

An aerial photograph of a city, likely Zurich, featuring a prominent glass skyscraper on the right side. The city is densely packed with buildings, and a large mountain range is visible in the background under a hazy sky. The overall tone is warm and slightly desaturated.

# Awesome Java applications with GraalVM and Java microservices frameworks

**Alina Yurenko**

Developer Advocate for GraalVM

Oracle Labs

Voxxed Days Zurich

by Patrick Federi @ Unsplash



\* Wenger Giant Swiss Army Knife, 141 Functions



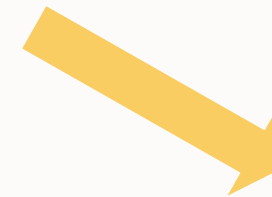


**GraalVM™**



**JIT**

`java MyMainClass`



**AOT**

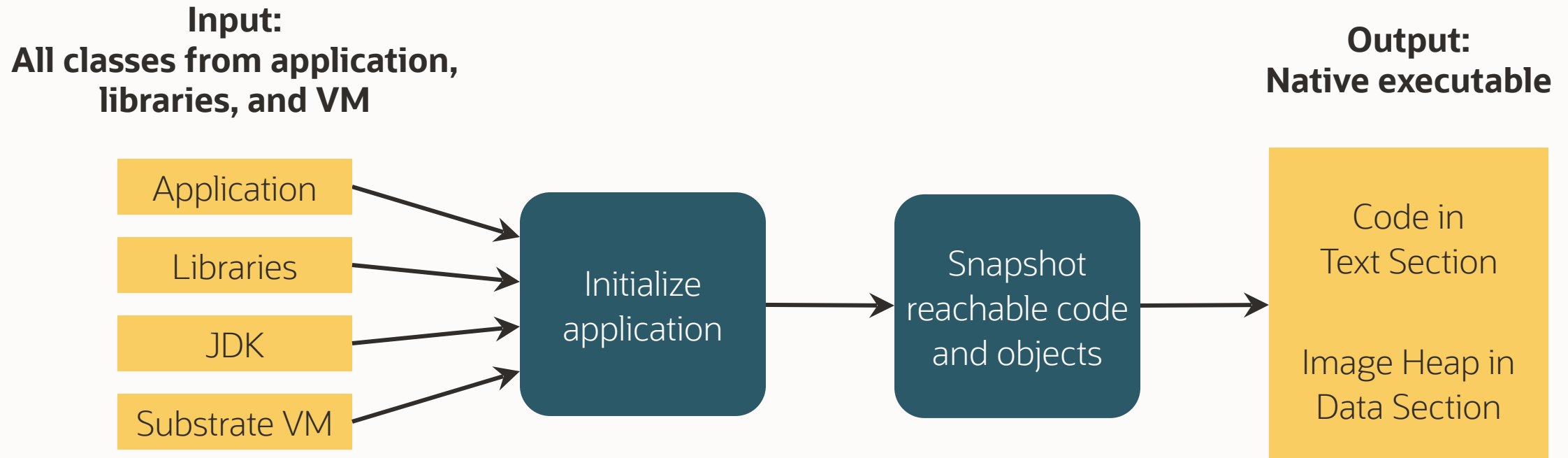


`native-image MyMainClass  
./mymainclass`

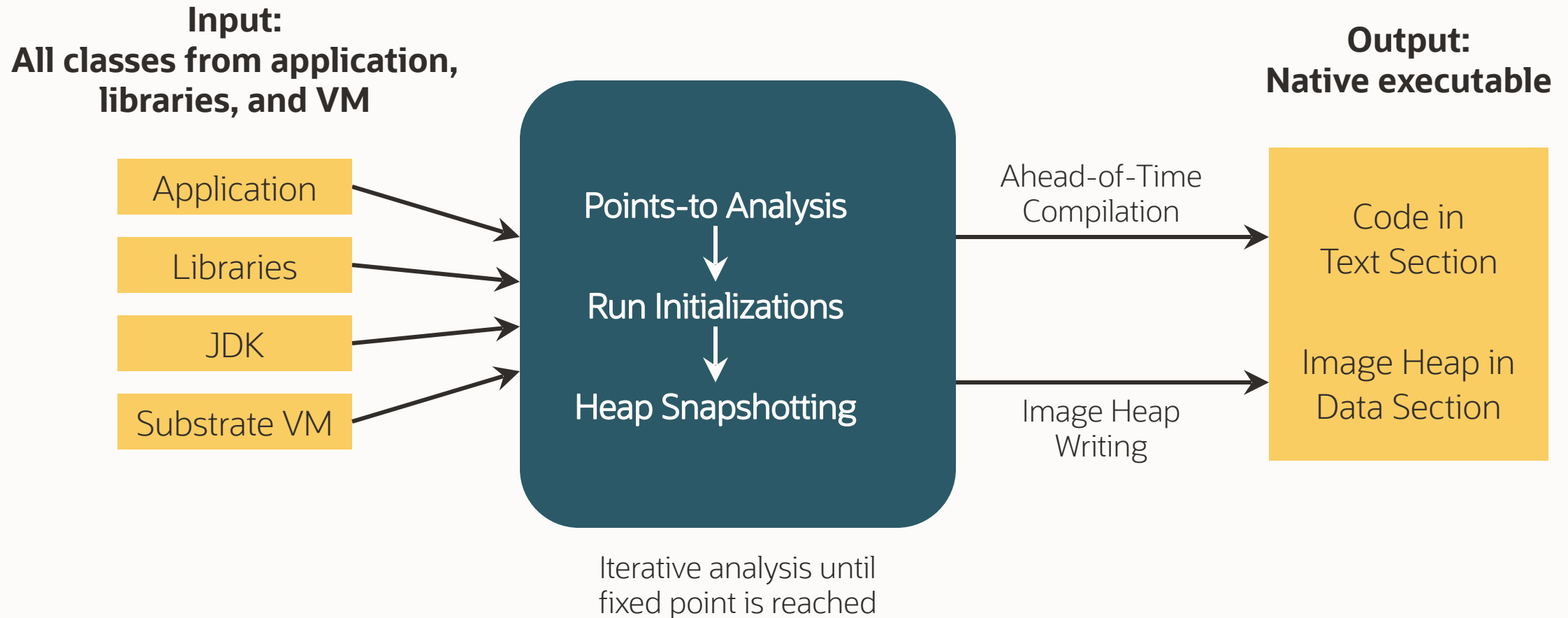
# **GraalVM**

## **Native Image**

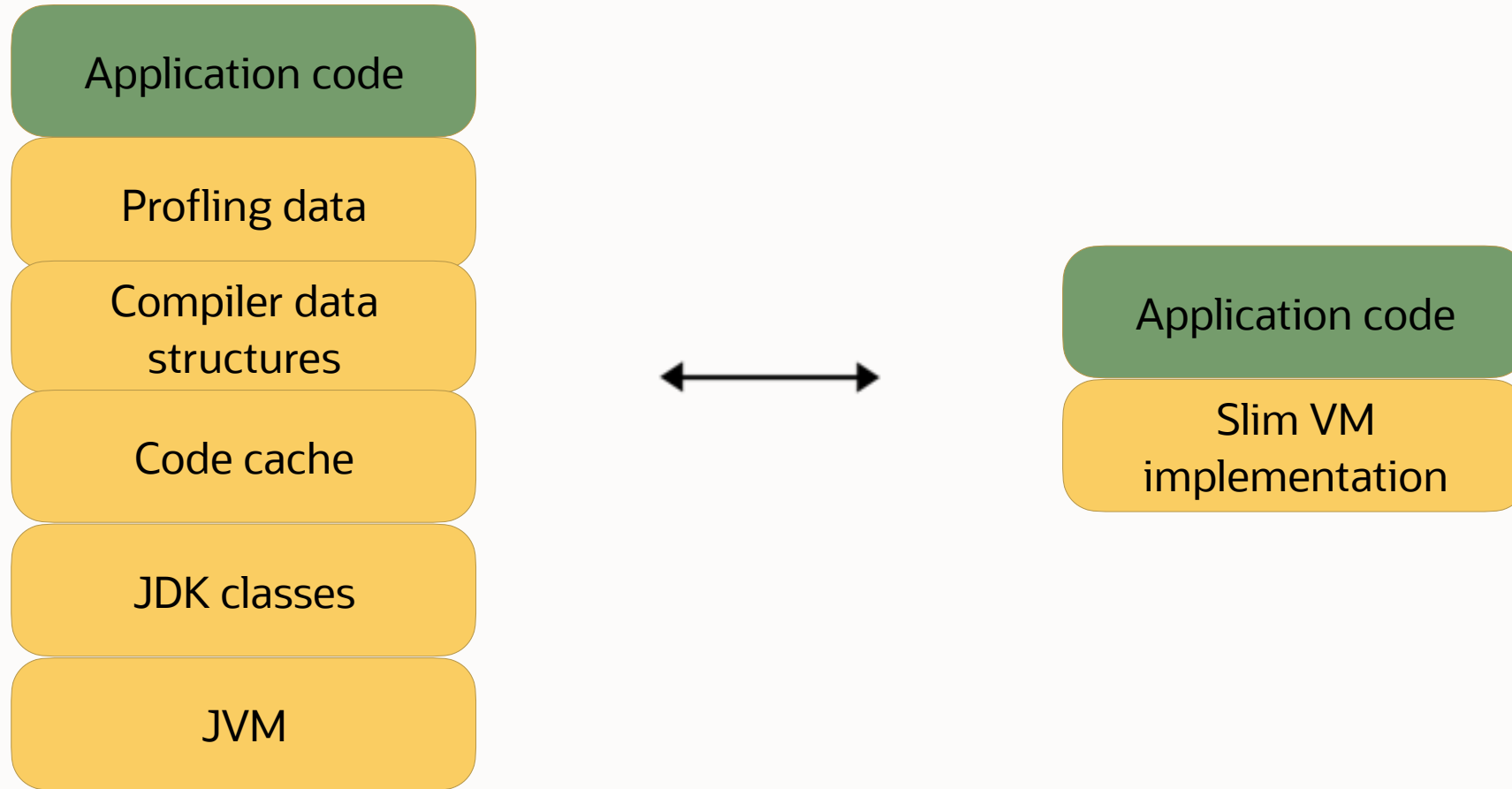
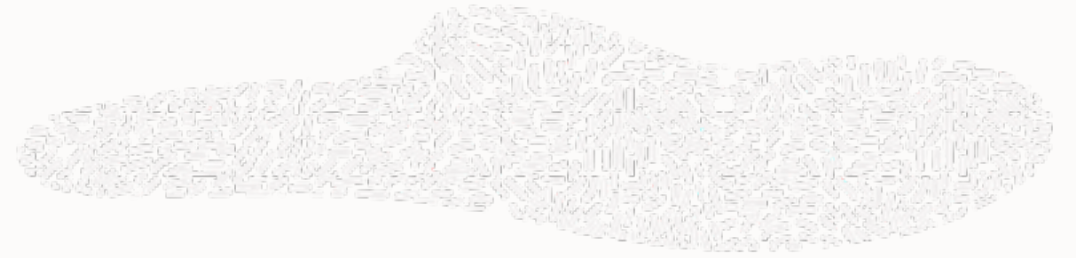
# Native Image Build Process: the idea



