_requests_active_seconds
# TYPE http_server_requests_active_seconds summary
http_server_requests_active_seconds_active_count(exception="none", method="GET", outcome="SUCCESS", status="200", uri="
UNKNOWN",) 1.0
http_server_requests_active_seconds_duration_sum(exception="none", method="GET", outcome="SUCCESS", status="200", uri="
UNKNOWN",) 0.008720054
# HELP system_cpu_usage The "recent cpu usage" of the system the application is running in
# TYPE system_cpu_usage gauge
system_cpu_usage 0.0
# HELP jvm_threads_states_threads The current number of threads
# TYPE jvm_threads_states_threads gauge
jvm_threads_states_threads(state="runnable",) 7.0
jvm_threads_states_threads(state="blocked",) 0.0
jvm_threads_states_threads(state="waiting",) 11.0
jvm_threads_states_threads(state="timed-waiting",) 3.0
jvm_threads_states_threads(state="new",) 0.0
jvm_threads_states_threads(state="terminated",) 0.0
# HELP jvm_memory_committed_bytes The amount of memory in bytes that is committed for the Java virtual machine to
use
# TYPE jvm_memory_committed_bytes gauge
jvm_memory_committed_bytes(area="heap", id="G1 Survivor Space",) 8388608.0
jvm_memory_committed_bytes(area="heap", id="G1 Old Gen",) 5.0331648E7
jvm_memory_committed_bytes(area="nonheap", id="Metaspace",) 3.3161216E7
jvm_memory_committed_bytes(area="nonheap", id="CodeCache",) 9895936.0
jvm_memory_committed_bytes(area="heap", id="G1 Eden Space",) 8.388608E7
jvm_memory_committed_bytes(area="nonheap", id="Compressed Class Space",) 4784128.0
# HELP jvm_threads_live_threads The current number of live threads including both daemon and non-daemon threads
# TYPE jvm_threads_live_threads gauge
jvm_threads_live_threads 21.0
```



graceful shutdown

 Configurer l'application pour qu'elle s'éteigne proprement, en terminant les dernières requêtes HTTP en cours de traitement

Permettra de faire des déploiement en Rolling Update sans coupure de service

```
server.shutdown=graceful
```



JVM ⚡

Ayez conscience des besoins de votre application CPU/RAM:

- vos futurs paramètres `-Xms` et `-Xmx`
- vos futurs `resources.requests/limits.cpu/memory` Kubernetes

Faites des tirs de performance pour les découvrir ou les valider



native-image

 Expérimentez le build d'image native

 : booste le temps de démarrage des applications (qq millisecondes !)

 : attention aux limitations de Spring Boot

 : coût de compilation élevé

 : avec GraalVM community, uniquement le GC Serial 😞

<https://docs.spring.io/spring-boot/docs/current/reference/html/native-image.html>

Spring Boot & JVM

Les pièges à éviter

“It’s a trap”

– Admiral Ackbar,
Return of the Jedi
(1983)





`stream.parallel()`

`CompletableFuture.run()/supply()`



ForkJoinPool, on oublie

Parallélisme par défaut = nombre de CPU “visibles” par la JVM



Usage de `stream.parallel()` à bannir

Aucun contrôle sur le pool utilisé, peut être contre productif si le pool ne contient que 2 threads !

`CompletableFuture` peut être utilisé en contrôlant le pool de threads utilisé.



new File()

new FileOutputStream()

—



Filesystem, on oublie

Les containers sont volatiles 🗑️

Créés/supprimés au bon vouloir des orchestrateurs.

Le filesystem ne persiste pas, sauf gestion de volumes.



limiter l'adhérence des applications au filesystem

Pour le code générant des fichiers (exports PDF, XLS), manipuler des `InputStream/OutputStream` qui pointent vers des buckets par exemple.



<https://incom.net/actuator/env>



spring-boot-starter-actuator sur 8080, on oublie

⚠ Attention à l'exposition de l'actuator

💡 Utiliser un port différent pour exposer l'actuator

ou configurer `spring-security` sur les endpoints (facile à faire, mais rend l'usage vraiment compliqué)

