Awesome Java applications with GraalVM and Java microservices frameworks

Alina Yurenko

Developer Advocate for GraalVM

Oracle Labs

Voxxed Days Zurich

by Patrick Federi @ Unsplash

GraalVM™

* Wenger Giant Swiss Army Knife, 141 Functions

GraalVM™

JIT

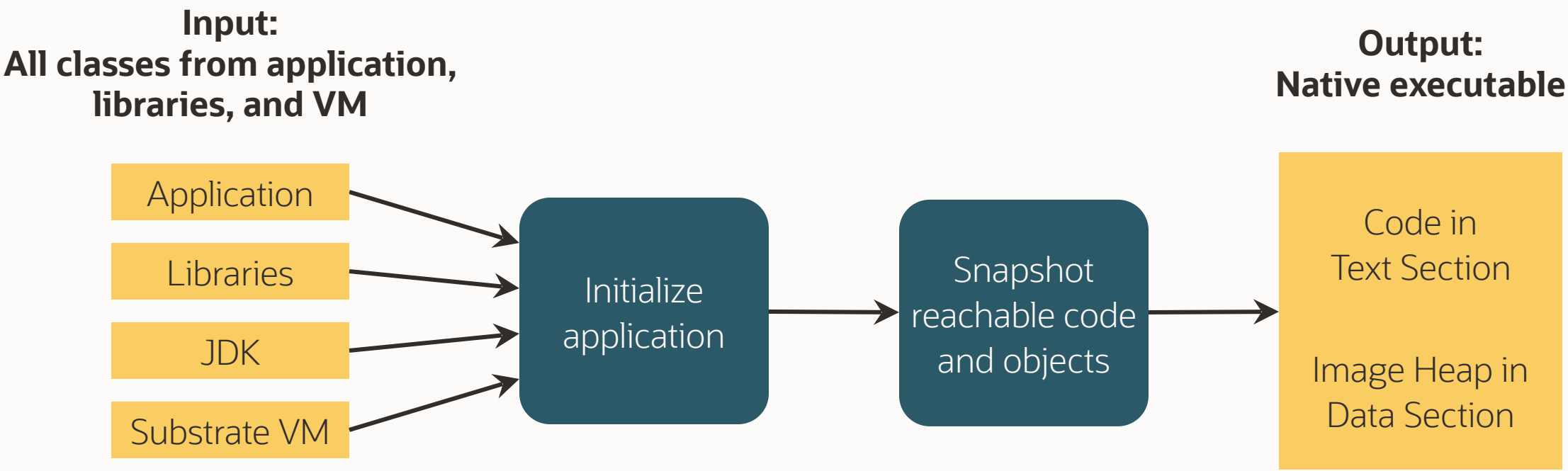
```
java MyMainClass
```

AOT

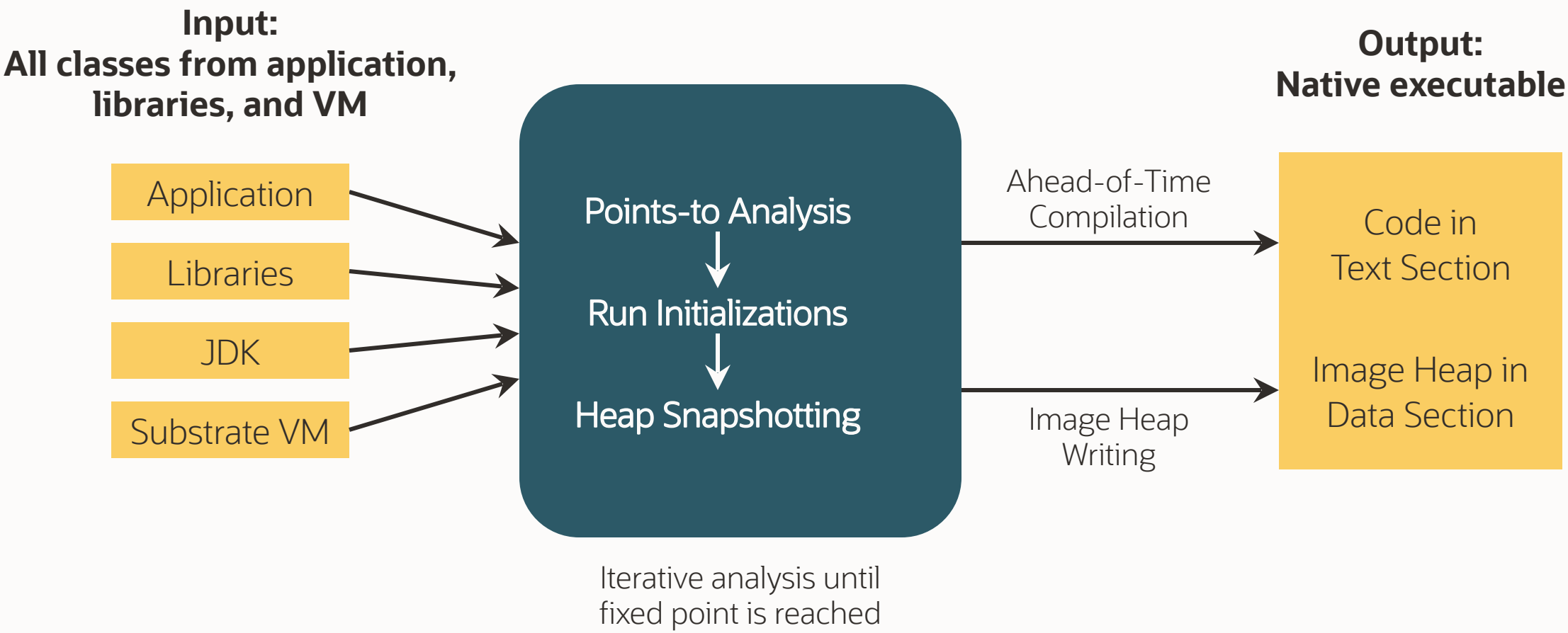
```
native-image MyMainClass
./mymainclass
```