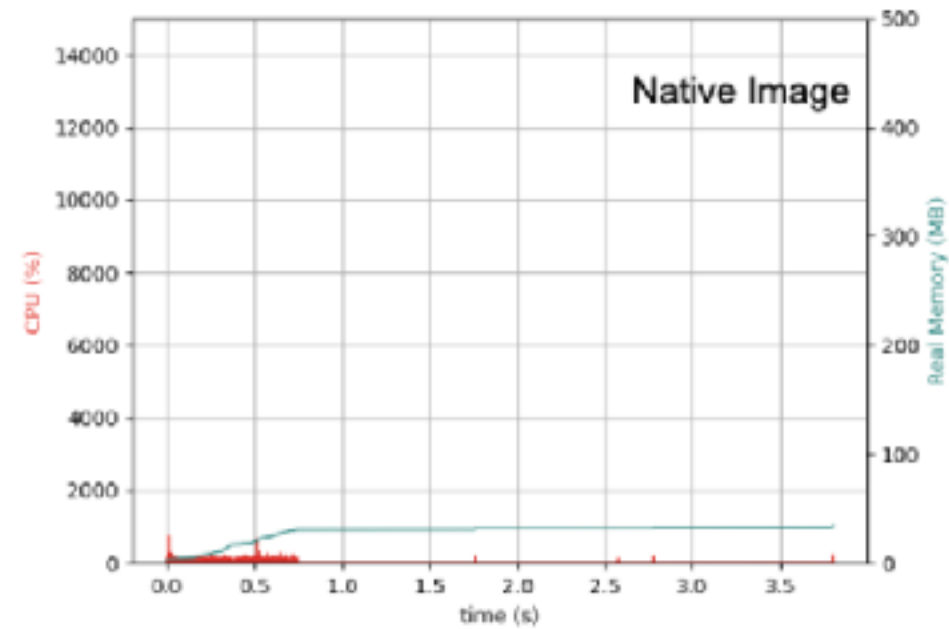
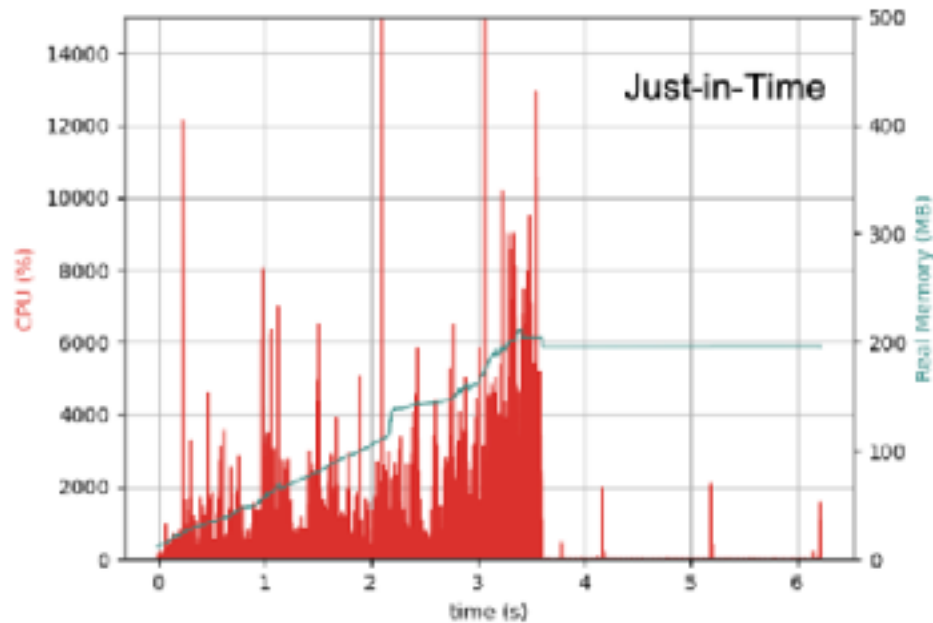
# Native Image Build Process



# Executing on the JVM vs Native Image



# CPU and memory usage of GraalVM Native Image applications

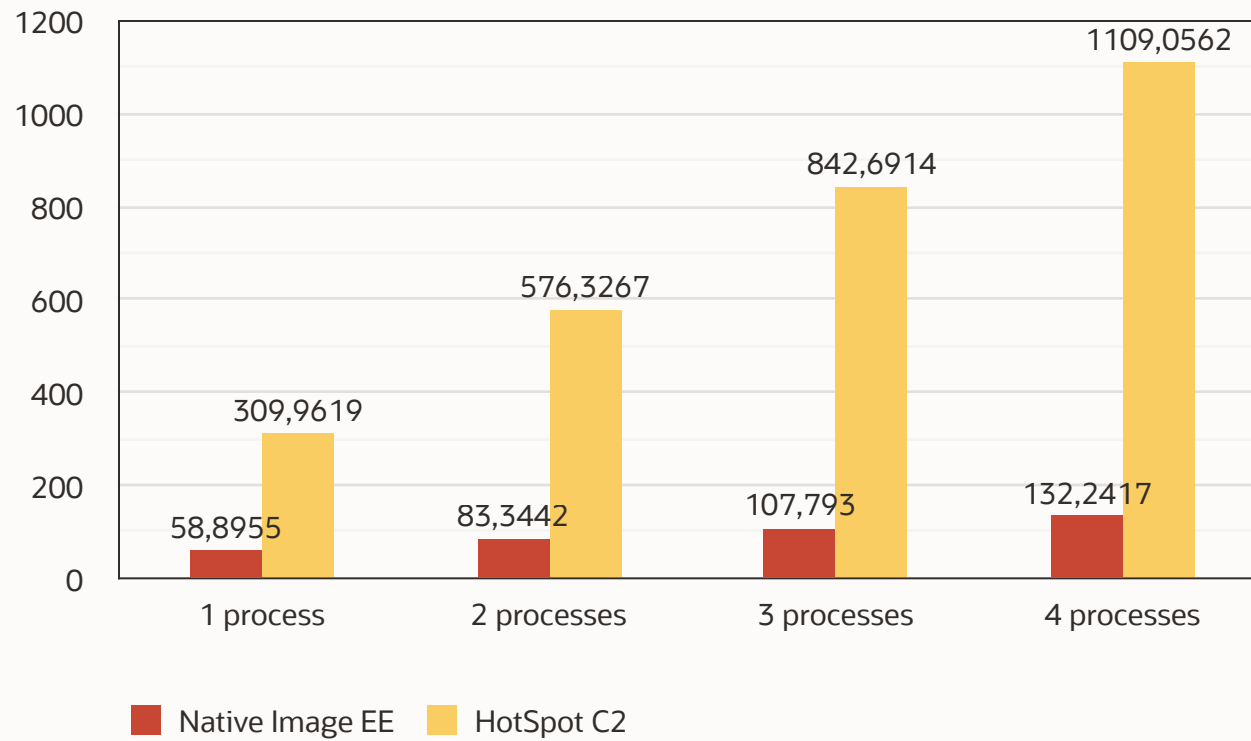




# Example: horizontal scaling of microservices

## Memory Usage in MB

Quarkus Apache Tika ODT in a “tiny” configuration and with the serial GC  
(1 CPU core per process, -Xms32m -Xmx128m) – JDK 11



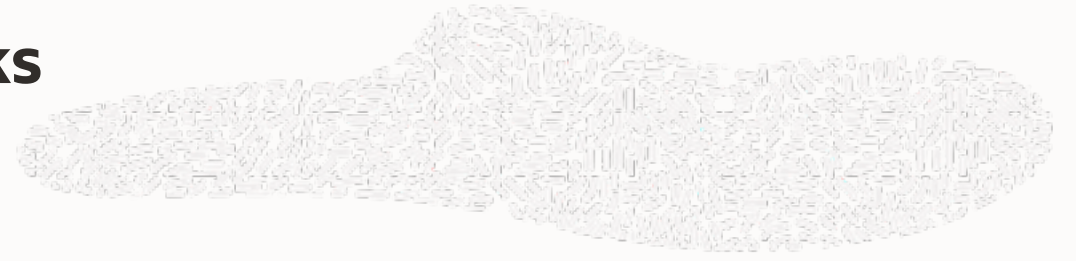
Java HotSpot VM

- **4 VM instances = 4 times the memory**

Native Image

- **4 VM instances = 2 times the memory**
- Image heap shared between processes
- Machine code shared between processes

## GraalVM 🤝 Java Microservice frameworks



# GraalVM™



spring

## Spring blog

[All Posts](#) [Engineering](#) [Releases](#) [News and Events](#) 

# Spring Boot 3.0 Goes GA


**RELEASES | ANDY WILKINSON | NOVEMBER 24, 2022 | 63 COMMENTS**

On behalf of the team, it is my very great pleasure to announce that Spring Boot 3.0 is now generally available and [3.0.0](#) can be found in Maven Central.