```
server.port=8080
```

```
management.server.port=9090
```



<Appenders>

<File name="MyFile" fileName="logs/app.log">

</File>

</Appenders>



logger sur le filesystem, on oublie

Pratique en VMs, couplé à une centralisation
À laisser tomber en container. (pas d'adhérence au filesystem)

 Logger sur la console stdout/stderr

Logger sur le filesystem du container implique:

- un filesystem dispo en écriture
- un outil pour extraire les logs du container (déployé en sidecar k8s 😞)
- `docker log` et `kubectl log` qui ne marchent pas ! 😞

 Prévoir une configuration double avec des profils différents si l'application est hybride.



**trop de trucs dans
application.properties/yaml**





application.properties/yaml

Ces fichiers contiennent les propriétés qui sont communes à tous les profils.

✗ interdit d'y mettre des propriétés d'un environnement de dev.

💡 Privilégiez la création d'un profil Spring 'local' par exemple.

[Doc Spring Externalized Configuration](#) à connaître par ❤️



application.properties/yaml

1. Default properties (specified by setting `SpringApplication.setDefaultProperties`).
2. `@PropertySource` annotations on your `@Configuration` classes. Please note that such property sources are not added to the `Environment` until the application context is being refreshed. This is too late to configure certain properties such as `logging.*` and `spring.main.*` which are read before refresh begins.
3. Config data (such as `application.properties` files).
4. A `RandomValuePropertySource` that has properties only in `random.*`.
5. OS environment variables.
6. Java System properties (`System.getProperties()`).
7. JNDI attributes from `java:comp/env`.
8. `ServletContext` init parameters.
9. `ServletConfig` init parameters.
10. Properties from `SPRING_APPLICATION_JSON` (inline JSON embedded in an environment variable or system property).
11. Command line arguments.
12. `properties` attribute on your tests. Available on `@SpringBootTest` and the [test annotations for testing a particular slice of your application](#).
13. `@TestPropertySource` annotations on your tests.
14. [Devtools global settings properties](#) in the `$HOME/.config/spring-boot` directory when devtools is active.



application.properties/yaml

1. **Application properties** packaged inside your jar (`application.properties` and YAML variants).
2. **Profile-specific application properties** packaged inside your jar (`application-{profile}.properties` and YAML variants).
3. **Application properties** outside of your packaged jar (`application.properties` and YAML variants).
4. **Profile-specific application properties** outside of your packaged jar (`application-{profile}.properties` and YAML variants).

Docker

Bonnes pratiques

“Would it help if I got out and push your container image ?”

– Leia, The Empire Strikes Back (1980)



Partir d'un Dockerfile simple, et l'améliorer

i Un bon guide : <https://spring.io/guides/topicals/spring-boot-docker/>





Dockerfile simple (simpliste)

Une image 🐳, un fat-jar 📦 buildé par
maven et hop 🚀



Dockerfile

```
FROM eclipse-temurin:17-jre  
COPY target/incom-4.5.0.jar app.jar  
ENTRYPOINT ["java", "-jar", "app.jar"]
```

Sélectionnez une image de base

Les questions à se poser (avec les ops) :

? layer distribution ? ubuntu ? alpine ?

? jdk ? / jre ?

? version de java (17 ? 20 ?)

Un bon point de départ:

 eclipse-temurin:17-jre

eclipse-temurin:17-jre	188 Mo
ubuntu:jammy	78 Mo

eclipse-temurin:17-jdk	377 Mo
ubuntu:jammy	78 Mo

eclipse-temurin:17-jre-alpine	163 Mo
alpine	7 Mo

Optimisez vos layers

- ✗ avoir le moins de layers possibles
- ✓ layers réutilisables
- ✓ petites layers
- ⚡ optimiser la construction
- ⚡ optimiser le déploiement si juste un bout de code a changé

mon-application.jar	80 Mo
eclipse-temurin:17-jre	188 Mo
ubuntu:jammy	78 Mo

mon-application/classes	10 Mo
mon-application/libs	70 Mo
eclipse-temurin:17-jre	188 Mo
ubuntu:jammy	78 Mo

Dockerfile multi-layers

- 📦 On build le fat-jar
- 💥 On expose le fat-jar
- 🐳 On ajoute une layer pour :

- les libs
- les meta data
- les classes

- ⚡ On tire partie du cache Docker
- ⚡ L'appli démarre plus vite

```
Shell

$ mkdir target/dependency
$ (cd target/dependency; jar -xf ../*.jar)
$ tree -L 2 target/dependency

.
├── BOOT-INF
│   ├── classes
│   ├── classpath.idx
│   ├── layers.idx
│   └── lib
├── META-INF
│   ├── MANIFEST.MF
│   └── maven
└── org
    └── springframework
```


```
Dockerfile

FROM eclipse-temurin:17-jre
ARG DEPENDENCY=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/*", "incom.Application"]
```

Dockerfile multi-layers & multi-stage

 Tout dans le Dockerfile

 Top pour la CI !

 : presque des builds
reproductibles



Dockerfile