GraalVM Native Image

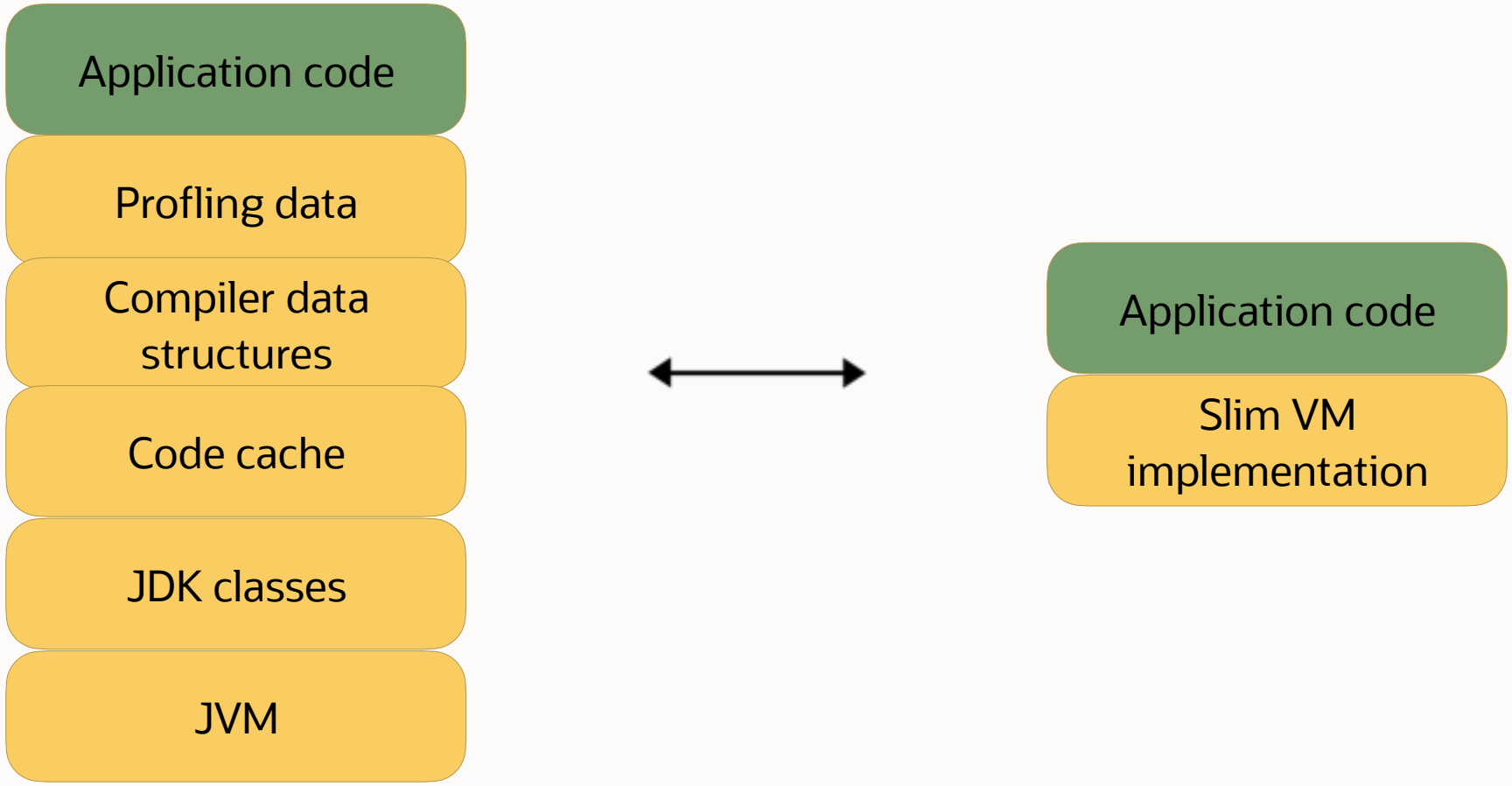
Native Image Build Process: the idea



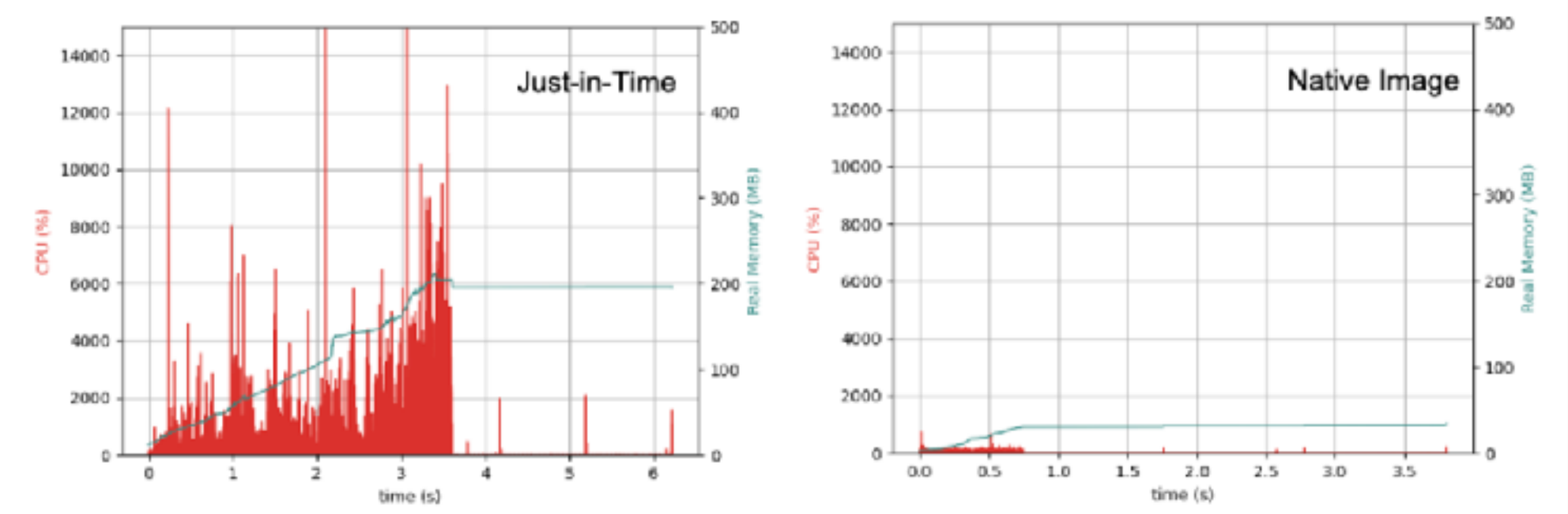
Native Image Build Process



Executing on the JVM vs Native Image



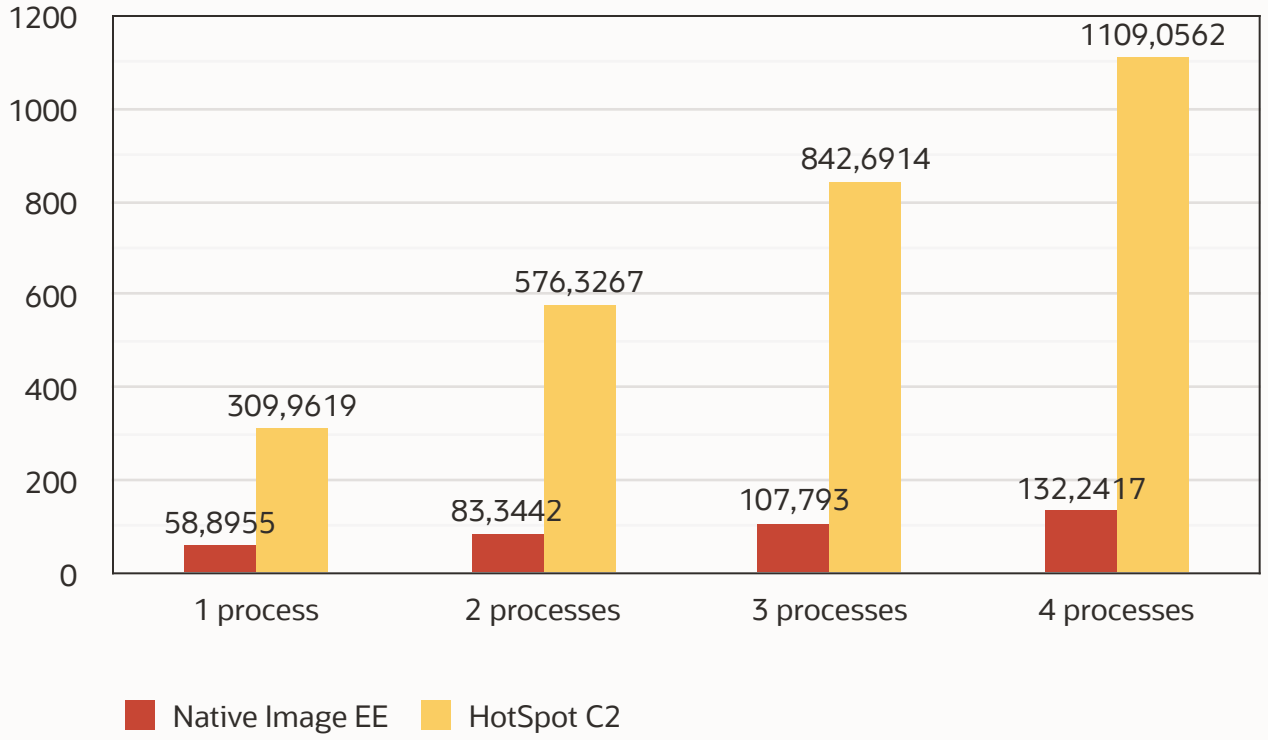
CPU and memory usage of GraalVM Native Image applications



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


spring

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.

