JAVA USER GROUP CH

# Master your Java applications in Kubernetes

04.2019

Andy Moncsek

# About me

ad cubum

- Andy Moncsek → Architect
- Creator or… see my Github
- Likes coffee & HiFi & cameras
- Twitter: @AndyAHCP

# Agenda

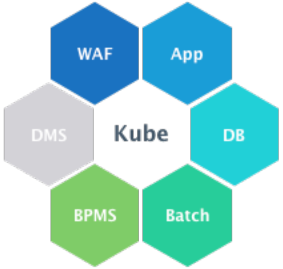
ad cubum

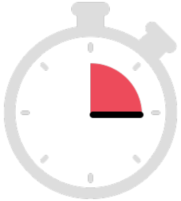
- Choose your (Java) Runtime
- Build & execute your applications
- Create your image
- Run your applications in Kubernetes
- Final thoughts

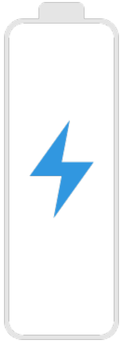
# Typical issues

You plan to move to Kubernetes?

- How to integrate?
- Slow startup?
- No more capacity?







ad cubum

think.insurance

# Choose your (Java) Runtime

# Choose your (Java) Runtime

- Support?
- License & LTS?
- Container aware?
  - since Java SE 8u131 & JDK 9
  - changes in JDK 8u191 & JDK 10

# Many (possible) options, out there

Choose your (Java) Runtime

Hotspot + C1 & C2 Jit

OpenJ9

GraalVM™ + Substrate VM

OpenJDK™

Java™ ORACLE®

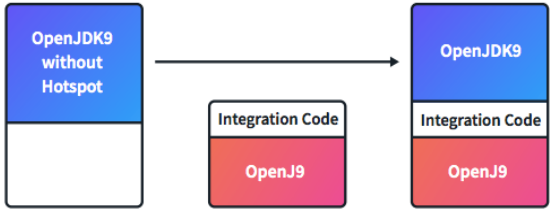
+ Graal

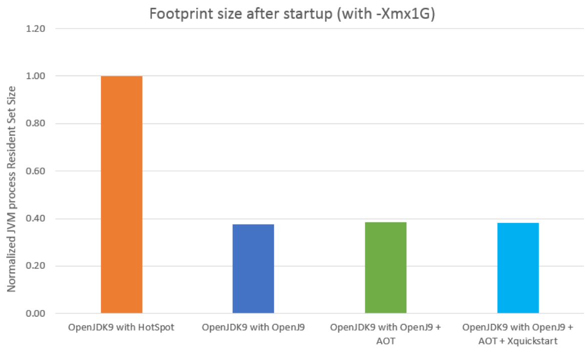
Hotspot

# OpenJ9

Choose your (Java) Runtime

- Contributed by IBM to the Eclipse Foundation in 2017
- It replaces HotSpot JVM in the OpenJDK build
- Small memory footprint & fast startup
- Optimization for virtualized environments