This release is the culmination of 12 months work and over [5700 commits](#) by 151 different individuals. A massive thank you to everyone that has contributed, and to all the early adopters that have been providing vital feedback on the milestones.

This is the first major revision of Spring Boot since 2.0 was released 4.5 years ago. It's also the first GA version of Spring Boot that provides support for [Spring Framework 6.0](#) and GraalVM.

Highlights of the new release include:

- A Java 17 baseline
- [Support for generating native images with GraalVM](#), superseding the experimental Spring Native project 
- Improved observability with Micrometer and Micrometer Tracing
- Support for Jakarta EE 10 with an EE 9 baseline

There's far too many features to list them all here in detail, so head over to the [release notes](#) page on our wiki to find out more. If you just want to get started, you can easily bootstrap a new project on [start.spring.io](#). If you'd like to try out the GraalVM support, [start.spring.io can help with that too](#).


Over the coming weeks we'll be publishing blog posts that cover some Spring Boot 3.0 features in detail.

← → ↻ ⌂

start.spring.io

🔖 ☆ 📄 ⚙️ 👤 ⋮

☰

 **spring initia**

Project

☒ Gradle - Groovy

☐ Gradle - Kotlin

☐ Maven

Spring Boot

☐ 3.1.0 (SNAPSHOT)

☐ 3.1.0

☐ 2.7.10 (SNAPSHOT)

☐ 2.7.9

Project Metadata

Group

com.example

Artifact

demo

Name

demo

Description

Demo project for...

Package name

com.example.dan

Packaging

☒ Jar

☐ War

Java

☒ 19

☐ 17

Web, Security, JPA, Actuator, Devtools...

Press ⌘ for multiple adds

DEVELOPER TOOLS

GraalVM Native Support

Support for compiling Spring applications to native executables using the GraalVM native-image compiler.

Spring Boot DevTools

Provides fast application restarts, LiveReload, and configurations for enhanced development experience.

Lombok

Java annotation library which helps to reduce boilerplate code.

Spring Configuration Processor

Generate metadata for developers to offer contextual help and "code completion" when working with custom configuration keys (ex.application.properties/.yml files).

WEB

Spring Web

Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

Spring Reactive Web

Build reactive web applications with Spring WebFlux and Netty.

Spring for GraphQL

Build GraphQL applications with Spring for GraphQL and GraphQL Java.

Rest Repositories

Exposing Spring Data repositories over REST via Spring Data REST

ADD DEPENDENCIES... ⌘ - B

🔧

🌙

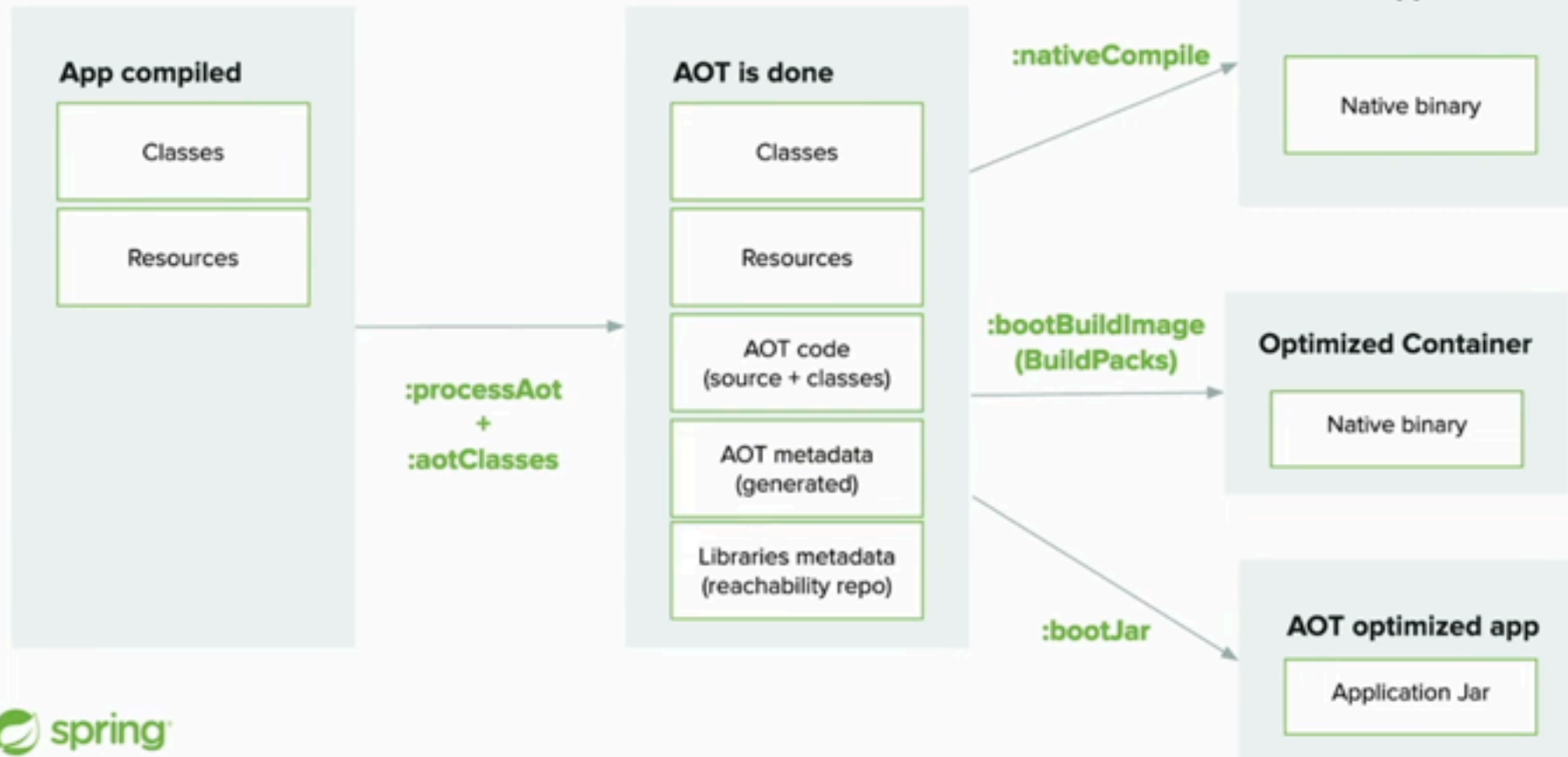
🔄

🐦





# AOT processing in Spring Boot 3.0



# Spring's AOT processing

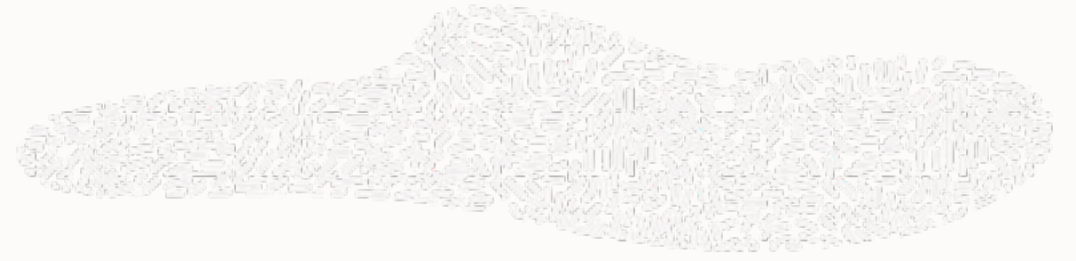


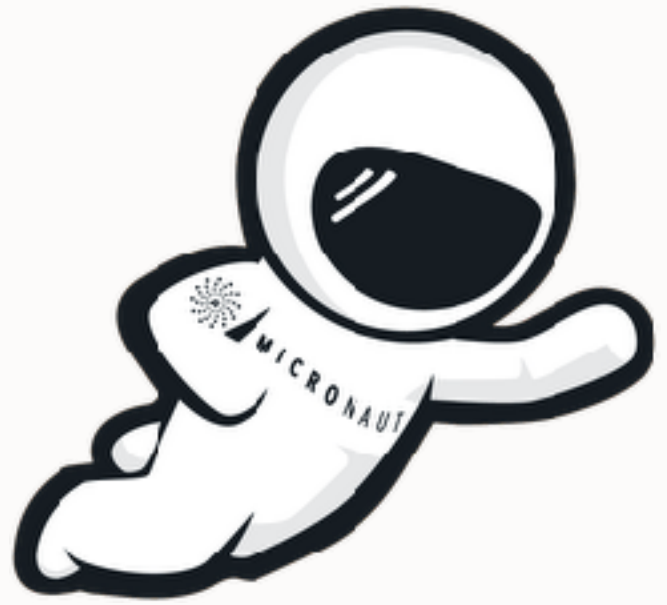
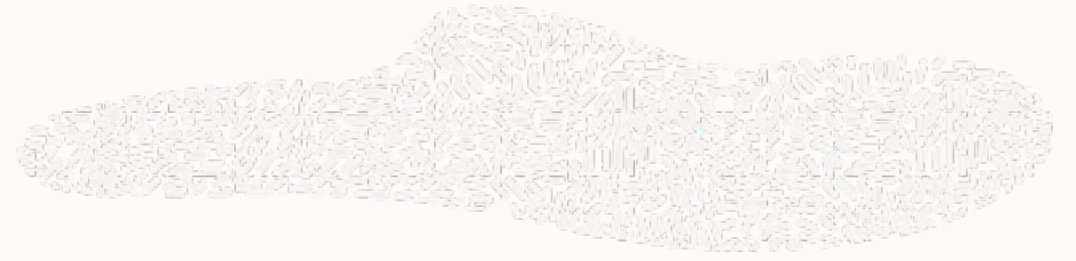
- The classpath is fixed and fully defined at build time
- The Spring @Profile annotation and profile-specific configuration is not supported