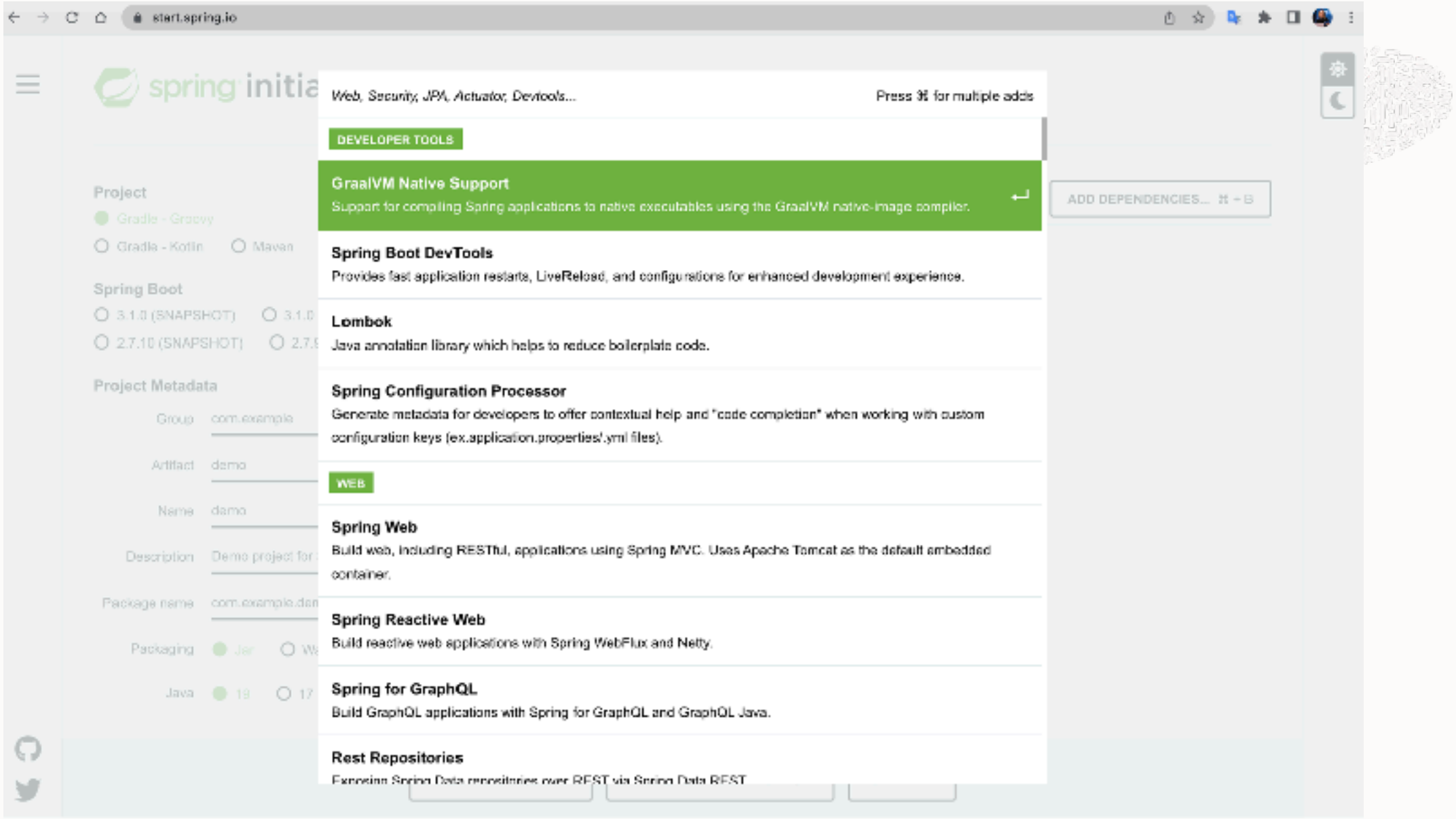
This is the first major revision of Spring Boot since 2.0 was released 4.5 years ago. It's also the first GA version of Spring Boot that provides support for Spring Framework 6.0 and GraalVM.

Highlights of the new release include:

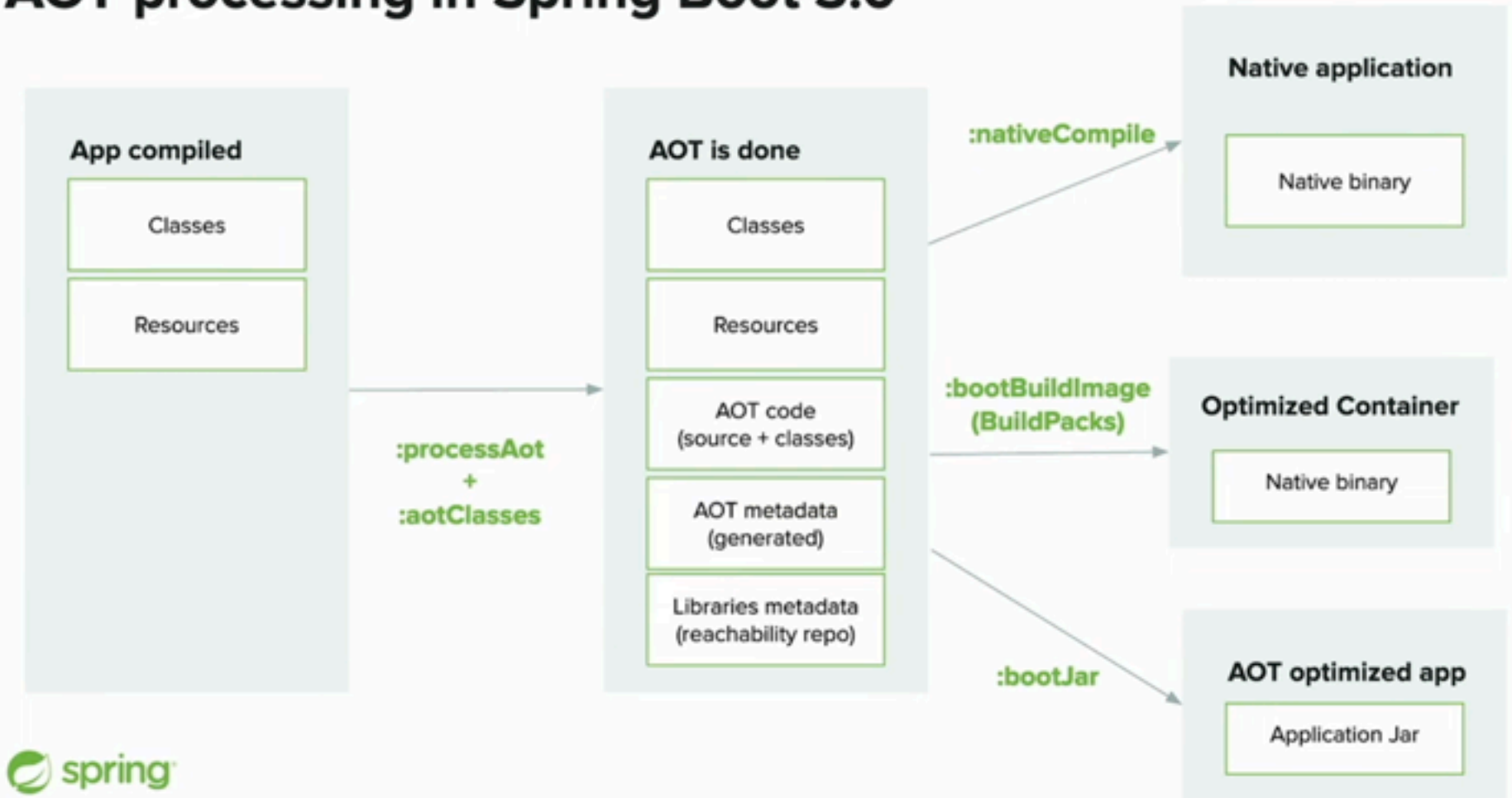
- A Java 17 baseline
- Support for generating native images with GraalVM, superseding the experimental Spring Native project
- Improved observability with Micrometer and Micrometer Tracing
- Support for Jakarta EE 10 with an EE 9 baseline

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
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
Project
Gradle - Groovy
Gradle - Kotlin
Maven
Spring Boot
Project Metadata
Group
Artifact
Name
Description
Package name
Packaging
Java
ADD DEPENDENCIES... ⌘ + B

AOT processing in Spring Boot 3.0



spring

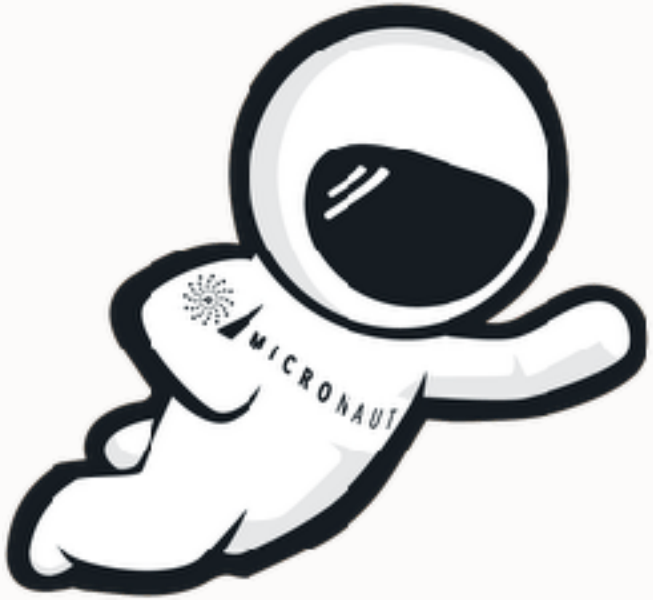
Spring's AOT processing

- 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



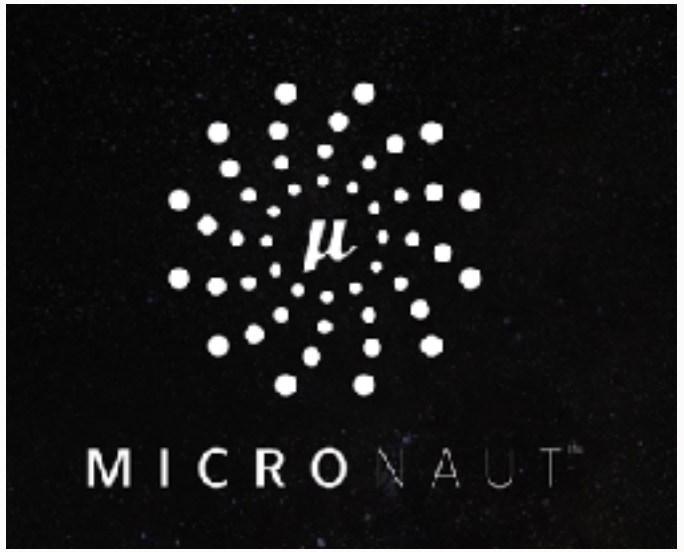
MICRONAUT™

Micronaut

A modern, JVM-based, full-stack framework for building modular, easily testable Microservice and Serverless applications.