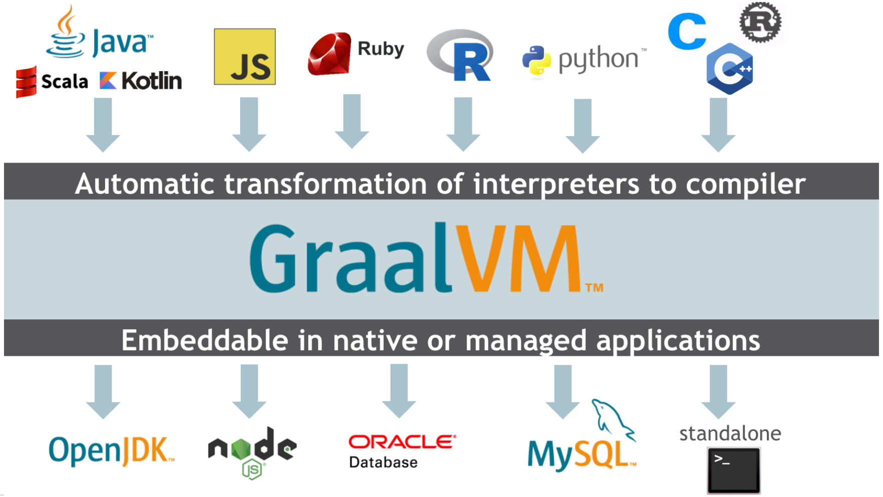


# GraalVM

Choose your (Java) Runtime

GraalVM™

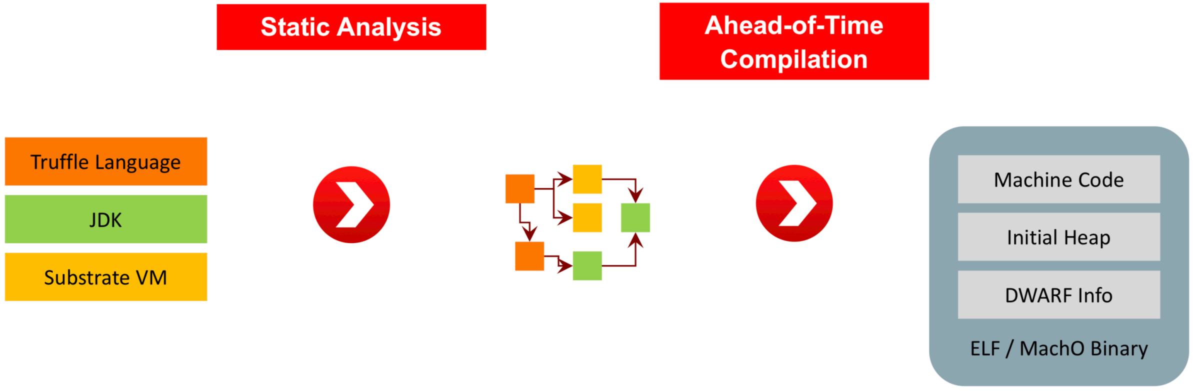
- Universal VM running various languages
- Removes isolation & enables interoperability between programming languages
- Can be integrated in various native & managed env. (OpenJDK, Node.js, OracleDB, MySQL,…)
- The Graal compiler
  - as JIT compiler since Java 10
  - as AOT compiler since Java 9


Java
Scala
Kotlin
JS
Ruby
R
python
C
Automatic transformation of interpreters to compiler
GraalVM
Embeddable in native or managed applications
OpenJDK
node
ORACLE Database
MySQL
standalone

# Substrate VM (SVM)

Choose your (Java) Runtime

## Substrate VM: Execution Model



All Java classes from Truffle language (or any application), JDK, and Substrate VM