```
FROM maven:3-eclipse-temurin-17 as build
WORKDIR /workspace/app

COPY pom.xml .
COPY src src

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

FROM eclipse-temurin:17-jre
ARG DEPENDENCY=/workspace/app/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/*", "incom.Application"]
```

Tirez partie du cache Docker 🐳

- 👍 Pratique pour une intégration continue
- ✅ Permet de ne pas reconstruire les layers n'ayant pas été modifiées
- 😬 Fastidieux à configurer, mais ça vaut le coup <https://docs.docker.com/build/cache/backends/>

mon-application/classes	10 Mo
mon-application/libs	70 Mo
eclipse-temurin:17-jre	188 Mo
ubuntu:jammy	78 Mo



cache

Remplissez votre container de Heap 😊

Si votre container a 2 CPU et 4Go de RAM, la JVM va réserver 1Go de heap seulement par défaut

✓ `-Xms` et `-Xmx` à 75% de la RAM allouée au container

ou

✓ `-XX:InitialRAMPercentage=75 -XX:MaxRAMPercentage=75`



Shell

```
$ docker container run --cpus=2 --memory=4G --entrypoint java eclipse-temurin:17 '-Xlog:gc*' '-version'
[0.002s][info][gc] Using G1
[0.008s][info][gc,init] Version: 17.0.6+10 (release)
[0.008s][info][gc,init] CPUs: 8 total, 2 available
[0.008s][info][gc,init] Memory: 4096M
[0.008s][info][gc,init] Heap Min Capacity: 8M
[0.008s][info][gc,init] Heap Initial Capacity: 64M
[0.008s][info][gc,init] Heap Max Capacity: 1G
```

Attention au GC 🧹 utilisé !



Shell

```
$ docker container run --cpus=1 --entrypoint java eclipse-temurin:17 '-Xlog:gc*' '-version'
[0.002s][info][gc] Using Serial
[0.002s][info][gc,init] Version: 17.0.6+10 (release)

$ docker container run --cpus=2 --entrypoint java eclipse-temurin:17 '-Xlog:gc*' '-version'
[0.016s][info][gc] Using G1
[0.018s][info][gc,init] Version: 17.0.6+10 (release)
```

# CPU	GC
<1	Serial
1	Serial
>1	G1
∞	G1

⚠ Le ZGC n'est pas par défaut, à forcer si vous voulez l'expérimenter pour viser des temps de pause de GC plus faibles.

Un bon article ici : <https://kstefani.github.io/2021/11/24/gc-progress-8-17.html>

⚠ Attention au GC Serial si vous utilisez une image native

Docker & la JVM

💡 Proposez des paramètres de JVM par défaut

Positionnez la variable `JAVA_TOOL_OPTIONS` avec ces valeurs




Mettez à dispo une variable supplémentaire `JAVA_OPTS` pour la customisation



Dockerfile

```
FROM eclipse-temurin:17-jre
ENV JAVA_OPTS ''
ENV JAVA_TOOL_OPTIONS '-XX:InitialRAMPercentage=75 -XX:MaxRAMPercentage=75 -XX:useZGC'
ENTRYPOINT ["sh", "-c", "java ${JAVA_OPTS} -jar incom-1.0.0.jar"]
```

Versionning des images

 Utilisez le même versionning pour vos images Docker  que pour votre application .

Une bonne idée peut-être de récupérer le champ `version` du `pom.xml` et de l'utiliser pour tagger vos images docker.



Shell

```
$ TAG=$(mvn help:evaluate -Dexpression=project.version -q -DforceStdout)
$ NAME=$(mvn help:evaluate -Dexpression=project.artifactId -q -DforceStdout)
$ docker image build -t $NAME:$TAG .
```


Sécurité



❌ n'utilisez jamais le user root , chaque Dockerfile doit avoir une directive USER

❌ jamais de secrets dans un Dockerfile (ça reste dans les layers)

✅ mettez à jour vos images “parentes” régulièrement

✅ préférez pointer sur des images parentes avec un tag précis, comme : `eclipse-temurin:17.0.6_10-jre`

Cela rend la montée version explicite, plutôt que refaire un `docker image build --no-cache` 

✅ extrayez un SBOM de vos images (distrib+middleware+software), vos RSO/RSSI seront contents ,
et ce sera plus facile pour vous lors de la prochaine faille log4j 

Autres outils de build

 Expérimentez `jib` ou un `buildpack`

Ces outils permettent de construire des images Docker , sans avoir de Dockerfile

Ils implémentent les bonnes pratiques, c'est ça en moins à gérer de votre côté

<https://github.com/GoogleContainerTools/jib>

<https://buildpacks.io/>

Observez vos images 👁️👁️

💡 utilisez `dive` pour explorer vos images/layers

observer le filesystem des images en interactif
afficher des axes d'optimisation

<https://github.com/wagoodman/dive>