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.

Website and documentation: micronaut.io



Cédric Champeau ↠ @melix@mastodon.xyz
@CedricChampeau

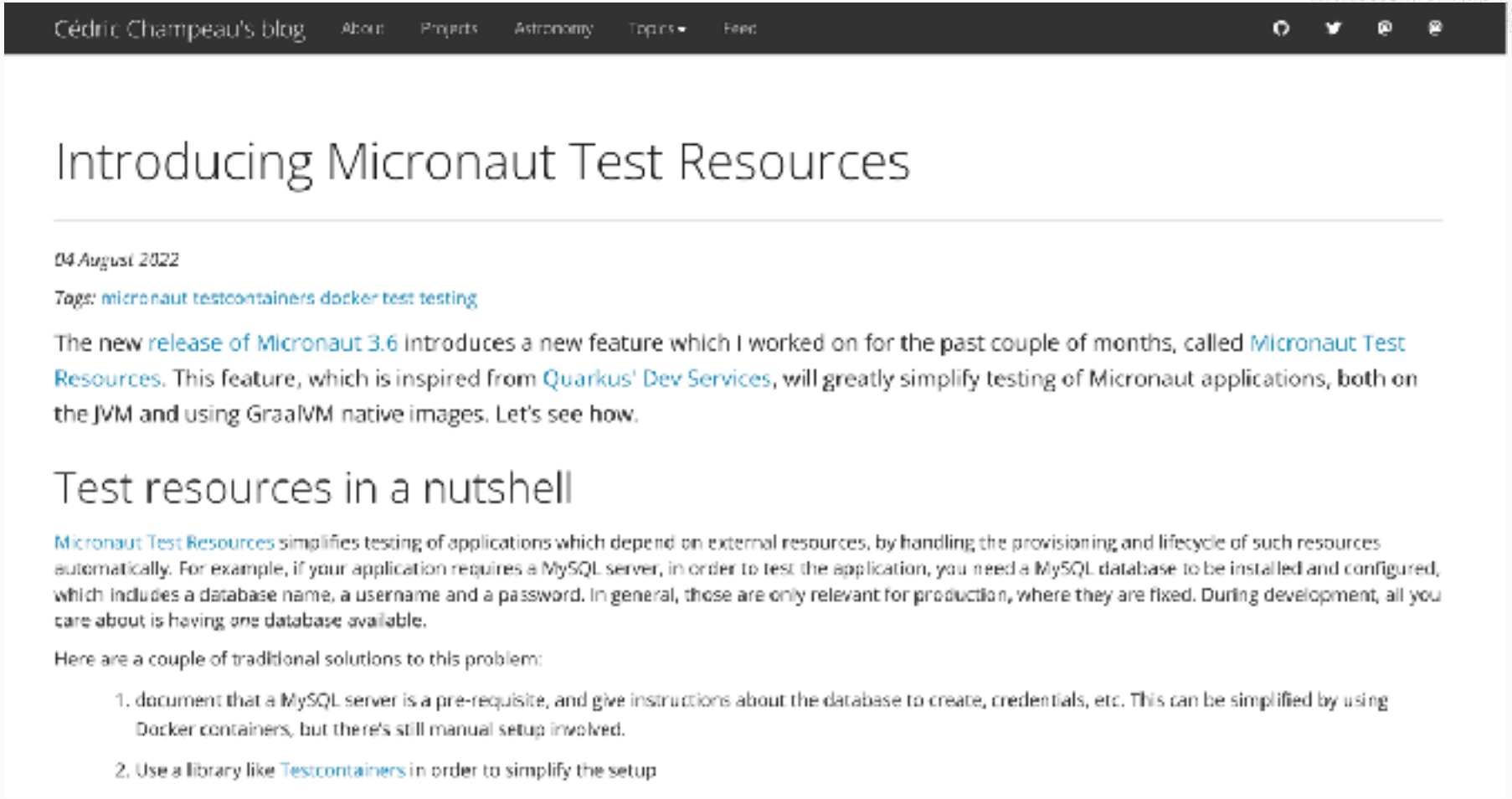
🤯 #Micronaut

```
                    _
  __ _ _   _| |_
 / _` | | | | __|
| (_| | |_| | |_
|\__,_|\__,_|\__|


icronaut.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.

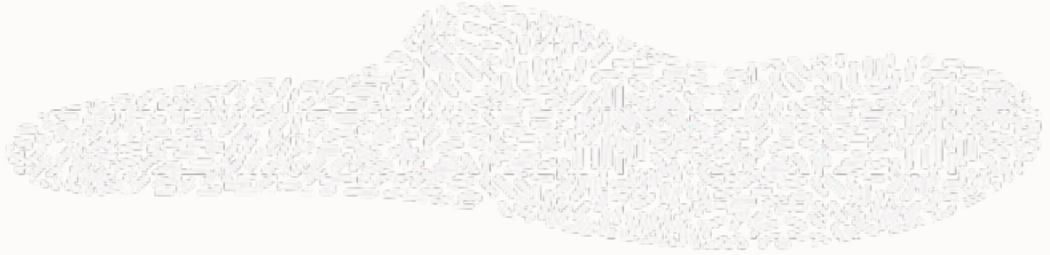
Test resources in a nutshell

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.

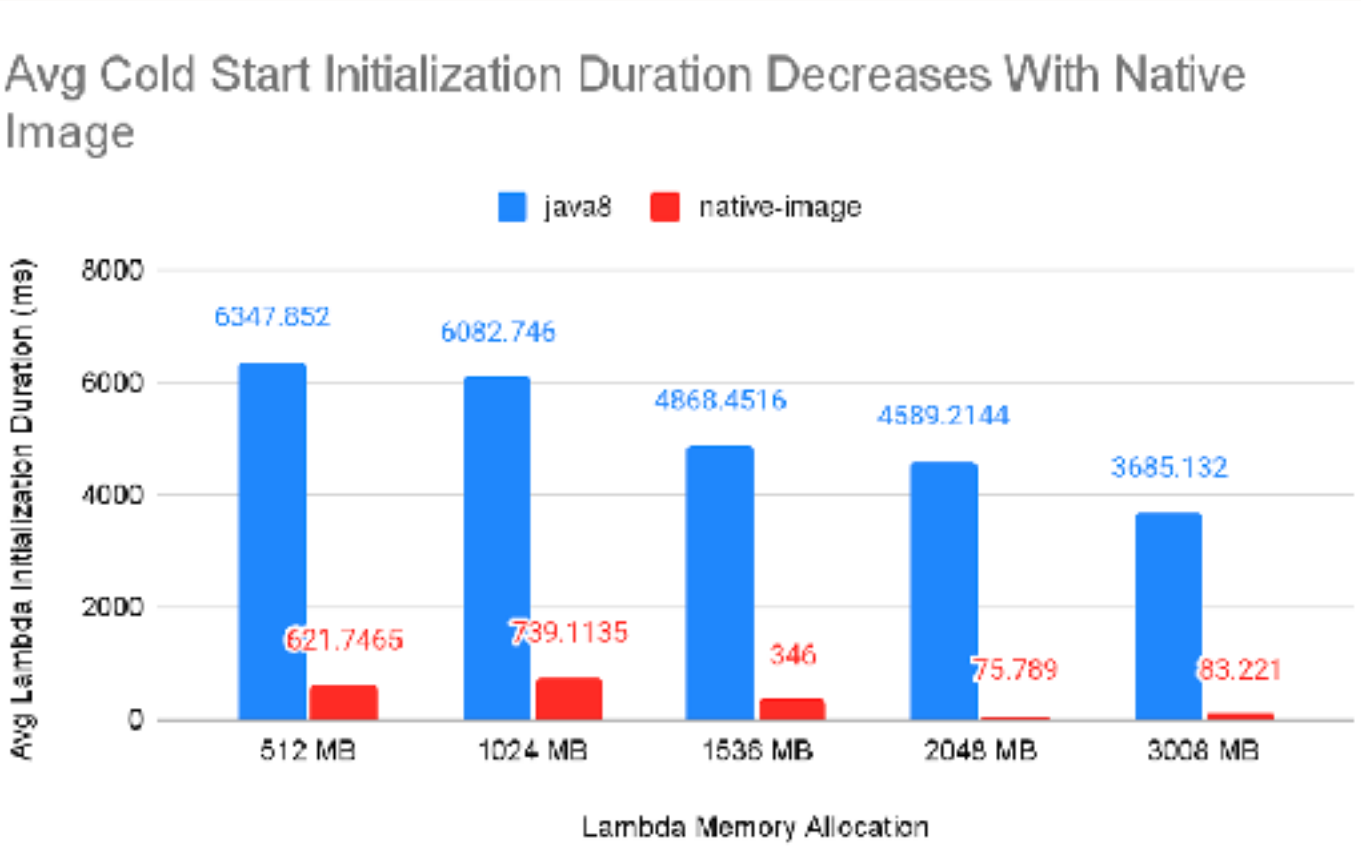
Here are a couple of traditional solutions to this problem:

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



"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";
    }
}
```