Reachable methods, fields, and classes

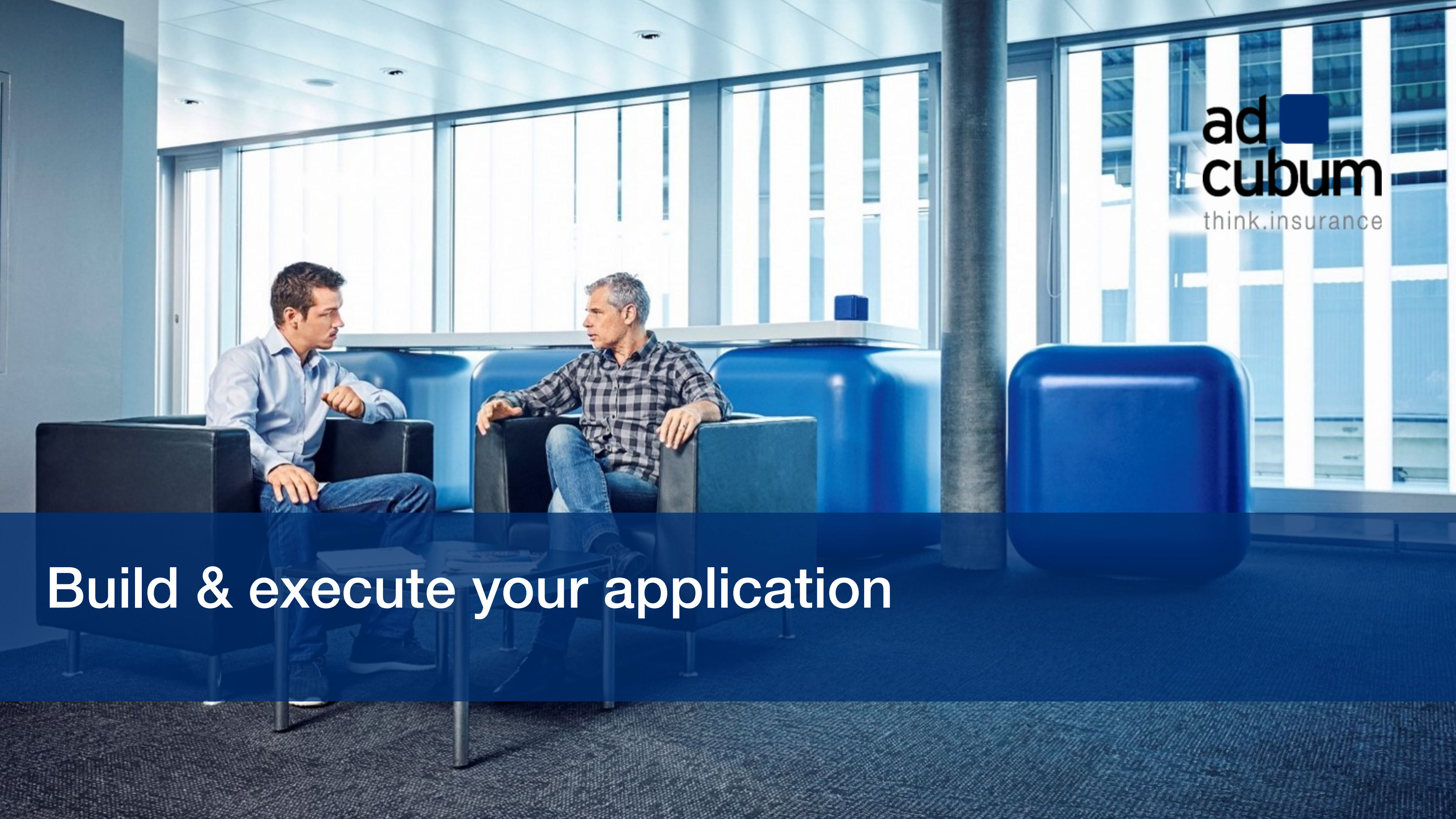
Application running without dependency on JDK and without Java class loading

https://www.oracle.com/technetwork/java/jvmls2015-wimmer-2637907.pdf

# Relation to Containers / Kubernetes?

Choose your (Java) Runtime

- JVM needs to be aware of containers (CPU & memory)
- Small memory/image footprint (run & deploy many containers)
- Fast startup time (auto scaler, elastic)



ad cubum
think.insurance

# Build & execute your application

# “Basic” container specific flags

Build & execute your application

Default max heap size ~1/4 of physical memory

- docker container run -it -m512M --entrypoint bash openjdk:8u151-jdk

```
MaxHeapSize                              := 4202692608
openjdk version "1.8.0_151"
```

VS.

- docker container run -it -m512M --entrypoint bash openjdk:8u191-jdk

```
MaxHeapSize                              := 134217728
openjdk version "1.8.0_191"
```

# “Basic” container specific flags

Build & execute your application

- **-XX:[+|-]UseContainerSupport**
  - Correct CPU count & total memory allocated to the container
- **-XX:InitialRAMPercentage**
  - Set initial heap size as a percentage of total memory (-Xms)
- **-XX:MaxRAMPercentage** & **-XX:MinRAMPercentage**
  - used to calculate *maximum heap size (-Xmx)*

phys_mem * MinRAMPercentage / 100  (if this value is less than 96M)

MAX(phys_mem * MaxRAMPercentage / 100, 96M)

# OpenJ9 specific container flags

Build & execute your application

- **-Xtune:virtualized**
  - Tuning for containers
  - Reduction in footprint & startup time (but also in throughput)
  - Enables VM idle management
- **-Xquickstart**
  - Designed for the fastest start-up
  - Ideal for short-lived tasks
  - May limit peak throughput

