



# Supercharge Your Native Image Applications

**Alina Yurenko**

Developer Advocate for GraalVM

Oracle Labs

Accenture

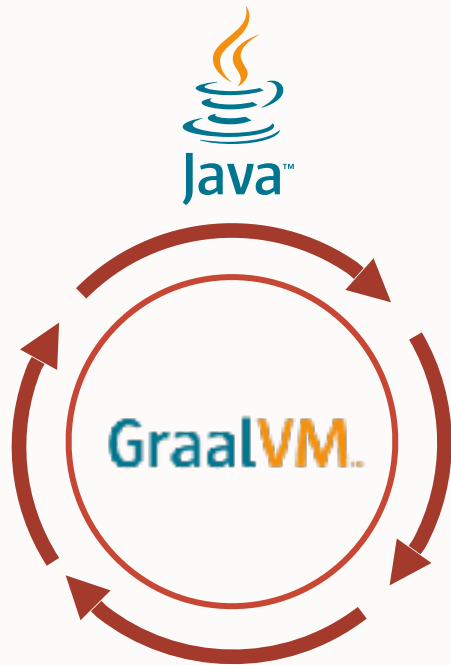
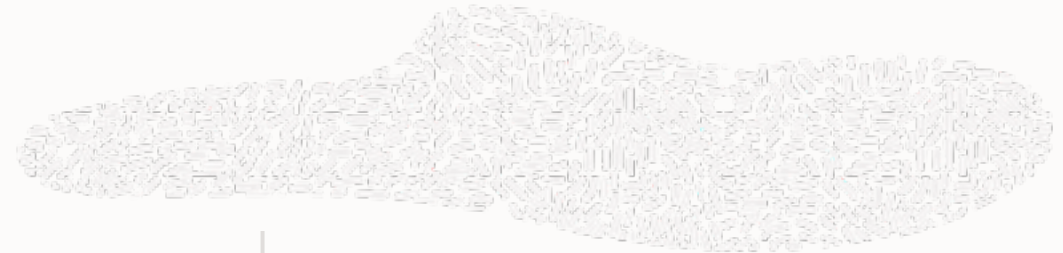
Photo by Andrea Brataas on Unsplash

# About me

- Alina Yurenko / @alina\_yurenko
- Developer Advocate for GraalVM at Oracle Labs
- Love open source and communities 🤝
- Love both programming 🧑💻 & natural languages 🗣️