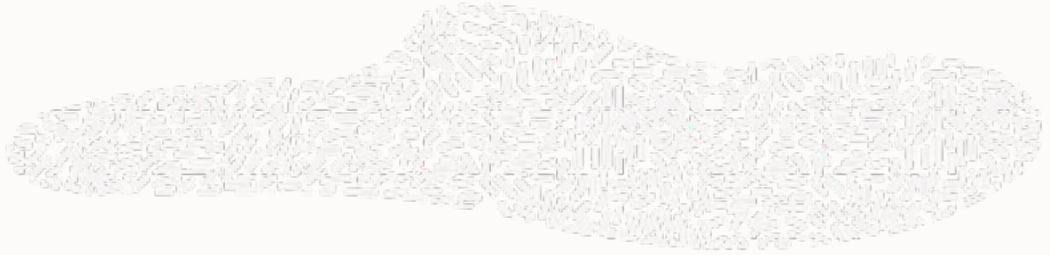
Helidon 🤝 GraalVM

Memory Footprint

Disk Space Usage

Startup Time

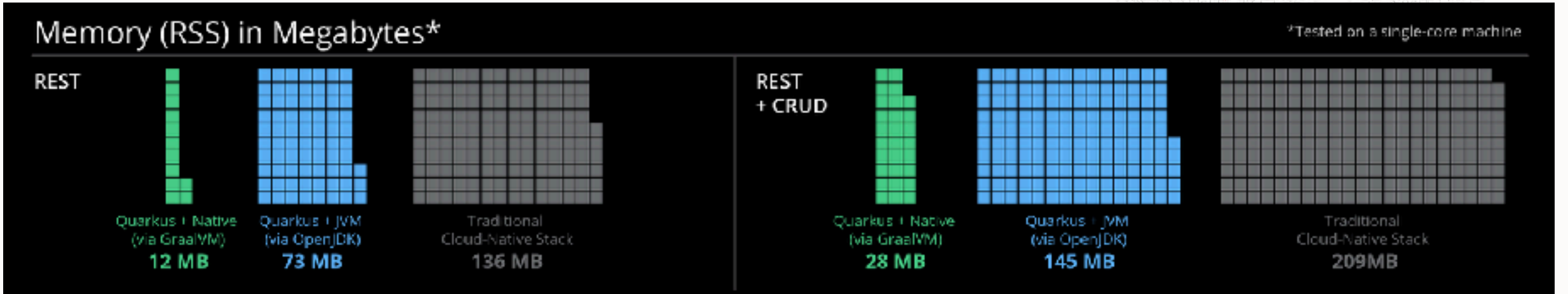
Helidon demo



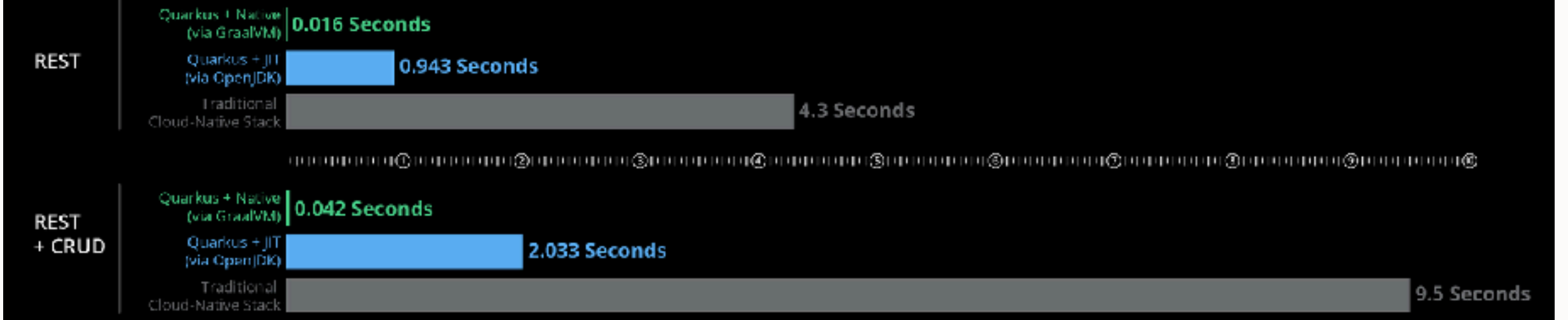
QUARKUS

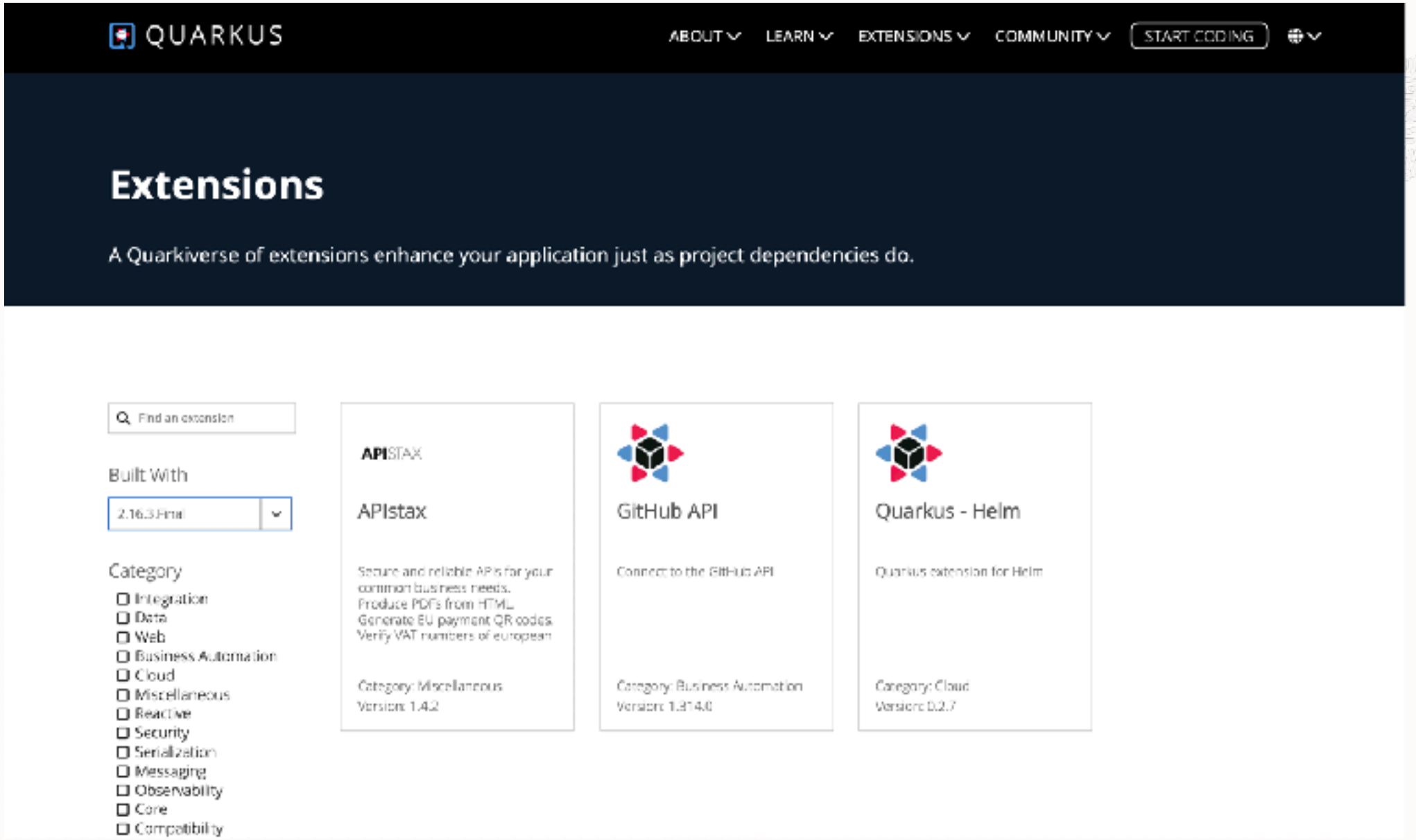
Memory (RSS) in Megabytes*

*Tested on a single-core machine



BOOT + First Response Time




QUARKUS
ABOUT
LEARN
EXTENSIONS
COMMUNITY
START CODING
Extensions
A Quarkiverse of extensions enhance your application just as project dependencies do.
Find an extension
Built With
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
Quarkus - Helm
Quarkus extension for Helm
Category: Cloud