```
$ dive --ci $NAME/$TAG
Using default CI config
Image Source: docker://incom:latest
Fetching image... (this can take a while for large images)
Analyzing image...
  efficiency: 99.7544 %
  wastedBytes: 2001695 bytes (2.0 MB)
  userWastedPercent: 0.4751 %
Inefficient Files:
Count  Wasted Space  File Path
2      1.4 MB        /var/cache/debconf/templates.dat
2      315 kB        /var/log/dpkg.log
2      223 kB        /var/lib/dpkg/status
2      35 kB         /var/log/apt/history.log
2      29 kB         /var/cache/debconf/config.dat
2      13 kB         /var/cache/ldconfig/aux-cache
2      12 kB         /etc/ld.so.cache
2      11 kB         /var/log/apt/eipp.log.xz
2      1.3 kB        /var/lib/apt/extended_states
2      230 B         /var/lib/dpkg/triggers/File
2      0 B           /var/lib/apt/lists
2      0 B           /var/lib/dpkg/lock
2      0 B           /var/lib/dpkg/triggers/Unincorp
2      0 B           /var/lib/dpkg/updates
2      0 B           /var/cache/debconf/passwords.dat
2      0 B           /var/lib/dpkg/lock-frontent
2      0 B           /var/cache/apt/archives/partial
2      0 B           /var/cache/apt/archives/lock
3      0 B           /tmp
2      0 B           /var/lib/dpkg/triggers/Lock
Results:
  PASS: highestUserWastedPercent
  SKIP: highestWastedBytes: rule disabled
  PASS: lowestEfficiency
Result:PASS [Total:3] [Passed:2] [Failed:0] [Warn:0] [Skipped:1]
```

Comp	Size	Command
78 MB	FROM 14e938040835241	
48 MB	apt-get update	&& DEBIAN_FRONTEND=noninteractive apt-get install -y --no-install-recommends tzdata curl ca-cert
329 MB	set -eux; ARCH=\$(dpkg --print-architecture); case "\${ARCH}" in aarch64 arm64) ESUM='2e3c19c19c1707205c6b'	
0 B	echo Verifying install ...	&& echo javac -version && javac -version && echo java -version && java -version && java --version
44 MB	#(nop) COPY file:a0a6837210679d2fbc470f56d1278b4010976308ec39bbf3cedfaeff1699 in /opt/target/gaia.jar	

Layer Details

Tags: (unavailable)

Id: 6812d148f57c8a3761069bf665cf7f3a0cd1748f3a0a2ccc17a4d35ee07e5f0c

Digest: sha256:e369ffcb8f5d3ec8be94988cccec7f99d36b6f5eeb197421bf7512cb2bb761ae0

Command: set -eux; ARCH=\$(dpkg --print-architecture); case "\${ARCH}" in aarch64|arm64) ESUM='2e3c19c1707205c6b'

Image Details

Image name: gaiaapp/gaia

Total Image size: 499 MB

Potential wasted space: 2.0 MB

Image efficiency score: 99 %

Count	Total Space	Path
2	1.4 MB	/var/cache/debconf/templates.dat
2	315 kB	/var/log/dpkg.log
2	223 kB	/var/lib/dpkg/status
2	35 kB	/var/log/apt/history.log
2	29 kB	/var/cache/debconf/config.dat
2	13 kB	/var/cache/ldconfig/aux-cache