A Spring AOT processed application will typically generate:

- Java source code
- Bytecode (for dynamic proxies etc)
- GraalVM JSON hint files:
  - Resource hints (resource-config.json)
  - Reflection hints (reflect-config.json)
  - Serialization hints (serialization-config.json)
  - Java Proxy Hints (proxy-config.json)
  - JNI Hints (jni-config.json)

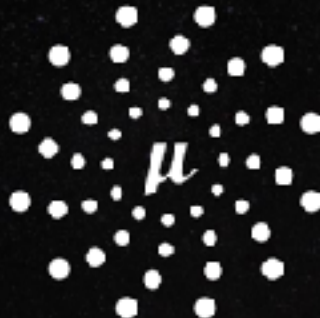
# Spring Boot demo





M I C R O N A U T <sup>TM</sup>

Features a Java annotation processor that hooks into your compiler and computes your framework infrastructure at compilation time eliminating reflection, runtime proxies and runtime code generation.



MICRONAUT<sup>™</sup>





**Cédric Champeau** → @melix@mastodon.xyz  
@CedricChampeau



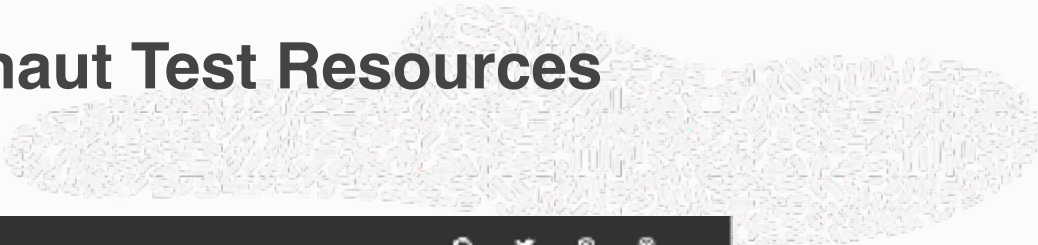
👨‍🚀 #Micronaut

```
      _
    _-_-_-_-_|_|
   /  _  | | | | _|
  | (| | | | | | |
  \_, _\_, _\_|
```

```
micronaut.runtime.Micronaut - Startup completed in 6ms. Server
[3s]
```

2:55 PM · Sep 15, 2021

# Testing Native Image applications: Micronaut Test Resources



Cédric Champeau's blog   About   Projects   Astronomy   Topics ▼   Feed

## Introducing Micronaut Test Resources

04 August 2022

Tags: [micronaut](#) [testcontainers](#) [docker](#) [test](#) [testing](#)

The new [release of Micronaut 3.6](#) introduces a new feature which I worked on for the past couple of months, called [Micronaut Test Resources](#). This feature, which is inspired from [Quarkus' Dev Services](#), will greatly simplify testing of Micronaut applications, both on the JVM and using GraalVM native images. Let's see how.

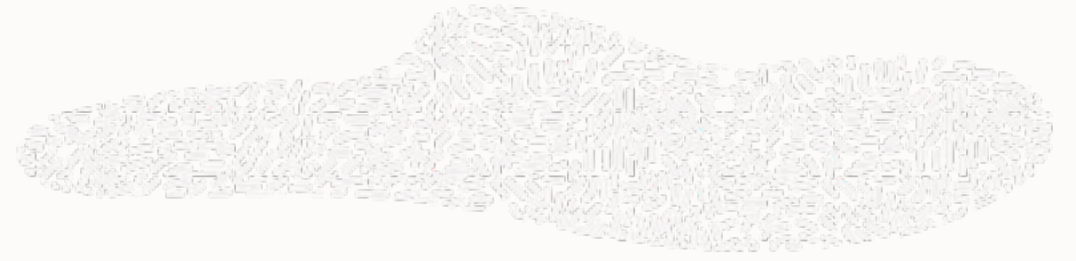
### Test resources in a nutshell

[Micronaut Test Resources](#) simplifies testing of applications which depend on external resources, by handling the provisioning and lifecycle of such resources automatically. For example, if your application requires a MySQL server, in order to test the application, you need a MySQL database to be installed and configured, which includes a database name, a username and a password. In general, those are only relevant for production, where they are fixed. During development, all you care about is having one database available.

Here are a couple of traditional solutions to this problem:

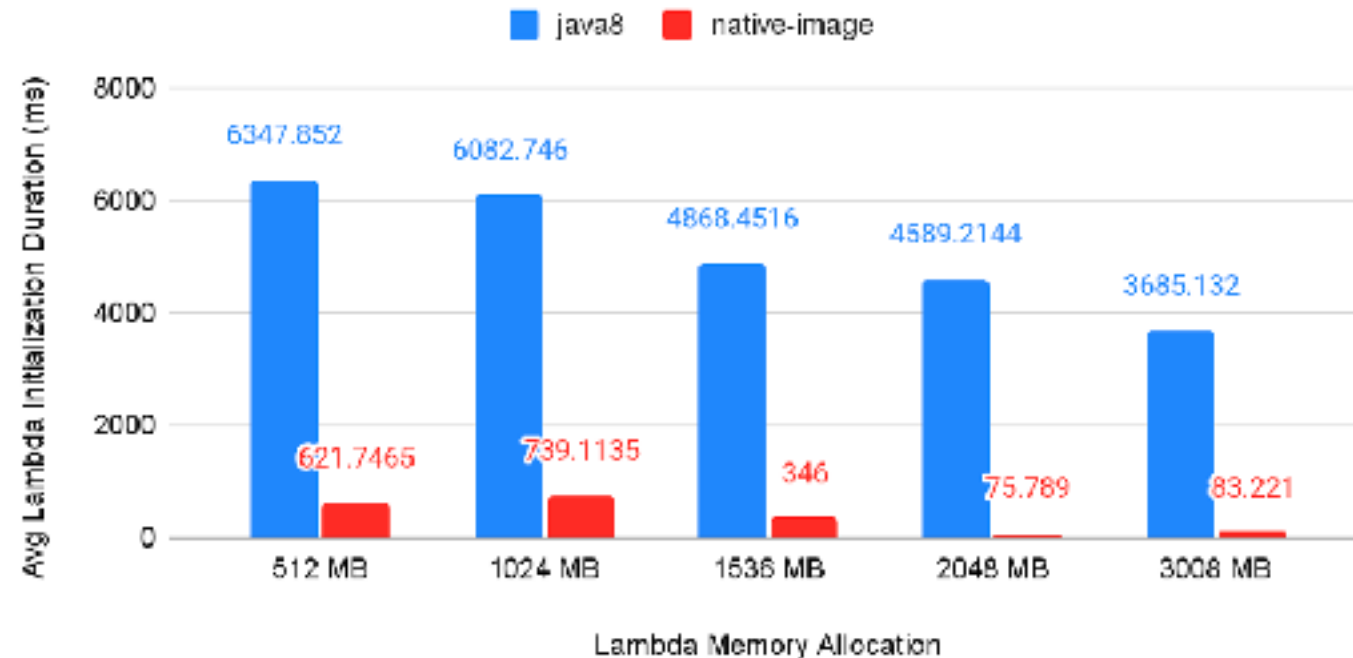
1. document that a MySQL server is a pre-requisite, and give instructions about the database to create, credentials, etc. This can be simplified by using Docker containers, but there's still manual setup involved.
2. Use a library like [Testcontainers](#) in order to simplify the setup

# Micronaut demo



# Native Image for serverless Java workloads

Avg Cold Start Initialization Duration Decreases With Native Image



“The same function with 3008 MB of memory that took 3.6 seconds to start with the JVM, started in under 100 milliseconds once compiled to a native executable using GraalVM’s native-image tool”

Disney Streaming: <https://aws.amazon.com/blogs/opensource/improving-developer-productivity-at-disney-with-serverless-and-open-source/>

# Micronaut session @ Voxxed Days Zurich



## Unleash the power of your applications with Micronaut and GraalVM

CONFERENCE *(BEGINNER LEVEL)*

Thursday from 14:20 – 15:05


Room 3

Login

In this talk, Micronaut committer Álvaro Sánchez-Mariscal, will demonstrate how you can quickly build optimised Microservices with Micronaut & GraalVM Native Image. Attendees will learn how the combination of GraalVM Native Image and Micronaut can lead to efficient, highly performant, and optimised applications that can be perfectly deployed to environments like Kubernetes or serverless platforms.