Quarkus demo

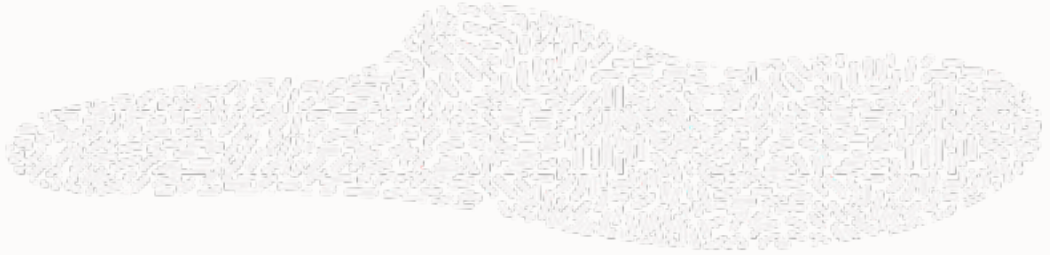
Tips & Tricks 

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

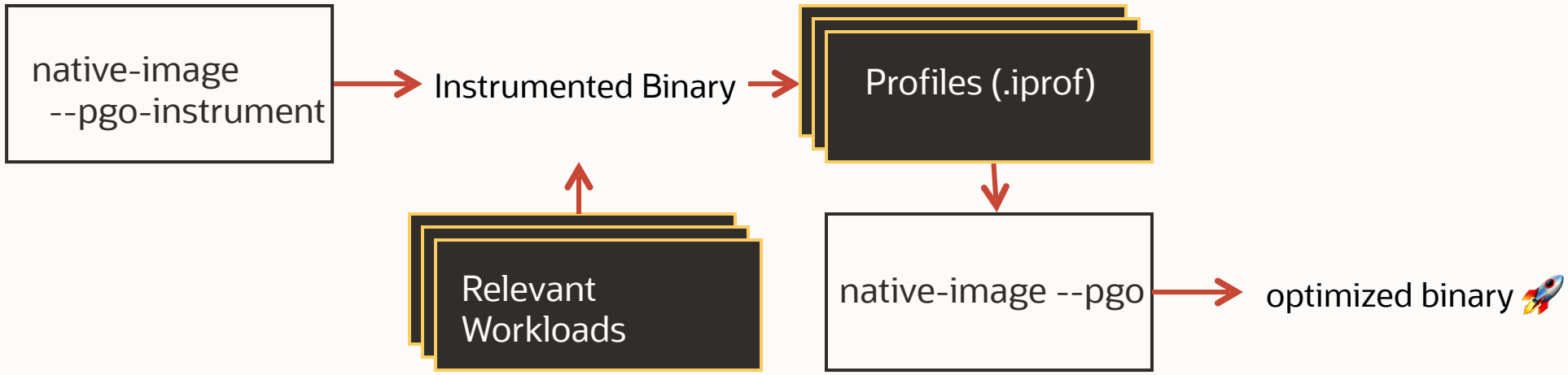


- `@EnabledInNativeImage`
  - used to signal that the annotated test class or test method is only *enabled* when executing within GraalVM native images
  - when applied at the class level, all test methods within that class will be enabled within a native image

- `@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

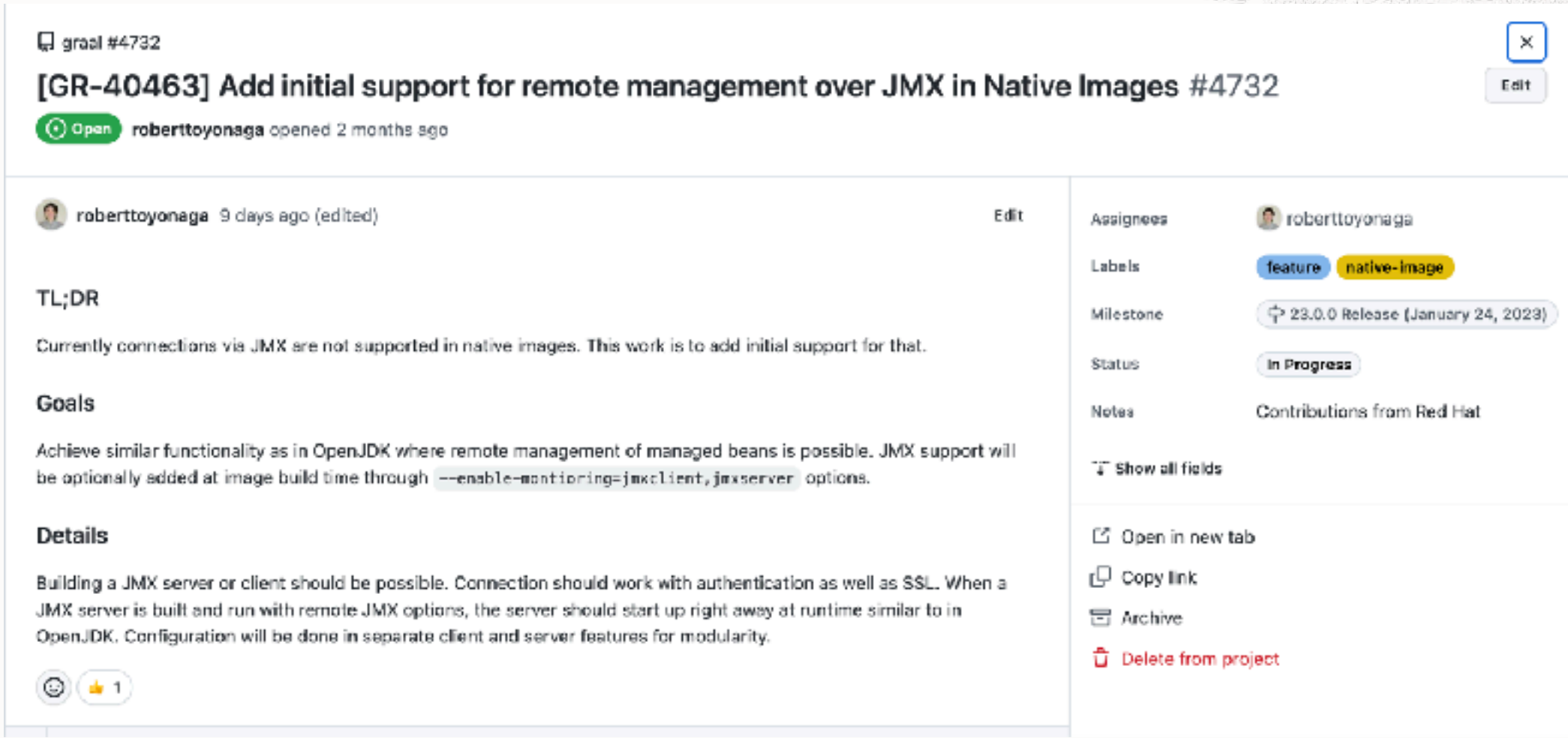
- Monitor and optimize performance of native images in production deployments
- include JFR at image build time:

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

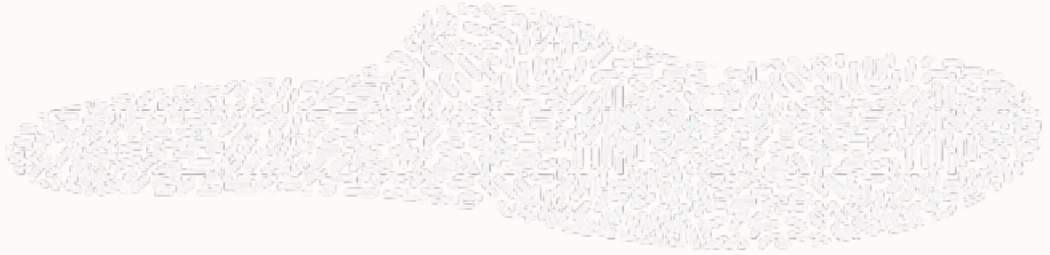
- To enable JFR and start a recording:

```
./javaapplication -XX:+FlightRecorder
-XX:StartFlightRecording="filename=recording.jfr"
```

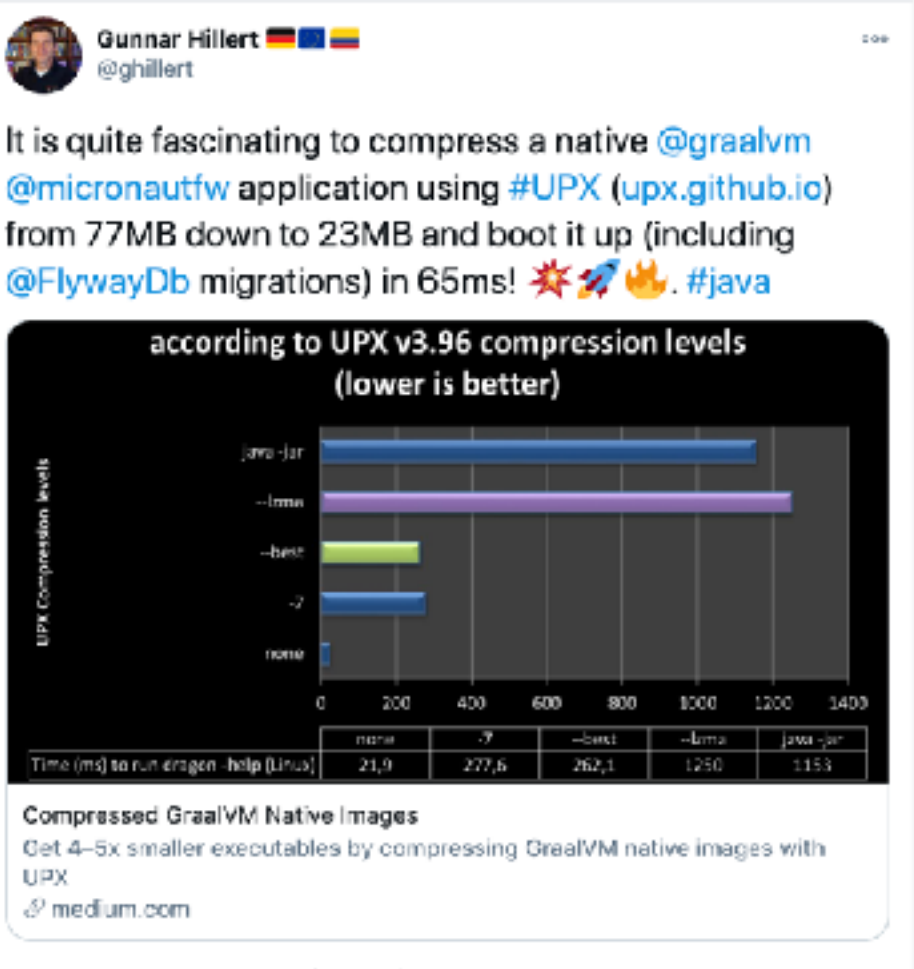
Work in progress: Adding JMX support



Native Image monitoring demo



Compressing native images with UPX



** more aggressive compression algorithms can have runtime impact*

Static and Mostly Static Images

Static native images

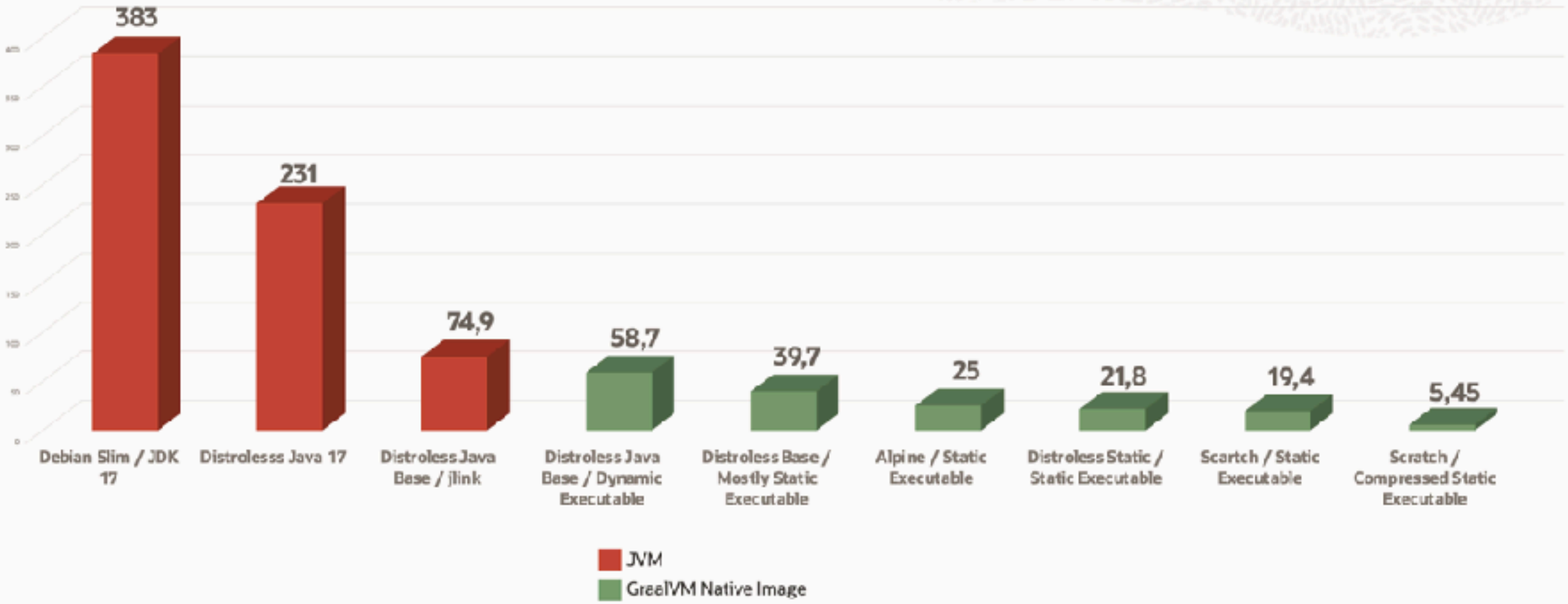
- statically linked against musl-libc , which can be used without any additional library dependencies
- great for deploying on slim or distroless container images

```
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



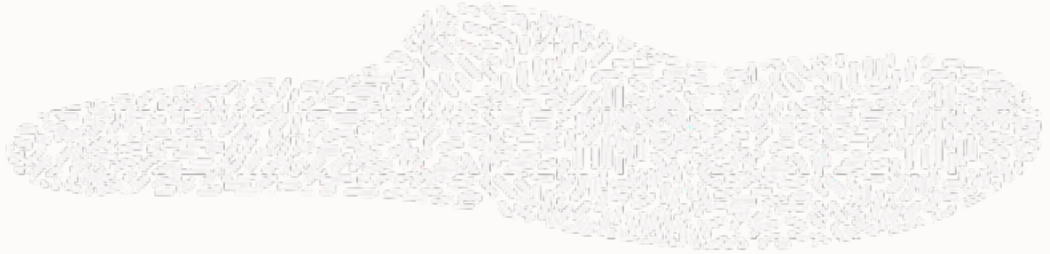
YouTube: A 1.5MB Java Container App? Yes you can! by Shaun Smith

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



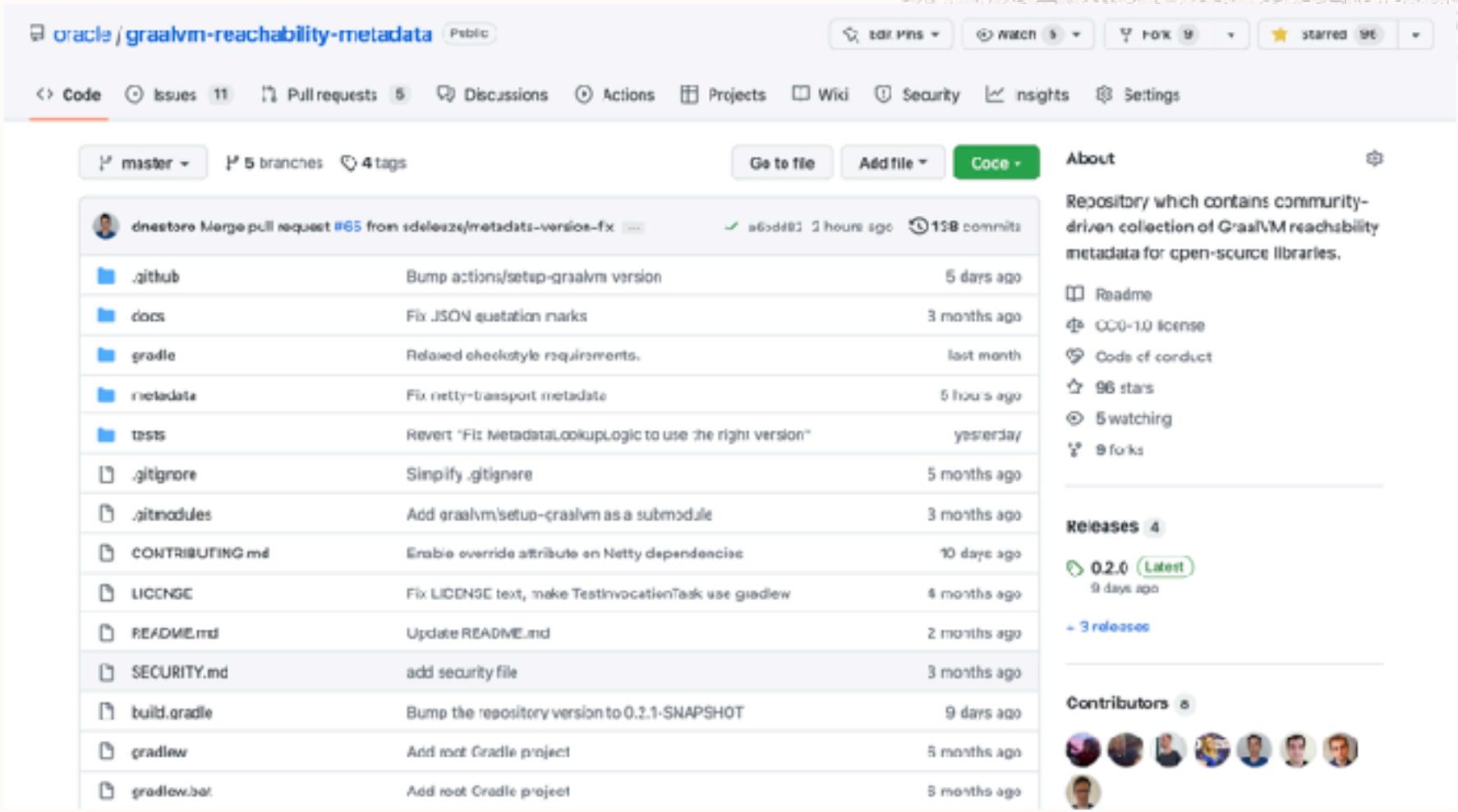
- Computational effort necessary at build time
- Need a powerful machine with the same target architecture & OS
  - Use it with GitHub Actions: github.com/marketplace/actions/github-action-for-graalvm
- Develop in JIT mode for fast development, only use AOT for final deployment
- For best throughput, 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 3rd-party libraries?