2	12 kB	/etc/ld.so.cache
2	11 kB	/var/log/apt/eipp.log.xz
2	1.3 kB	/var/lib/apt/extended_states
2	230 B	/var/lib/dpkg/triggers/File
2	0 B	/var/lib/apt/lists
2	0 B	/var/lib/dpkg/lock
2	0 B	/var/lib/dpkg/triggers/Unincorp
2	0 B	/var/lib/dpkg/updates
2	0 B	/var/cache/debconf/passwords.dat
2	0 B	/var/lib/dpkg/lock-frontent
2	0 B	/var/cache/apt/archives/partial
2	0 B	/var/cache/apt/archives/lock
3	0 B	/tmp
2	0 B	/var/lib/dpkg/triggers/Lock



Permission	UID:GID	Size	Filetree
-rwxrwxrwx	0:0	0 B	bin → usr/bin
drwxr-xr-x	0:0	0 B	boot
drwxr-xr-x	0:0	0 B	dev
drwxr-xr-x	0:0	435 kB	etc
drwxr-xr-x	0:0	0 B	home
-rwxrwxrwx	0:0	0 B	lib → usr/lib
-rwxrwxrwx	0:0	0 B	lib32 → usr/lib32
-rwxrwxrwx	0:0	0 B	lib64 → usr/lib64
-rwxrwxrwx	0:0	0 B	libx32 → usr/libx32
drwxr-xr-x	0:0	0 B	media
drwxr-xr-x	0:0	0 B	mnt
drwxr-xr-x	0:0	329 MB	opt
drwxr-xr-x	0:0	329 MB	java
drwxr-xr-x	0:0	329 MB	openjdk
-rw-r--r--	0:0	2.4 kB	NOTICE
drwxr-xr-x	0:0	458 kB	bin
drwxr-xr-x	0:0	99 kB	conf
drwxr-xr-x	0:0	209 kB	include
drwxr-xr-x	0:0	70 MB	jmods
drwxr-xr-x	0:0	160 kB	legal
drwxr-xr-x	0:0	248 MB	lib
drwxr-xr-x	0:0	746 kB	man
-rw-r--r--	0:0	1.6 kB	release
drwxr-xr-x	0:0	0 B	proc
drwxr-xr-x	0:0	3.3 kB	root
-rw-r--r--	0:0	3.1 kB	.bashrc
-rw-r--r--	0:0	161 B	.profile
drwxr-xr-x	0:0	7 B	run
drwxrwxrwx	0:0	0 B	lock
drwxr-xr-x	0:0	0 B	mount
drwxr-xr-x	0:0	7 B	systemd
-rw-r--r--	0:0	7 B	container
-rwxrwxrwx	0:0	0 B	sbin → usr/sbin
drwxr-xr-x	0:0	0 B	srv
drwxr-xr-x	0:0	0 B	sys
drwxrwxrwx	0:0	0 B	tmp
drwxr-xr-x	0:0	118 MB	usr
drwxr-xr-x	0:0	29 MB	bin
-rwxr-xr-x	0:0	52 kB	addpart
-rwxr-xr-x	0:0	15 kB	addr2line → x86_64-linux-gnu-addr2line
-rwxr-xr-x	0:0	19 kB	apt
-rwxr-xr-x	0:0	84 kB	apt-cache
-rwxr-xr-x	0:0	27 kB	apt-cdrom
-rwxr-xr-x	0:0	27 kB	apt-config
-rwxr-xr-x	0:0	52 kB	apt-get
-rwxr-xr-x	0:0	28 kB	apt-key
-rwxr-xr-x	0:0	52 kB	apt-mark
-rwxrwxrwx	0:0	0 B	ar → x86_64-linux-gnu-ar
-rwxr-xr-x	0:0	31 kB	arch
-rwxrwxrwx	0:0	0 B	as → x86_64-linux-gnu-as
-rwxrwxrwx	0:0	0 B	awk → /etc/alternatives/awk
-rwxr-xr-x	0:0	52 kB	b2sum
-rwxr-xr-x	0:0	35 kB	base32
-rwxr-xr-x	0:0	35 kB	base64
-rwxr-xr-x	0:0	35 kB	basenamer
-rwxr-xr-x	0:0	48 kB	basenc
-rwxr-xr-x	0:0	1.4 MB	bash
-rwxr-xr-x	0:0	6.8 kB	bashbug
-rwxrwxrwx	0:0	0 B	c++filt → x86_64-linux-gnu-c++filt
-rwxr-xr-x	0:0	7.4 kB	c_rehash
-rwxrwxrwx	0:0	0 B	captoinfo → tic
-rwxr-xr-x	0:0	35 kB	cat



Scannez vos images à la recherche de CVE

 utilisez `trivy` pour analyser vos images/layers

Listing des CVE sur toutes les layers:

-  : distrib
-  : appli

Va même fouiller dans les jar !

<https://aquasecurity.github.io/trivy/>