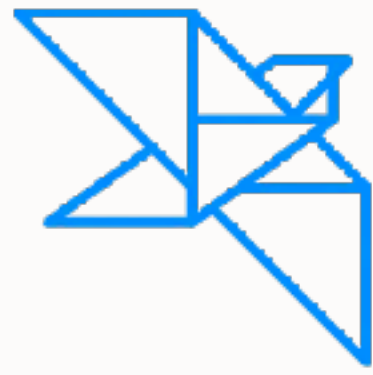
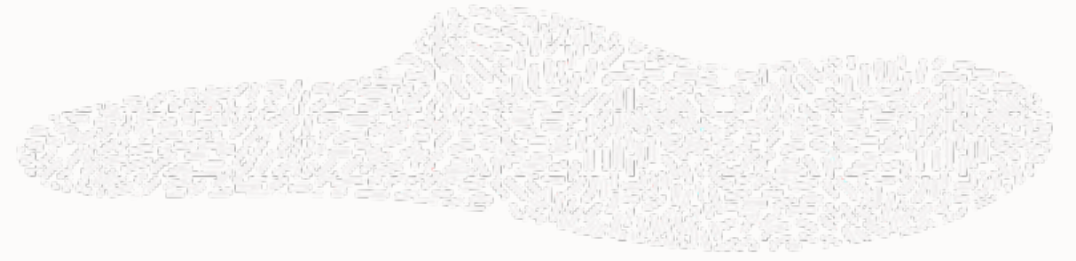
There will be a live coding demo of an application using Micronaut Data and GraalVM.



ÁLVARO SÁNCHEZ-MARISCAL   
Oracle

Álvaro is a passionate developer and agile enthusiast with over 21 years of experience. He is now a Principal Member of Technical Staff at Oracle Labs, where he is a Micronaut committer, helping to maintain and evolve the open-source framework.





**helidon.io**

Helidon SE	Helidon MP
Gives full transparency and puts you in control	Built on top of the Helidon SE libraries and provides a platform that is familiar to enterprise Java developers
Microframework model with a very small footprint and limited functionality (~7 MB)	MicroProfile implementation; slightly larger footprint than SE (~13 MB)
Functional style, reactive, non-blocking	Declarative style with dependency injection
Transparent "no magic" development experience; pure java application development with no annotations and no dependency injections	Developer experience similar to that of Spring Boot, Jakarta EE and MicroProfile; layers on some Jakarta EE components (CDI, JAX-RS, JSON-P, JSON-B).

# A RESTful service with Helidon SE and Helidon MP

```
Routing routing = Routing.builder()
    .get("/hello",
        (req, res) -> res.send("Hello World"))
    .build();

WebServer.create(routing)
    .start();
```

```
@Path("hello")
public class HelloWorld {
    @GET
    public String hello() {
        return "Hello World";
    }
}
```

# Helidon 🤝 GraalVM

## Memory Footprint



## Disk Space Usage



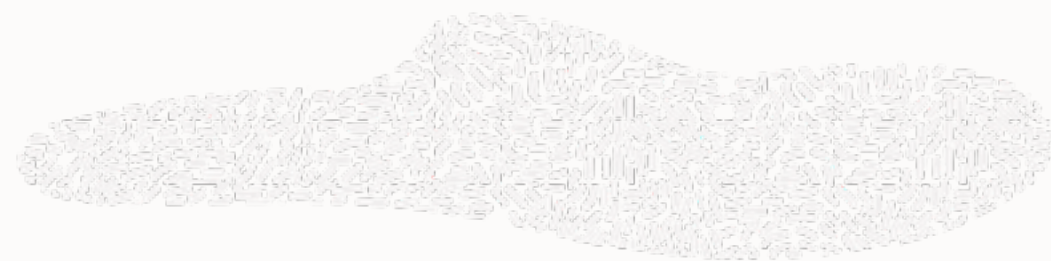
## Startup Time



# Helidon demo





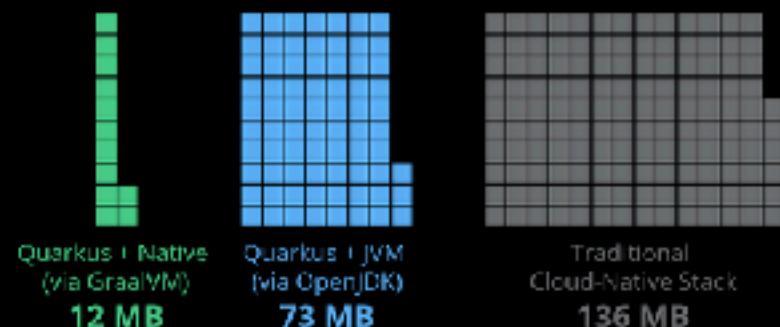


# QUARKUS

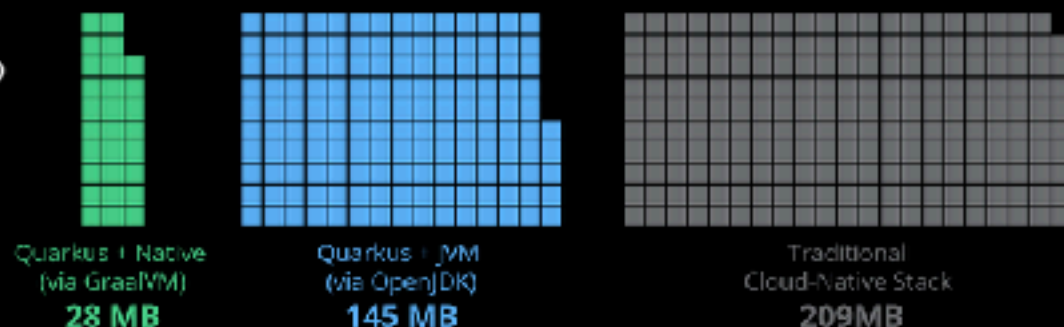
## Memory (RSS) in Megabytes\*

\*Tested on a single-core machine

REST



REST  
+ CRUD



## BOOT + First Response Time

REST



REST  
+ CRUD



# Extensions

A Quarkiverse of extensions enhance your application just as project dependencies do.

Built With

2.16.3.Final

Category

- ☐ Integration
- ☐ Data
- ☐ Web
- ☐ Business Automation
- ☐ Cloud
- ☐ Miscellaneous
- ☐ Reactive
- ☐ Security
- ☐ Serialization
- ☐ Messaging
- ☐ Observability
- ☐ Core
- ☐ Compatibility

**APISTAX**

**APIstax**

Secure and reliable APIs for your common business needs.  
Produce PDFs from HTML.  
Generate EU payment QR codes.  
Verify VAT numbers of european

Category: Miscellaneous  
Version: 1.4.2



**GitHub API**

Connect to the GitHub API

Category: Business Automation  
Version: 1.314.0



**Quarkus - Helm**

Quarkus extension for Helm

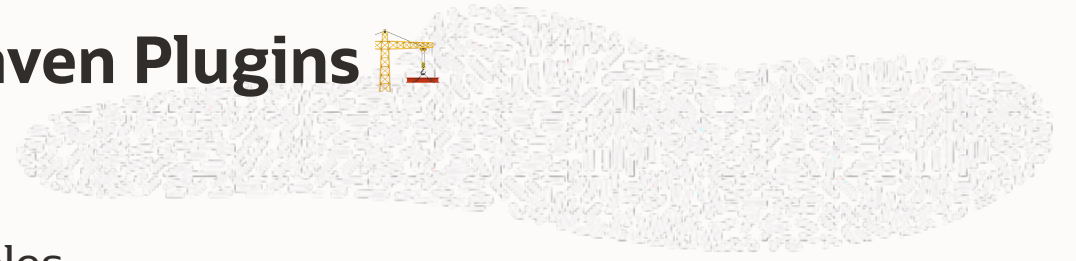
Category: Cloud  
Version: 0.2.7

# Quarkus demo



# Tips & Tricks 🛠️

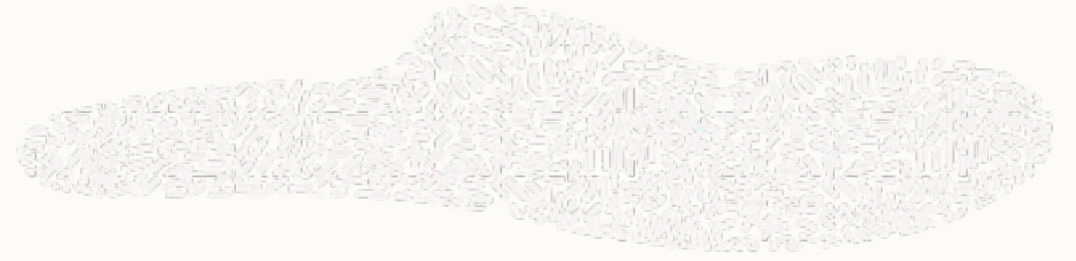
# Native Build tools: Official Gradle and Maven Plugins



- Build, test and run Java applications as native executables
- Out-of-the-box support for native JUnit 5 testing
  - testing Java code with *JUnit 5* behaves in the same way in native execution as with the JVM
  - allows libraries in the JVM ecosystem to run their test suites via GraalVM Native Image

```
plugins {  
  id 'org.graalvm.buildtools.native' version "0.9.20" // or a newer version  
}
```