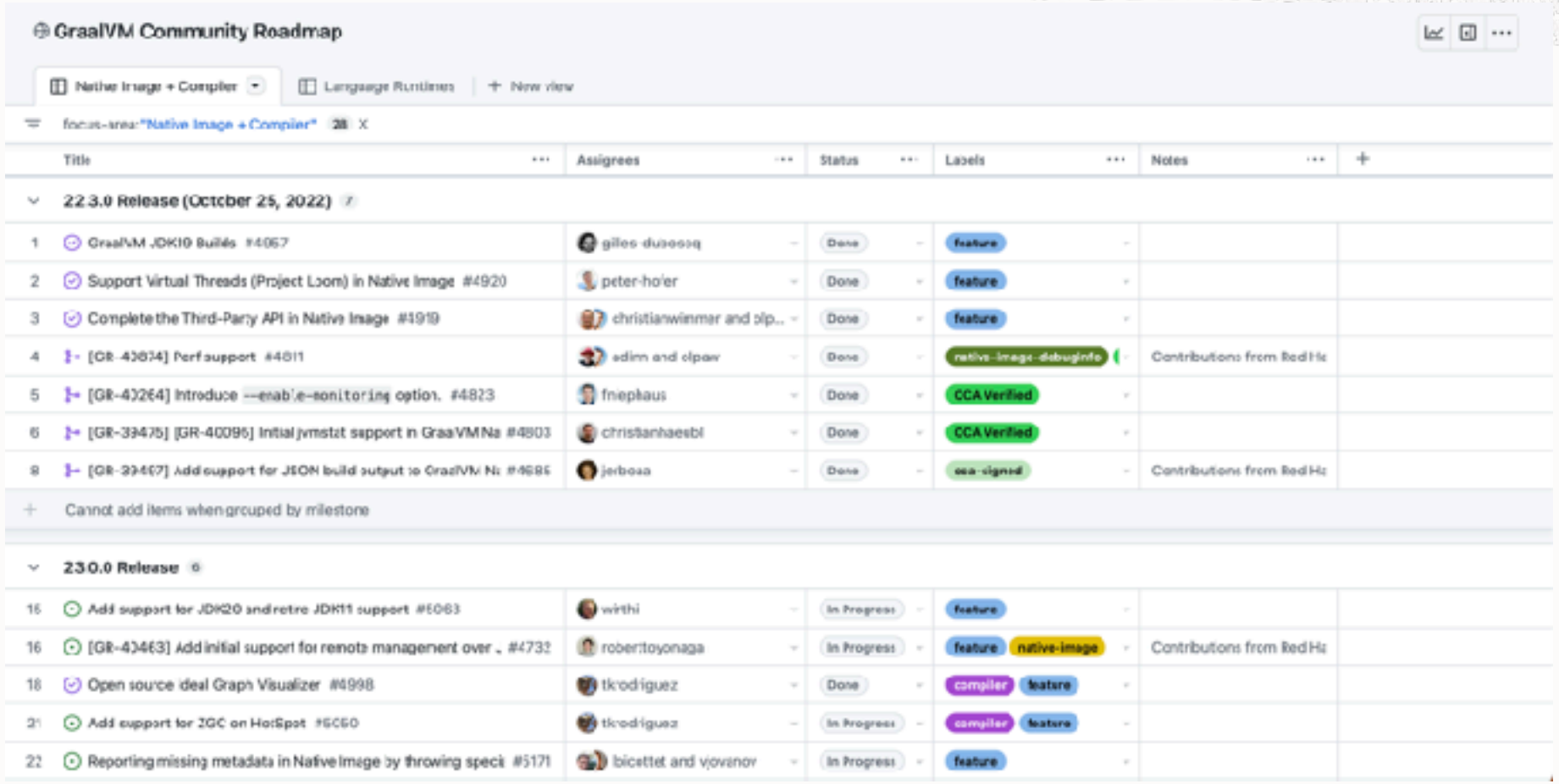
Is there an easier way to handle reflection? Yes!

```
<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™
JDK 20
dev builds
now available!

GraalVM Community roadmap on GitHub



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

What's next for GraalVM

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	1.2.11 - latest	★
com.datastax.oss:java-driver-core	4.13 - latest	★
com.ecwid.consul:consul-api	1.4.5 - latest	★
com.github.luben:zstd-jni	1.5.2-5 - latest	★
com.graphql-java:graphql-java	19.2 - latest	★
com.h2database:h2	2.1.210 - latest	★
com.hazelcast:hazelcast	5.2.1 - latest	★
com.mysql:mysql-connector-j	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	2.0.1 - latest	★
com.zaxxer:HikariCP	5.0.1 - latest	★
io.grpc:grpc-netty	1.51.0 - latest	★
io.helidon.config:helidon-config	1.0.0 - latest	★★
io.helidon.microprofile.bundles:helidon-microprofile	2.0.0 - latest	★★
io.helidon.microprofile.bundles: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	0.11.5 - latest	★
io.jsonwebtoken:jjwt-jackson	0.11.5 - latest	★
io.jsonwebtoken:jjwt-orgjson	0.11.5 - latest	★
io.micronaut.acme:micronaut-acme	3.2.0 - latest	★★

graalvm.org/native-image/libraries-and-frameworks

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?