```

~/workspaces/trivy > ./trivy image goiaapp/gaia --severity HIGH,CRITICAL
2023-04-27T07:35:16.979+0200 INFO Vulnerability scanning is enabled
2023-04-27T07:35:16.979+0200 INFO Secret scanning is enabled
2023-04-27T07:35:16.979+0200 INFO If your scanning is slow, please try '--scanners vuln' to disable secret scanning
2023-04-27T07:35:16.979+0200 INFO Please see also https://aquasecurity.github.io/trivy/v0.40/docs/secret/scanning/#recommendation for faster secret detection
2023-04-27T07:35:16.984+0200 INFO Detected OS: ubuntu
2023-04-27T07:35:16.984+0200 INFO Detecting Ubuntu vulnerabilities...
2023-04-27T07:35:16.986+0200 INFO Number of language-specific files: 1
2023-04-27T07:35:16.986+0200 INFO Detecting jar vulnerabilities...

```

goiaapp/gaia (ubuntu 22.04)
 Total: 6 (HIGH: 6, CRITICAL: 0)

Library	Vulnerability	Severity	Installed Version	Fixed Version	Title
libssl3	CVE-2022-3602	HIGH	3.0.2-0ubuntu1.2	3.0.2-0ubuntu1.7	OpenSSL: X.509 Email Address Buffer Overflow https://avd.aquasec.com/nvd/cve-2022-3602
	CVE-2022-3786				OpenSSL: X.509 Email Address Variable Length Buffer Overflow https://avd.aquasec.com/nvd/cve-2022-3786
	CVE-2023-0286			3.0.2-0ubuntu1.8	X.400 address type confusion in X.509 GeneralName https://avd.aquasec.com/nvd/cve-2023-0286
openssl	CVE-2022-3602			3.0.2-0ubuntu1.7	OpenSSL: X.509 Email Address Buffer Overflow https://avd.aquasec.com/nvd/cve-2022-3602
	CVE-2022-3786				OpenSSL: X.509 Email Address Variable Length Buffer Overflow https://avd.aquasec.com/nvd/cve-2022-3786
	CVE-2023-0286			3.0.2-0ubuntu1.8	X.400 address type confusion in X.509 GeneralName https://avd.aquasec.com/nvd/cve-2023-0286

2023-04-27T07:35:16.994+0200 INFO Table result includes only package filenames. Use '--format json' option to get the full path to the package file.

Dava (jar)

Total: 11 (HIGH: 8, CRITICAL: 3)

Library	Vulnerability	Severity	Installed Version	Fixed Version	Title
com.fasterxml.jackson.core:jackson-databind (gaia.jar)	CVE-2022-42003	HIGH	2.13.3	2.12.7.1, 2.13.4.1	deep wrapper array nesting wrt UMRAP_SINGLE_VALUE_ARRAYS https://avd.aquasec.com/nvd/cve-2022-42003
	CVE-2022-42004			2.12.7.1, 2.13.4	jackson-databind: use of deeply nested arrays https://avd.aquasec.com/nvd/cve-2022-42004
net.minidev:json-smart (gaia.jar)	CVE-2023-1370		2.4.8	2.4.9	Uncontrolled Resource Consumption vulnerability in json-smart (Resource Exhaustion) https://avd.aquasec.com/nvd/cve-2023-1370
org.apache.tomcat.embed:tomcat-embed-core (gaia.jar)	CVE-2022-45143		9.0.65	9.0.69	JsonErrorReportValve injection https://avd.aquasec.com/nvd/cve-2022-45143
org.springframework.security:spring-security-core (gaia.jar)	CVE-2022-31690		5.7.2	5.6.9, 5.7.5	spring-security-oauth2-client: Privilege Escalation in spring-security-oauth2-client https://avd.aquasec.com/nvd/cve-2022-31690
org.springframework.security:spring-security-oauth2-client (gaia.jar)					

Docker

Les pièges à éviter

“You’ll be malfunctioning within a day, you nearsighted scrap container !”

– C3PO, A New Hope
(1977)





Pas de `.dockerignore`



.dockerignore 🙈

Liste les fichiers à ignorer lors de la construction de l'image Docker

Ignorer les répertoires habituels:

`.git/`

`target/`

`build/`

`node_modules/`

`.idea/`

`.settings/`

⚡ Booste la construction des images



**Chercher à tout prix à utiliser
une image customisée**





FROM mon_image_custom:latest

✘ Construire et maintenir une image parente nécessite beaucoup d'effort en suivi des mise à jour distribution, et middleware

😞 Vous aurez déjà assez de travail pour suivre et monter de version vos applis en prod

Si vous avez des contraintes particulières (comme des certificats à injecter dans vos JVM), c'est obligatoire, sinon oubliez



**Chercher à tout prix à utiliser
des images minimalistes
(alpine)**



ubuntu ✂ alpine jdk ✂ jre

- i Docker/Containerd optimise le stockage
- i une layer n'est présente qu'une fois sur le filesystem final (le node Kubernetes ou votre machine)

💡 Penser à la prod :

Une image plus petite contiendra moins d'outils qui peuvent être utiles (curl, jfr...)

Que cherche t-on à optimiser ? Les machines chez les Clouds ont un réseau performant, le stockage est peu coûteux ...

eclipse-temurin:17-jre	188 Mo
ubuntu:jammy	78 Mo

eclipse-temurin:17-jdk	377 Mo
ubuntu:jammy	78 Mo

eclipse-temurin:17-jre-alpine	163 Mo
alpine	7 Mo



Se casser la tête avec DinD (Docker in Docker)



Utilisez Kaniko ou Jib



builder vos images Docker dans un cluster Kubernetes, utilisez Kaniko ou Jib:

- ✓ Pas besoin de docker pour builder une image
- ✓ Fonctionne en “user-space” Linux, pas besoin d’être root
- ✓ **Kaniko** “exécute” les Dockerfile comme un `docker image build`
- ✓ **Jib** crée directement une image sans avoir besoin d’un Dockerfile

<https://github.com/GoogleContainerTools/kaniko>

<https://github.com/GoogleContainerTools/jib>

Kubernetes

Bonnes pratiques

“Pit tit tu ti ti tuuuuuuu
ti pip tu pi piiiip ti ti”

– R2D2, every damn
SW movie (1973-2019)





Utilisez des deployments, et pas des pods tous seuls

Déployez vos applications dans des namespaces

 des classiques, mais toujours bon à rappeler

Utilisez des requests/limits

 Un bon guide : <https://learn.microsoft.com/en-us/azure/developer/java/containers/overview>

Visez une QoS “Burstable” ou “Guaranteed”:

- ✓ CPU request
- ✓ CPU limits optionnelle : permet le Burst de CPU
- ✓ Memory request
- ✓ Memory limit : bonne pratique pour des JVM

Personne n'aime les `OutOfMemoryError` 