# GraalVM Native Image & JUnit

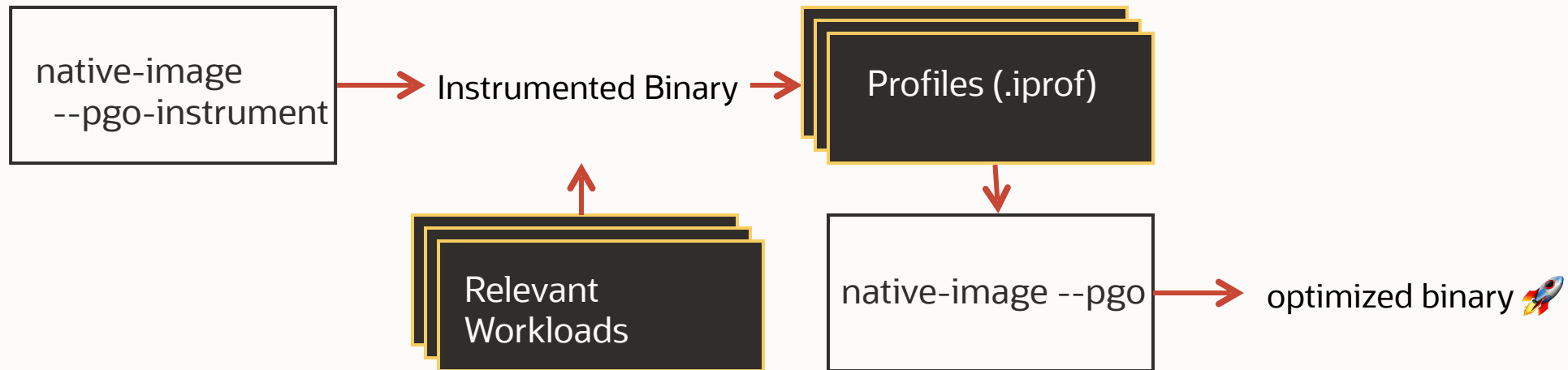


- `@EnabledInNativeImage`
  - used to signal that the annotated test class or test method is only *enabled* when executing within GraalVM native images
  - when applied at the class level, all test methods within that class will be enabled within a native image
- `@DisabledInNativeImage`
  - used to signal that the annotated test class or test method is only *disabled* when executing within a GraalVM native image.

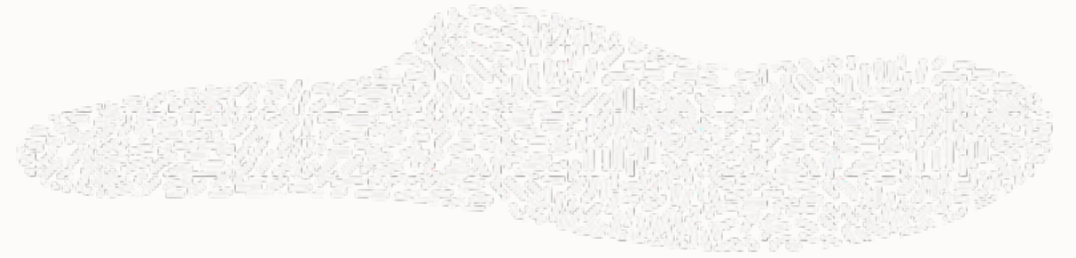


# Optimizing Performance 🚀

# Optimizing performance of native image



# Memory management in Native Image



## Serial GC

- default option
- optimized for low memory footprint and small Java heap sizes

## G1 GC

- optimized to reduce stop-the-world pauses and therefore improve latency
- enable it with `--gc=G1` in GraalVM Enterprise

## Epsilon GC

- no-op garbage collector that does not do any GC = never frees any allocated memory
- enable it with `--gc=epsilon`

# Monitor performance with JFR



- Monitor and optimize performance of native images in production deployments
- include JFR at image build time:


```
native-image --enable-monitoring=jfr JavaApplication
```

- To enable JFR and start a recording:


```
./javaapplication -XX:+FlightRecorder
```


```
-XX:StartFlightRecording="filename=recording.jfr"
```

# Work in progress: Adding JMX support

 grafal #4732

**[GR-40463] Add initial support for remote management over JMX in Native Images #4732**

 Open roberttoyonaga opened 2 months ago

 roberttoyonaga 9 days ago (edited) Edit

### TL;DR



Currently connections via JMX are not supported in native images. This work is to add initial support for that.


### Goals

Achieve similar functionality as in OpenJDK where remote management of managed beans is possible. JMX support will be optionally added at image build time through `--enable-monitoring=jmxclient,jmxserver` options.

### Details

Building a JMX server or client should be possible. Connection should work with authentication as well as SSL. When a JMX server is built and run with remote JMX options, the server should start up right away at runtime similar to in OpenJDK. Configuration will be done in separate client and server features for modularity.

  1


**Assignees**  roberttoyonaga


**Labels** feature native-image


**Milestone** 📌 23.0.0 Release (January 24, 2023)


**Status** In Progress


**Notes** Contributions from Red Hat

 Show all fields

 Open in new tab

 Copy link

 Archive

 Delete from project

# Native Image monitoring demo



# Compressing native images with UPX



Julien Dubois  
@juliendubois

Having fun using UPX [github.com/upx/upx](https://github.com/upx/upx) to compress my @springboot + @graalvm native images

- unzip time goes from 444ms to 77ms 🤖
- native image goes from 66Mb to 17Mb 🤖

RT if you're as excited as me 🤖

```
this Dec 18 12:46:17 CET AOH
+ Downloads mkdir original
+ Downloads mv Functionapp_20201238350.zip original
+ Downloads cd original
+ original time unzip Functionapp_20201238350.zip
Archive: Functionapp_20201238350.zip
  creating: foobar/
  inflating: host.json
  inflating: spring-native-image
  inflating: foobar/function.json
unzip Functionapp_20201238350.zip 0.41s user 0.93s system 98% cpu 0.444 total
+ original ll spring-native-image
-moz-ns-01 julien staff 624 Dec 3 08:34 spring-native-image
+ original ll

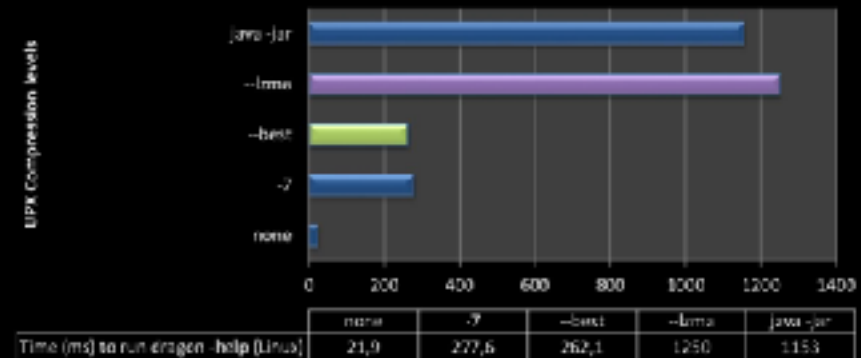
X _[downloading]-[rm]
+ cd Downloads
+ Downloads mkdir upx
+ Downloads mv Functionapp_2020121011843.zip upx
+ Downloads cd upx
+ upx time unzip Functionapp_2020121011843.zip
Archive: Functionapp_2020121011843.zip
  extracting: host.json
  extracting: spring-native-image
  creating: foobar/
  extracting: foobar/function.json
unzip Functionapp_2020121011843.zip 0.05s user 0.01s system 58% cpu 0.077 total
+ upx ll spring-native-image
-moz-ns-01 julien staff 174 Dec 18 12:06 spring-native-image
+ upx ll
```



Gunnar Hillert  
@ghillert

It is quite fascinating to compress a native @graalvm @micronautfw application using #UPX ([upx.github.io](https://upx.github.io)) from 77MB down to 23MB and boot it up (including @FlywayDb migrations) in 65ms! 🚀🔥. #java

according to UPX v3.96 compression levels  
(lower is better)



## Compressed GraalVM Native Images

Get 4–5x smaller executables by compressing GraalVM native images with UPX

[medium.com](https://medium.com)

\* more aggressive compression algorithms can have runtime impact





# Static and Mostly Static Images

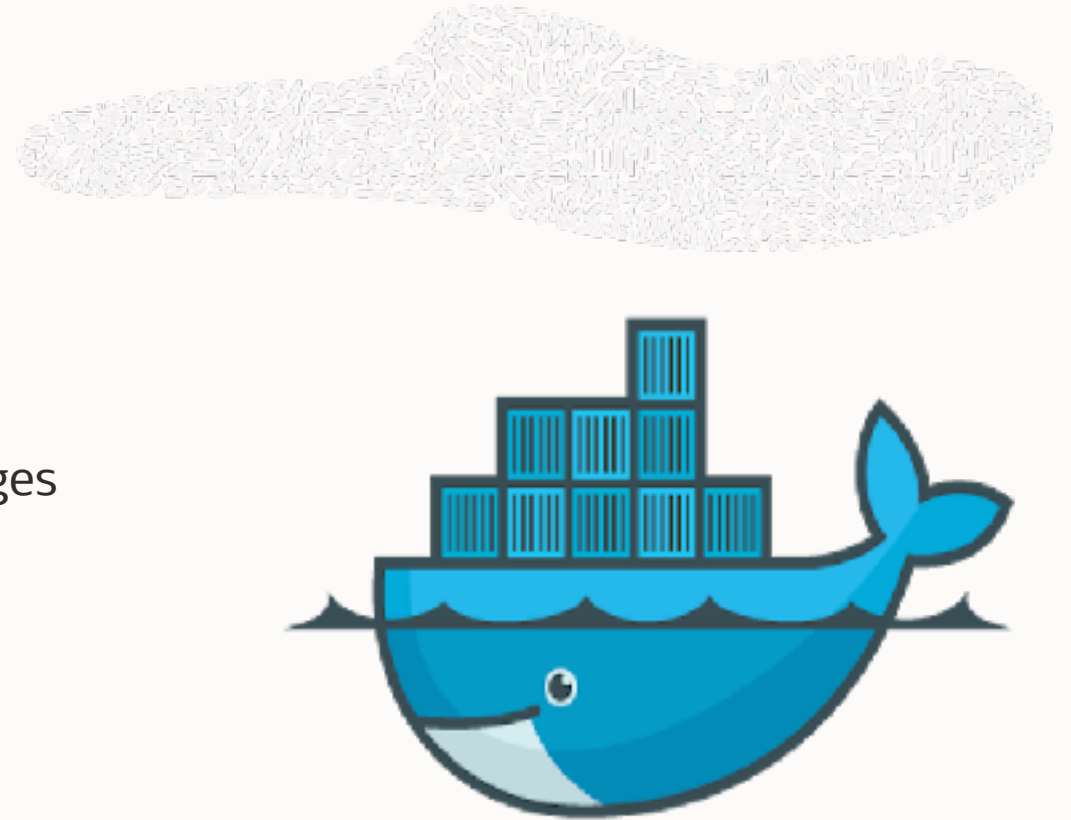
## Static native images

- statically linked against [musl-libc](#), which can be used without any additional library dependencies
- great for deploying on slim or distroless container images