# Resident Set Size (RSS)

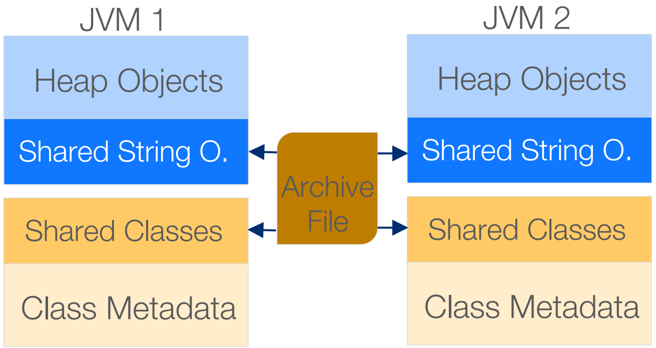
Build & execute your application

- RSS → amount of physical memory allocated & used by a process
- Java MaxHeapSize != Docker stats (“MEM USAGE”)
  - Java ~= heap + metaspace + off-heap
    (DirectBuffer + threads + compiled code +  GC data)

# AppCDS

Build & execute your application

- Since JDK10 (JEP310)
- Sharing of classes **loaded by the application class loader**
- Still some limitations (since Java 11 support for module path)
- Flag **UseAppCDS** (introduced in Java 10) removed in Java12
  - Automatically enabled in Java 12
- Reduce memory footprint/startup time
- Needs two preparation steps



# AppCDS Usage

Build & execute your application

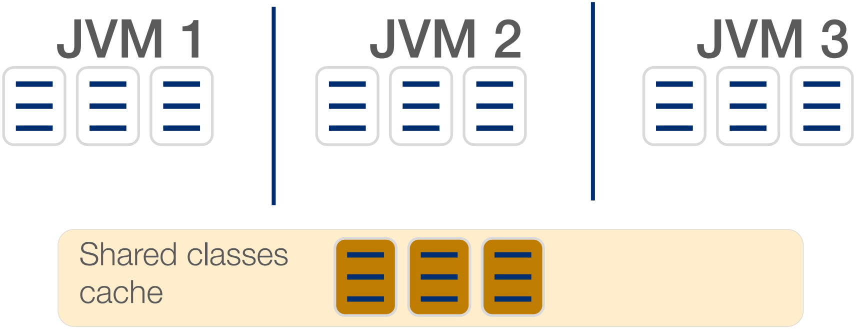
```
// step1: run the app & record all classes
java -XX:+UseAppCDS -XX:DumpLoadedClassList=classes.lst -jar \
app.jar

// step 2: create an archive
java -XX:+UseAppCDS -Xshare:dump -XX:SharedClassListFile=classes.lst \
-XX:SharedArchiveFile=cds.jsa --class-path app.jar

// step 3: start/use your application
java -XX:+UseAppCDS -Xshare:on -XX:SharedArchiveFile=cds.jsa -jar \
app.jar
```

# CDS & AOT in OpenJ9

Build & execute your application

- Enable class data sharing → AOT compilation is also enabled by default
  - dynamically compiles certain methods into AOT code at runtime
  - applicable to boot, extension, & application loaders & all URLClassloader-subclasses

- **-Xshareclasses** option to enable class sharing & AOT
- **-Xshareclasses:cacheDir=/opt/shareclasses**



# Ahead-of-time (AOT)

Build & execute your application

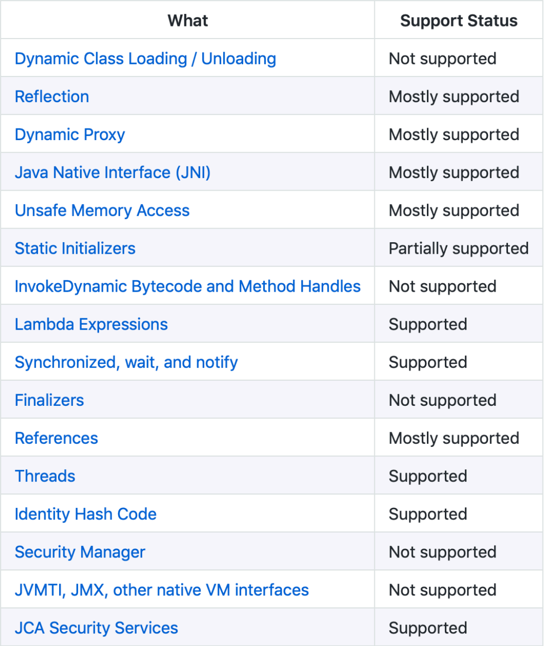
- **AOT compilation** (since JDK9 / JEP 295)
  - Transforms Java bytecode to OS-specific machine code
    - Performs simple optimizations of Java bytecode
- **AOT vs. JIT compiled** (rule of thumb)
  - **Pro**: better startup time
  - **Cons**: worse performance of long-running applications

```
> jaotc --output app.so --jar microservice.jar --module jdk.httpserver --module java.base
> java -XX:AOTLibrary=./app.so -jar microservice.jar
```