```
resources:  
  requests:  
    cpu: "2"  
    memory: "4Gi"  
  limits:  
    memory: "4Gi"
```



Chargez votre configuration avec des configMap ou des secrets

✓ N'hésitez pas à utiliser ces objets "natifs" Kubernetes

Les `ConfigMap` et `Secret` k8s peuvent être montés en variables d'environnement ou en fichier


Spring Boot comprend bien les variables d'environnement, et charge les fichiers dispo dans `spring.config.location`

<https://docs.spring.io/spring-boot/docs/current/reference/html/features.html#features.external-config.typesafe-configuration-properties.relaxed-binding.environment-variables>

<https://docs.spring.io/spring-boot/docs/current/reference/html/features.html#features.external-config.files>



Les secrets k8s ne sont pas secrets

Oui, c'est du bluff 

Pour les secrets vraiment secrets, clés d'API et autres mots de passe, plusieurs solutions:

- ✓ : `Sealed Secrets` (Bitnami) : le plus simple
- ✓ : `Vault` (Hashicorp) : le plus complexe (avec sidecar)
- ✓ : Key vaults managées des Cloud : le plus vendor lock-in

Jamais 2 pods sur le même node ou sur la même zone



Un classique, mais toujours bon à rappeler

Ça se fait facilement avec des contraintes de topologie sur un `Deployment` ou au niveau du cluster

<https://kubernetes.io/docs/concepts/scheduling-eviction/topology-spread-constraints/>

```
constraint.yaml

apiVersion: kubescheduler.config.k8s.io/v1beta3
kind: KubeSchedulerConfiguration

profiles:
  - schedulerName: default-scheduler
    pluginConfig:
      - name: PodTopologySpread
        args:
          defaultConstraints:
            - maxSkew: 1
              topologyKey: topology.kubernetes.io/zone
              whenUnsatisfiable: ScheduleAnyway
          defaultingType: List
```

Ne montez pas de token de service-account sur vos pods, sauf si nécessaire

ça évite de laisser un 🤖 faire le fofou dans votre namespace si votre RBAC est mal configuré

<https://kubernetes.io/docs/tasks/configure-pod-container/configure-service-account/#opt-out-of-api-credentials-automounting>