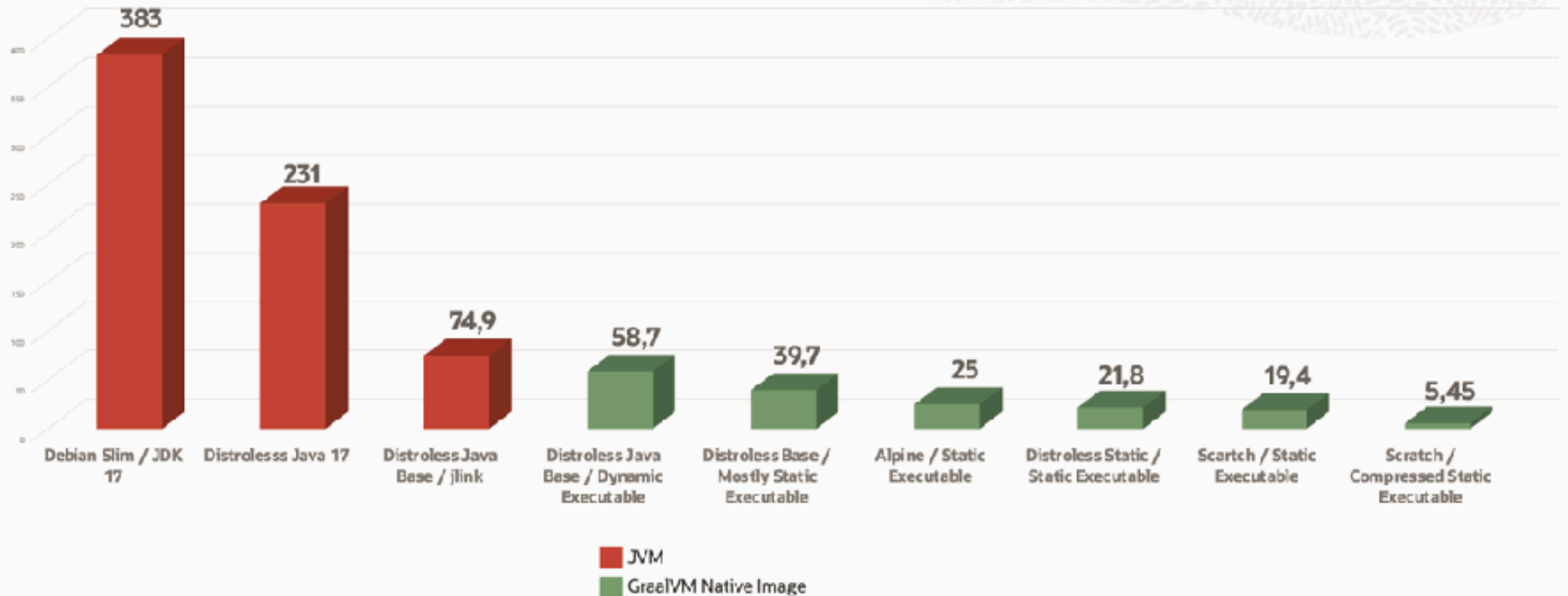
```
FROM gcr.io/distroless/base  
COPY build/native-image/application app  
ENTRYPOINT ["/app"]
```

## Mostly static native images

- statically link against all libraries except libc
- great for deploying such native images on distroless container images



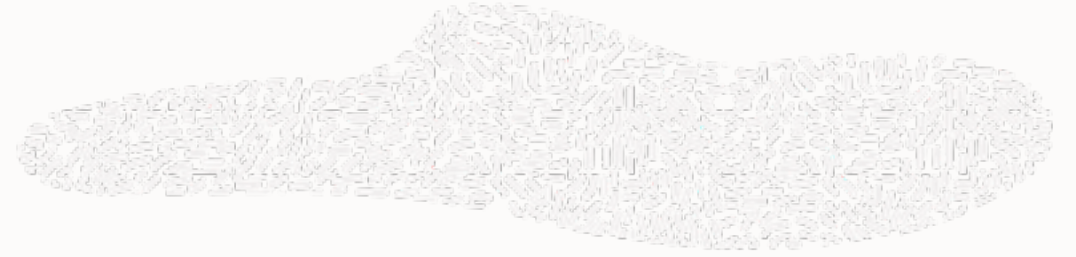
# Lightweight containerized applications



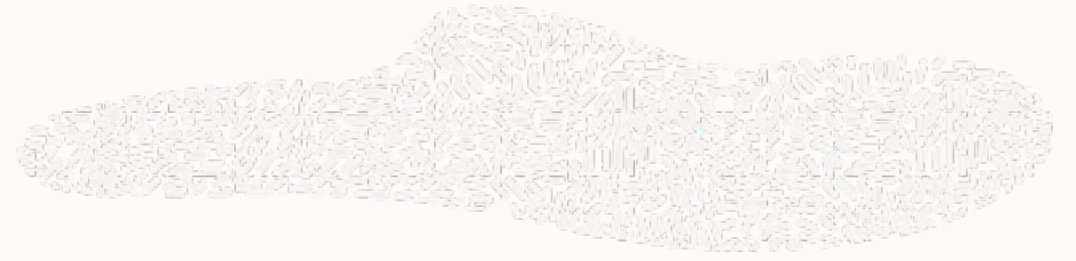
YouTube: A 1.5MB Java Container App? Yes you can! by Shaun Smith



## Security aspects of Native Image



- No new unknown code can be loaded at run time
- Only paths proven reachable by the application are included in the image
- Reflection is disabled by default and needs an explicit include list
- Deserialization only enabled for specified list of classes
- Just-in-time compiler crashes, wrong compilations, or “JIT spraying” to create machine code gadgets are impossible



# **What's the catch?**

# Required Build Time Step



- Computational effort necessary at build time
- Need a powerful machine with the same target architecture & OS
  - Use it with GitHub Actions: [github.com/marketplace/actions/github-action-for-graalvm](https://github.com/marketplace/actions/github-action-for-graalvm)
- Develop in JIT mode for fast development, only use AOT for final deployment
- For best throughput, use profile-guided optimizations:

```
native-image --pgo-instrument MyMainClass
./mymainclass

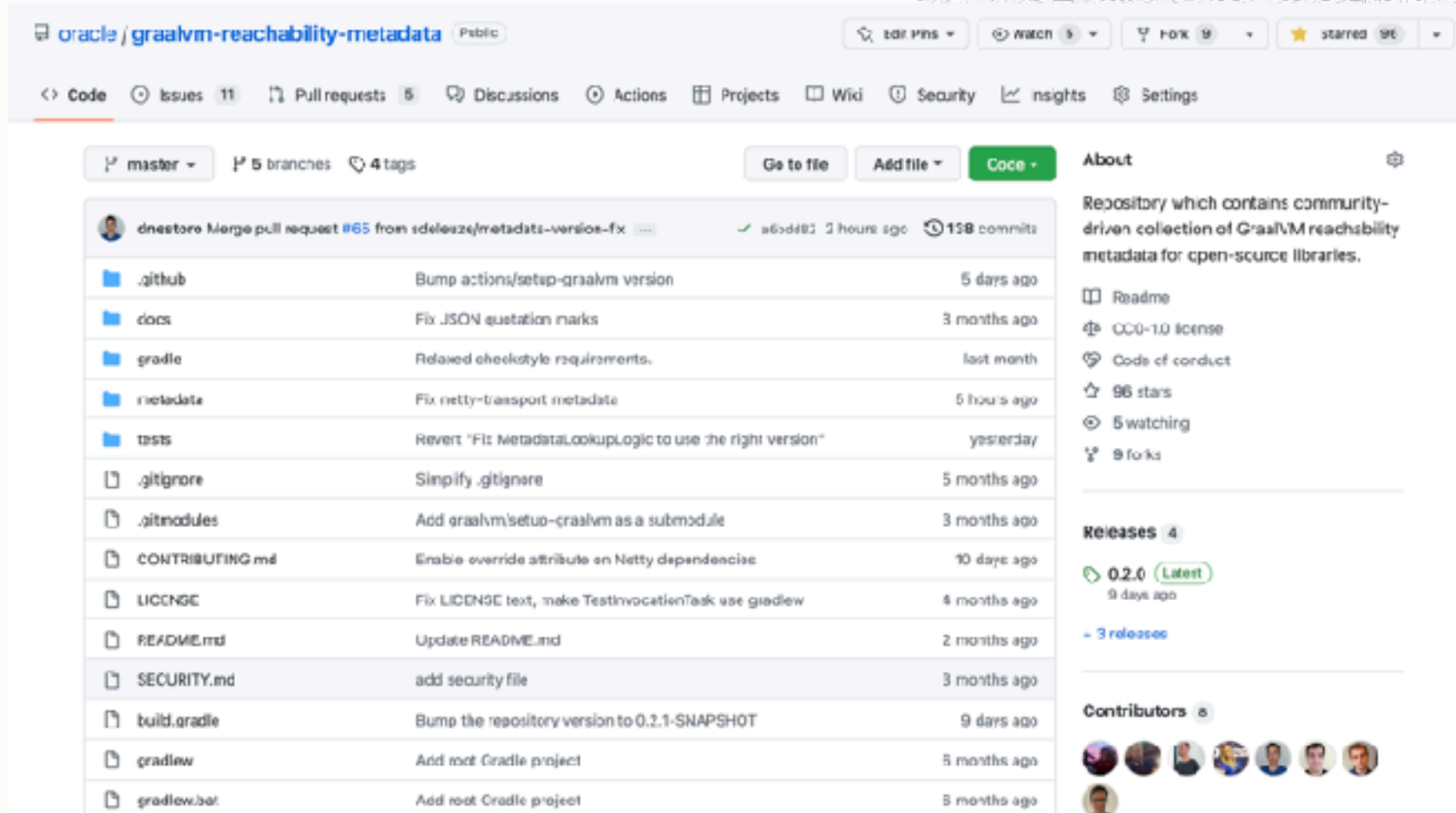
native-image --pgo=profile.iprof MyMainClass
./mymainclass
```