# Native compilation

Build & execute your application

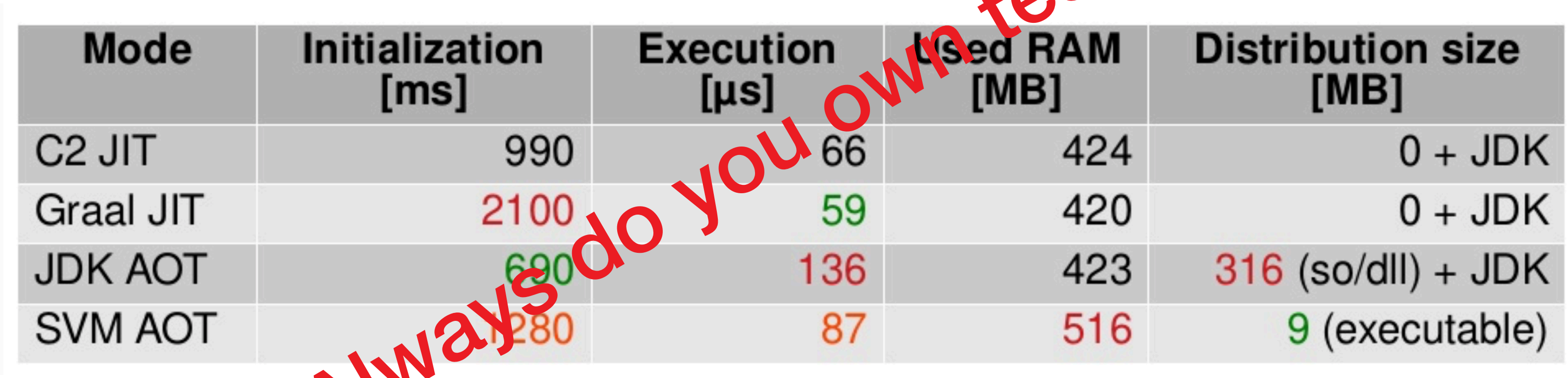
- Native compilation (in a nutshell)
  - Based on Graal compiler & SubstrateVM
  - Still many limitations (class loading, reflection,… )
- How to use
  - Download GraalVM & Build your (fat) jar
  - `native-image -jar app.jar && ./app`
- Working frameworks?
  - Micronaut, Spark Java, Vert.x



| What | Support Status |
|---|---|
| Dynamic Class Loading / Unloading | Not supported |
| Reflection | Mostly supported |
| Dynamic Proxy | Mostly supported |
| Java Native Interface (JNI) | Mostly supported |
| Unsafe Memory Access | Mostly supported |
| Static Initializers | Partially supported |
| InvokeDynamic Bytecode and Method Handles | Not supported |
| Lambda Expressions | Supported |
| Synchronized, wait, and notify | Supported |
| Finalizers | Not supported |
| References | Mostly supported |
| Threads | Supported |
| Identity Hash Code | Supported |
| Security Manager | Not supported |
| JVMTI, JMX, other native VM interfaces | Not supported |
| JCA Security Services | Supported |