```
serviceAccount.yaml

apiVersion: v1
kind: ServiceAccount
metadata:
  name: build-robot
automountServiceAccountToken: false
```

Laissez mourir vos pods en paix

Quand Kubernetes supprime un Pod:

 : il SIGTERM le Pod

 : il met à jour les règles de routage (iptables)

 : il supprime le Pod des Service concernés

Pendant ce temps, il se peut que des requêtes soient envoyées au Pod, alors qu'il est en train de s'éteindre (tous les Nodes ne sont pas mis à jour au même instant)

Un preStop hook permet de retarder le SIGTERM pendant que le reste se met à jour

<https://www.thoughtworks.com/insights/blog/cloud/shutdown-services-kubernetes>



deployment.yaml

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

Kubernetes

Les pièges à éviter

“I find your lack of resource reservation disturbing”

– Darth Vader, A New Hope (1977)





**rebuild your application
to change a conf**



 **vous n'avez rien suivi, je vous envoie le seigneur
Vador**



Charger les configMap/secrets depuis l'API k8s



spring-cloud-kubernetes

✅ : Facile à utiliser, charge la config directement depuis l'API k8s, nécessite un `ServiceAccount` avec les droits adéquats.

⚠️ : Attention, votre code va commencer à dépendre de l'infrastructure !

🙄 : C'est probablement peu souhaitable. A désactiver en local.


🙅 : Aucun développeur ne veut faire tourner un cluster k8s sur son poste.



Charger la configuration avec un sidecar



Sidecars, Vaults et autres

 Déposer un fichier de propriétés contenant des secrets pré-chargés dans le filesystem d'un pod.

Bonne idée pour externaliser les secrets

 : Attention à la complexité des sidecar

 : Aucun développeur ne souhaite devoir exécuter un agent Vault sur son poste.



Utiliser un VPA/HPA en surveillant la RAM

Vertical & Horizontal Pod Autoscalers

💡 : c'est une bonne idée de les utiliser

✘ : N'utilisez jamais la `memory` comme métrique

⚠️ Vos ReplicaSet vont scaler au max et ne jamais redescendre !

On configure la JVM pour occuper tout l'espace disponible

La JVM ne rend (presque) jamais la mémoire qui n'est plus utilisée

Observer le CPU c'est déjà un bon début

```
hpa.yaml

apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: incom
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: incom
  minReplicas: 1
  maxReplicas: 10
  metrics:
  - type: Resource
    resource:
      name: cpu
      target:
        type: Utilization
        averageUtilization: 80
```




Utiliser le tag 'latest'





Une version toujours à jour ?

? Comment on rollback une image qui s'appelle `latest`

✗ On peut pas !

```
kubectl rollout undo deployment/incom-deployment
```



imagePullPolicy: IfNotPresent

Économiser un pull

💡 une bonne idée si on est rigoureux dans la gestion du versionning des images.

🔊 “Mon image ne se met pas à jour”

- Un développeur triste 😞 qui a publié une nouvelle image en écrasant le numéro de version.

❌ Ne cherchez pas à économiser l'exécution d'un `image pull`
Si les layers sont les mêmes (même hash), l'image ne sera pas re-téléchargée

💡 Utilisez `imagePullPolicy: Always`

```
pod.yaml

spec:
  containers:
  - name: app
    image: incom:1.2.0
    imagePullPolicy: Always
```



Laisser la sécurité au niveau du Dockerfile

securityContext et autres trucs rigolos 🗝️

🔊 “J’ai déclaré un user dans mon Dockerfile” - Un dev sérieux

🔊 “Ouais, mais peut être que tout le monde le fait pas” - Un ops sérieux

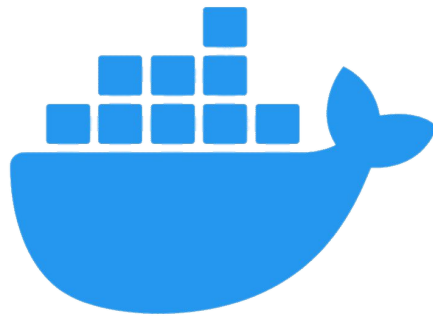
✅ `securityContext` dans les déploiements k8s pour forcer un user non-root, un filesystem en read-only, et empêcher les élévations de privilèges:

```
deployment.yaml

spec:
  securityContext:
    runAsNonRoot: true
    readOnlyRootFilesystem: true
    allowPrivilegeEscalation: false
```

42

C'est le nombre de bonnes pratiques qu'on a vu ensemble



Learn you own path

“R2-D2, you know better than to trust a strange speaker (or ChatGPT).”

– C3PO, The Empire Strikes Back (1980)



“Questions ?”





🙏 Merci ❤️

Follow me on :

 @CodeKaio

 julien-wittouck