# GraalVM & Reflection?



- GraalVM 🤝 Reflection!
- Native Image tries to resolve the target elements through a static analysis that detects calls to the Reflection API
  - If the analysis can not automatically detect your use of reflection, you might need additional configuration
- Trace reflection, JNI, resource usage on the JVM with the tracing agent
  - Manual adjustment / addition might still be necessary

# What about reflection in 3<sup>rd</sup>-party libraries?



The screenshot displays the GitHub interface for the repository 'oracle/graalvm-reachability-metadata'. The repository is public and has 96 stars, 5 watchers, and 9 forks. It contains 5 branches and 4 tags. The main content area shows a list of files and their commit history:

File	Commit Message	Commit Date
.github	Bump actions/setup-graalvm version	5 days ago
docs	Fix JSON quotation marks	3 months ago
gradle	Relaxed checkstyle requirements.	last month
metadata	Fix netty-transport metadata	5 hours ago
tests	Revert "Fix MetadataLookupLogic to use the right version"	yesterday
.gitignore	Simplify .gitignore	5 months ago
.gitmodules	Add graalvm/setup-graalvm as a submodule	3 months ago
CONTRIBUTING.md	Enable override attribute on Netty dependencies	10 days ago
LICENSE	Fix LICENSE text, make TestInvocationTask use gradlew	4 months ago
README.md	Update README.md	2 months ago
SECURITY.md	add security file	3 months ago
build.gradle	Bump the repository version to 0.2.1-SNAPSHOT	9 days ago
gradlew	Add root Gradle project	5 months ago
gradlew.bat	Add root Gradle project	5 months ago

The right sidebar shows the repository's description: 'Repository which contains community-driven collection of GraalVM reachability metadata for open-source libraries.' It also lists the repository's metadata: CC0-1.0 license, Code of conduct, 96 stars, 5 watching, and 9 forks. The 'Releases' section shows the latest release, 0.2.0, which was released 9 days ago. The 'Contributors' section shows 6 contributors.

# Is there an easier way to handle reflection? Yes!

```
<plugin>
  <groupId>org.graalvm.buildtools</groupId>
  <artifactId>native-maven-plugin</artifactId>
  <version>${native.maven.plugin.version}</version>
  <extensions>true</extensions>
  <executions>
    <execution>
      <id>build-native</id>
      <goals>
        <goal>compile-no-fork</goal>
      </goals>
      <phase>package</phase>
    </execution>
  </executions>
  <configuration>
    <!-- tag::metadata-default[] -->
    <metadataRepository>
      <enabled>true</enabled>
    </metadataRepository>
    <!-- end::metadata-default[] -->
  </configuration>
</plugin>
```



# What's new in GraalVM

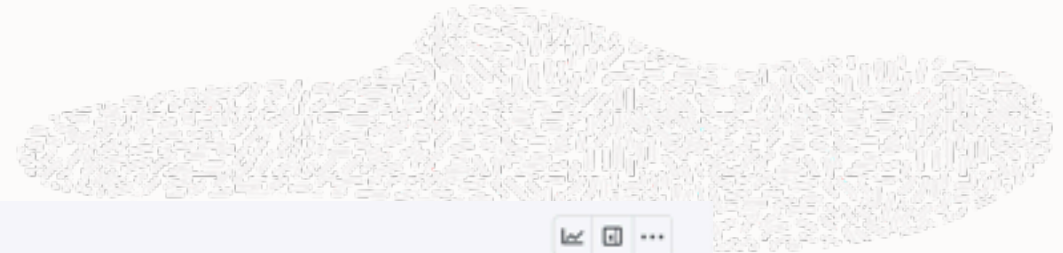
# GraalVM<sup>TM</sup>

# JDK 20

dev builds

now available! 🎉

# GraalVM Community roadmap on GitHub



GraalVM Community Roadmap

<https://github.com/orgs/oracle/projects/6>

# What's next for GraalVM

## Add support for ZGC on HotSpot #5050

Open

tkrodriguez opened this issue on Sep 22, 2022 · 0 comments



tkrodriguez commented on Sep 22, 2022 · edited by fniephaus

Member



### TL;DR

Add support for [Z Garbage Collector](#) to the Graal compiler.

### Goals

Add required ZGC barriers on HotSpot along with any relevant performance optimizations, allowing the use of ZGC when the Graal is used as a JIT compiler.

### Non-Goals

- Add support for ZGC to GraalVM Native Image
- Add support for Shenandoah GC (although ZGC support will make it easier to support other GCs in the future)

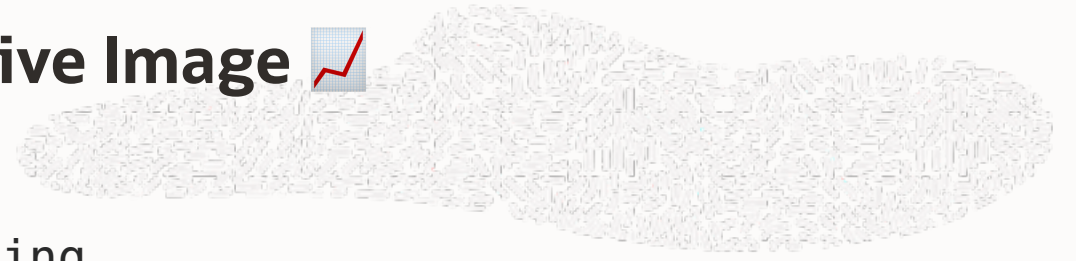


2



11

# New monitoring features in GraalVM Native Image



- `-H:+AllowVMInspection -> --enable-monitoring`
  - `--enable-monitoring=<all,heapdump,jfr,jvmstat>`
- added support for `jvmstat` in Native Image
- keep building out the JFR support in Native Image



## What's next for Native Image

- Simplifying configuration and compatibility for Java libraries
- Continuing with peak performance improvements
- Keep working with Java framework teams to leverage all Native Image features, develop new ones, improve performance, and ensure a great developer experience
- Further reduce build time and footprint of the Native Image builder
- IDE support for Native Image configuration and agent-based configuration
- Further improving GC performance and adding new GC implementations

# Libraries and Frameworks Ready for Native Image

Name	Version	Test Level
ch.qos.logback:logback-classic <sup>1</sup>	1.2.11 - latest	★
com.datastax.oss:java-driver-core	4.1.5 - latest	★
com.ecwid.consul:consul-api <sup>1</sup>	1.4.5 - latest	★
com.github.luben:zstd-jni <sup>1</sup>	1.5.2-5 - latest	★
com.graphql-java:graphql-java <sup>1</sup>	19.2 - latest	★
com.h2database:h2 <sup>1</sup>	2.1.210 - latest	★
com.hazelcast:hazelcast <sup>1</sup>	5.2.1 - latest	★
com.mysql:mysql-connector-j <sup>1</sup>	8.0.31 - latest	★
com.oracle.database.jdbc:ojdbc11	21.1.0.0 - latest	★★
com.oracle.database.jdbc:ojdbc8	21.1.0.0 - latest	★★
com.sun.mail:jakarta.mail <sup>1</sup>	2.0.1 - latest	★
com.zaxxer:HikariCP <sup>1</sup>	5.0.1 - latest	★
io.grpc:grpc-netty <sup>1</sup>	1.51.0 - latest	★
io.helidon.config:helidon-config	1.0.0 - latest	★★
io.helidon.microprofile.bundle:helidon-microprofile	2.0.0 - latest	★★
io.helidon.microprofile.bundle:helidon-microprofile-core	2.0.0 - latest	★★
io.helidon.webclient:helidon-webclient	1.0.0 - latest	★★
io.helidon.webserver:helidon-webserver	1.0.0 - latest	★★
io.jsonwebtoken:jjwt-gson <sup>1</sup>	0.11.5 - latest	★
io.jsonwebtoken:jjwt-jackson <sup>1</sup>	0.11.5 - latest	★
io.jsonwebtoken:jjwt-argson <sup>1</sup>	0.11.5 - latest	★
io.micronaut.aop:micronaut-aop	3.9.0 - latest	★★

# Get started with GraalVM



## Get started with GraalVM

```
bash <(curl -sL https://get.graalvm.org/jdk)\  
    graalvm-ce-java19-22.3.0
```

```
    sdk install java 22.3.r19-grl
```

# Danke schön!

Presentation & resources:



Got questions?