# Some benchmarks ;-) from October 2018 / Java 10

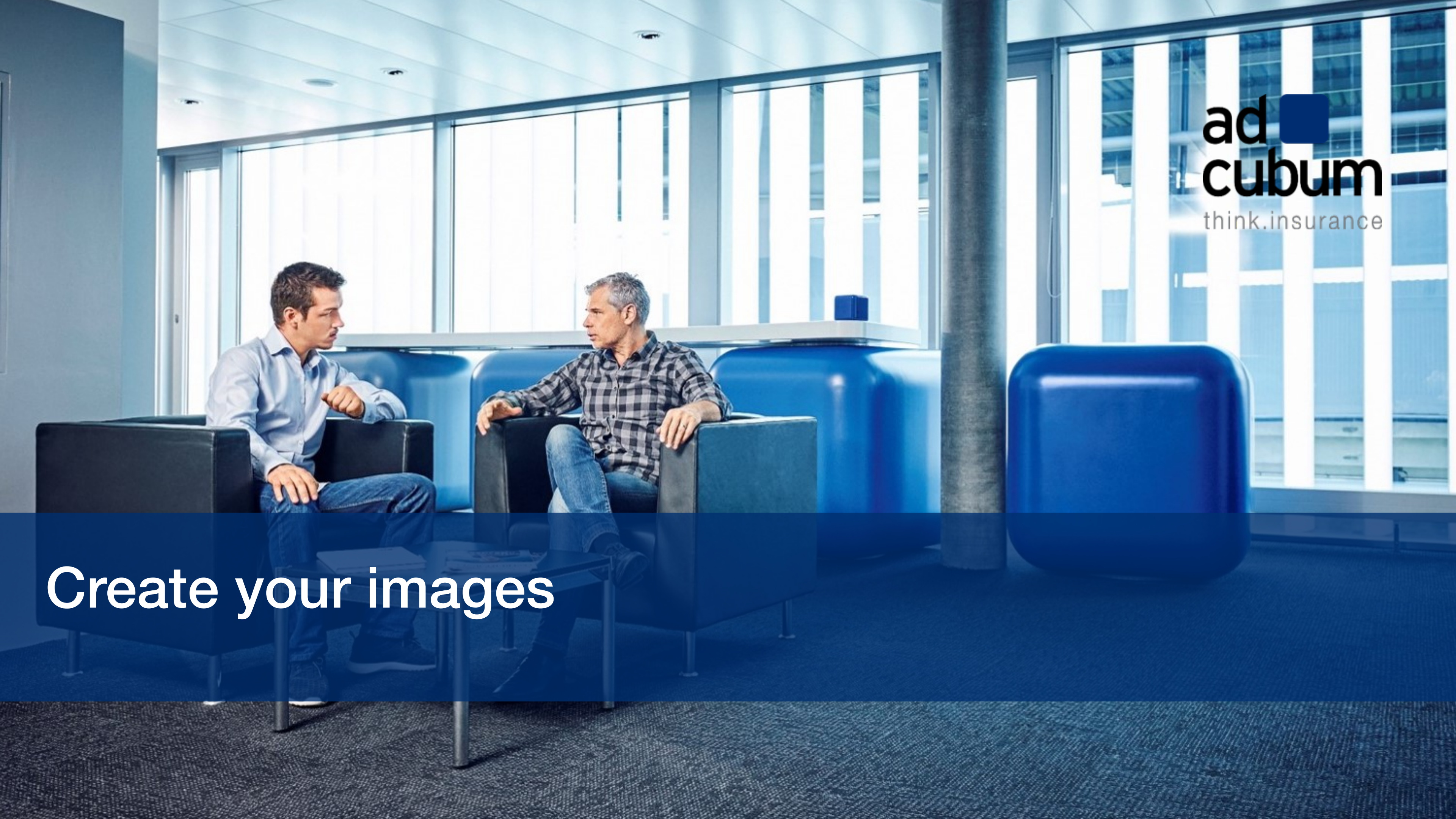
Build & execute your application



| Mode | Initialization [ms] | Execution [µs] | Used RAM [MB] | Distribution size [MB] |
|---|---|---|---|---|
| C2 JIT | 990 | 66 | 424 | 0 + JDK |
| Graal JIT | 2100 | 59 | 420 | 0 + JDK |
| JDK AOT | 690 | 136 | 423 | 316 (so/dll) + JDK |
| SVM AOT | 1280 | 87 | 516 | 9 (executable) |

Always do you own tests!

https://www.slideshare.net/trivadis/techevent-graalvm-performance-interoperability



# Create your images

# Usage of modular run-time images & multi-stage builds

Create your image

- Shrink your image to a minimum
- Works also with non-modular Java applications (like spring-boot)

```
FROM openjdk:12-ea-jdk-alpine3.8 as builder
RUN jlink \ // works for spring-boot
    --add-modules java.sql, java.naming, java.net.http, java.management,
      java.instrument,java.security.jgss,java.desktop,jdk.unsupported \
    --verbose --strip-debug --compress 2 --no-header-files \
    --no-man-pages --output /opt/jre-minimal

FROM alpine:3.8
COPY --from=builder /opt/jre-minimal /opt/jre-minimal
PATH=${PATH}:/opt/jre-minimal/bin
. . .
CMD
```

# Create lean images

Create your image

```
# Do
RUN apt-get update && \
    apt-get install package bar
# Don’t
RUN apt-get update
RUN apt-get install package bar
```

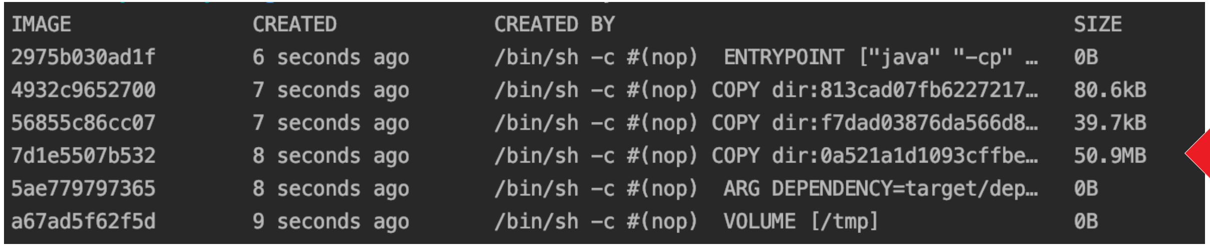
- Use multistage builds, if needed
- Split up layers, based on their potential for reuse
- Put any components that **will update very rarely at the top of the Dockerfile**
- Merge commands together, because **each RUN line adds a layer to the image**
- **Advantage:** faster builds, less storage, pull faster

# Create lean images

Create your image

```
FROM adoptopenjdk/openjdk11-openj9:alpine-slim
RUN mkdir -p /usr/src/app && mkdir -p /usr/src/app/config
ARG APPLICATION=target/application
COPY ${APPLICATION}/BOOT-INF/lib /app/lib
COPY ${APPLICATION}/META-INF /app/META-INF
COPY ${APPLICATION}/BOOT-INF/classes /app
ENTRYPOINT ["java","-cp","app:app/lib/*:/usr/src/app/config",»com.*.KafkaAdapterMain"]
```

(reuse) dependencies

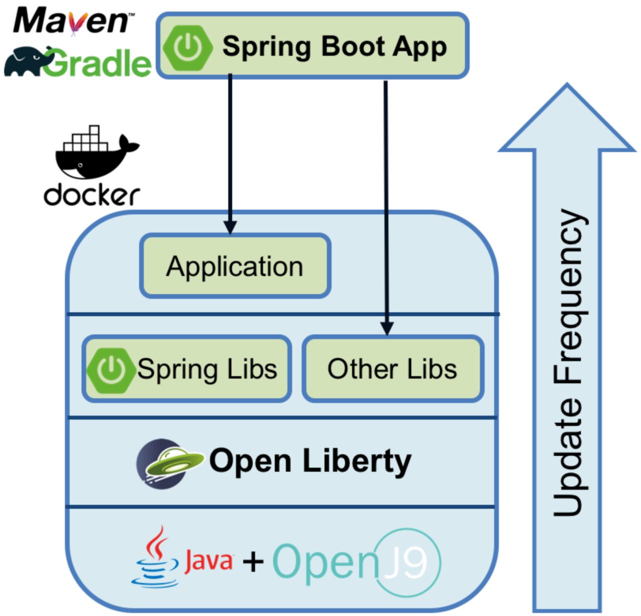


```
IMAGE           CREATED         CREATED BY                                       SIZE
2975b030ad1f    6 seconds ago   /bin/sh -c #(nop)  ENTRYPOINT ["java" "-cp" …    0B
4932c9652700    7 seconds ago   /bin/sh -c #(nop) COPY dir:813cad07fb6227217…    80.6kB
56855c86cc07    7 seconds ago   /bin/sh -c #(nop) COPY dir:f7dad03876da566d8…    39.7kB
7d1e5507b532    8 seconds ago   /bin/sh -c #(nop) COPY dir:0a521a1d1093cffbe…    50.9MB
5ae779797365    8 seconds ago   /bin/sh -c #(nop)  ARG DEPENDENCY=target/dep…    0B
a67ad5f62f5d    9 seconds ago   /bin/sh -c #(nop)  VOLUME [/tmp]                 0B
```

# Think about your layers

Create your image

- Try to **avoid “fat” jars & wars**
  - Use skinny / thin
- Different approaches for:
  - Tomcat, Open Liberty, WildFly, Spring
- Spring
  - Maven dependency plugin
  - Open Liberty boost plugin
  - spring-boot-thin-launcher



Dual Layer with Liberty & Open J9

https://openliberty.io/blog/2018/07/02/creating-dual-layer-docker-images-for-spring-boot-apps.html

# Choose your image

Be frugal

| Image type | Red Hat Enterprise 7 Standard | Red Hat Enterprise Atomic | Debian Stable | Alpine |
|---|---|---|---|---|
| C Library | glibc | glibc | glibc | musl c |
| **Size on Disk** | **200MB** | **78MB** | **100MB** | **4MB** |

- Look for vendor-specific (or official) images
- Don’t use :latest or no tag!
- Do you really need an application server?

# Run your application (in Kubernetes)

# Share your CDS files in Kubernetes

Run your application

```
apiVersion: v1
kind: Pod
metadata:
   name: myApp
spec:
   containers:
   - image: myApp
     name: myApp
     volumeMounts:
      - mountPath: /cdscache
        name: cdscache
   volumes:
   - name: cdscache
     hostPath:
         path: /cdscache  #location on host
         type: Directory
```

```
CMD java -jar -Xshareclasses:cacheDir=/cdscache app.jar
```

# Use resource request and limits

Run your application

- Each Container has a **request** of 0.25 cpu and 64MB of memory.
- Each Container has a **limit** of 0.5 cpu and 128MB of memory.

```
kind: Pod
metadata:
  name: frontend
spec:
  containers:
   - name: wp
    image: wordpress
    resources:
      requests:
          memory: "64M"
          cpu: "250m"
      limits:
          memory: "128M"
          cpu: "500m"
....
```

# Quotas

Run your application

- Apply **quotas on Namespace** level
  - Constraints on number of objects, cpu & memory
  - When enabled, resource limits must be set

```
kind: ResourceQuota
metadata:
  name: quota
spec:
  hard:
   cpu: "2"
   memory: 1G
   pods: "10"
   replicationcontrollers: " 5"
   resourcequotas: "1"
   services: "5"
```

# ConfigMaps & Secrets to externalize your configuration

Run your application

```
kind: ConfigMap
metadata:
  name: spring-prop
data:
  application.yml: |
        server:
         port: 8080
```

```
kind: Pod
spec:
  volumes:
    - name: config
      configMap:
        name: spring-prop
        items:
          - key: application.yml
            path: application-kube.yml
  containers:
    - volumeMounts:
        - name: config
          mountPath: /usr/src/app/config
```

ad cubum

think.insurance

# Final thoughts

# Final thoughts

Be inventive

- Avoid/reduce infrastructure code in your applications
  - Service mesh with circuit breaker
  - Service Discovery (using DNS or Labels in Kubernetes)
  - Configuration via ConfigMap
- Helm Charts to ship your application
  - Package manager like “apt” in Debian
  - Kubernetes Operators helps you to manage upgrades, lifecycle & insights

# Final thoughts

Be inventive

- Don’t overlook Serverless!!
    - Better utilization of your cluster
    - Many measured run-times (AWS, Serverless,..)
    - Knative pushed to Kubernetes (by Google & Pivotal)
- GraalVM and other projects focusing low footprint & fast startup

# Final thoughts

Be inventive

- **Why** you should **optimize** your applications **for containers** & cloud?
  - **Costs!!!!!** → smaller footprint, CPU cycles, pay-per-use
  - The application life-cycle has changed
    - no longer dominated by uptime
    - startup is now critical to your application
  - Expect to spend more and more time looking at resource usage, performance and footprint.

ad cubum

think.insurance

# Any questions?

# Links

- https://github.com/amoAHCP/JavaContainerTests
- https://www.slideshare.net/DanHeidinga/j9-under-the-hood-of-the-next-open-source-jvm
- https://github.com/eclipse/openj9-website/blob/master/benchmark/daytrader3.md
- https://static.rainfocus.com/oracle/oow18/sess/1525896302003001DAHT/PF/CodeOne_2018_hw_1540823752812001Nmsq.pdf
- https://hub.docker.com/r/adoptopenjdk/openjdk10-openj9
- https://www.eclipse.org/openj9/docs/xshareclasses/