# GraalVM™



High-performance  
optimizing compiler



Ahead-of-Time (AOT)  
“Native Image” toolchain



Language Runtimes



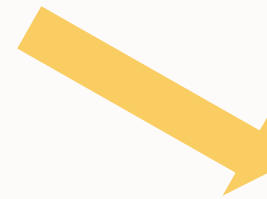


**GraalVM™**



**JIT**

`java MyMainClass`



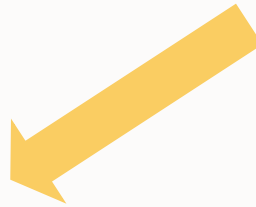
**AOT**

`native-image MyMainClass  
./mymainclass`



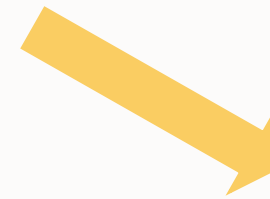


**GraalVM**<sup>TM</sup>



**JIT**

`java MyMainClass`

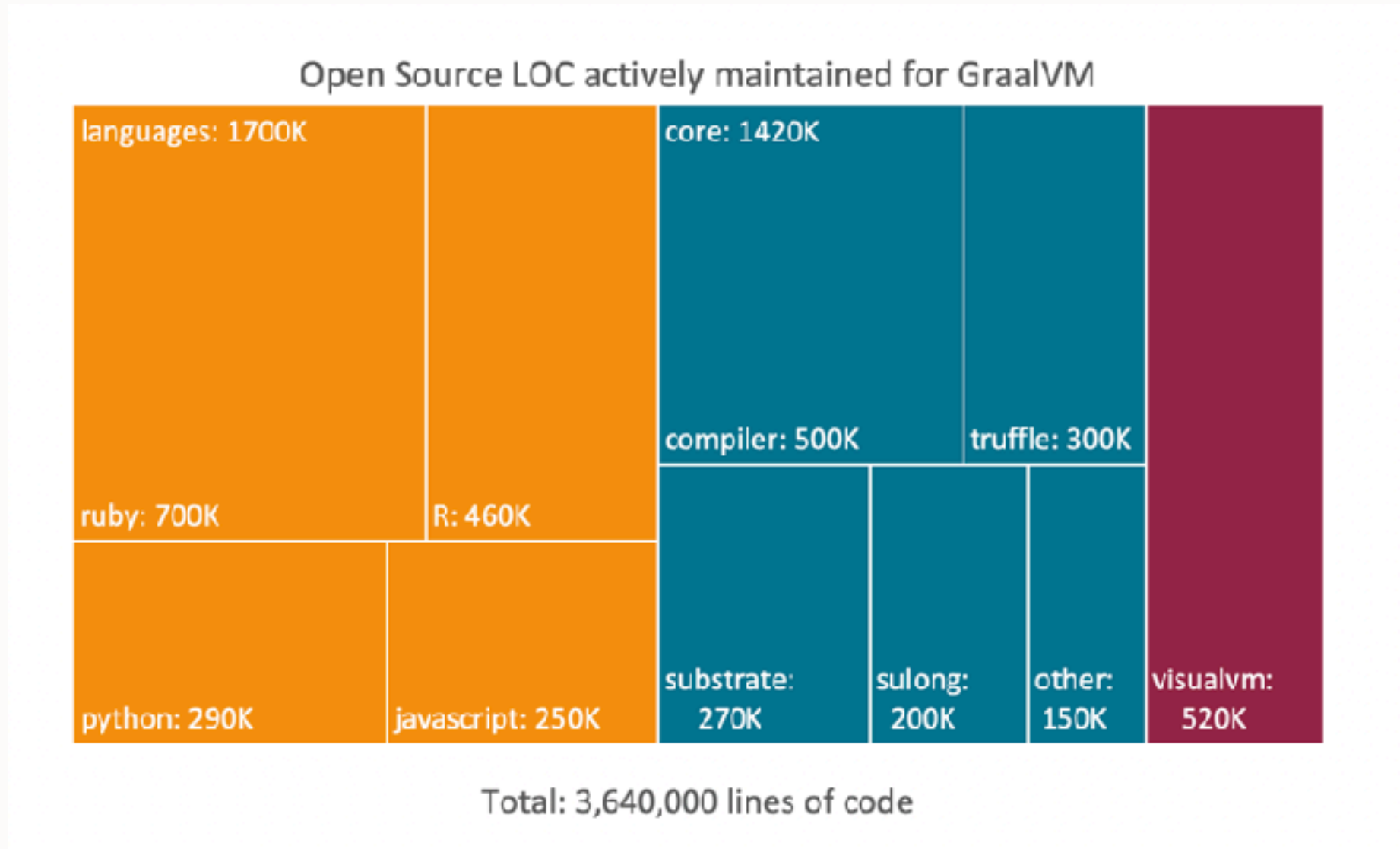


**AOT**

`native-image MyMainClass  
./mymainclass`



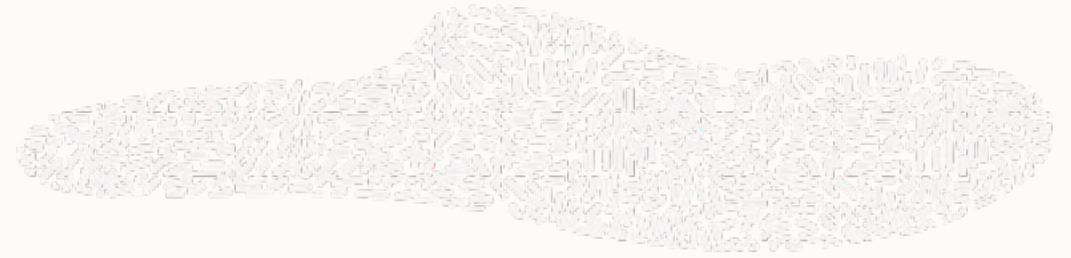
# Open source on GitHub: [github.com/oracle/graal](https://github.com/oracle/graal)



# GraalVM Native Image

# GraalVM Native Image

- Enables compiling Java programs into standalone native executables
- Performs static analysis to identify all code reachable from the entry point
- Instant startup, low memory footprint, perfect for cloud deployments
- Integrations with Java microservices frameworks





# Native Image Build Process

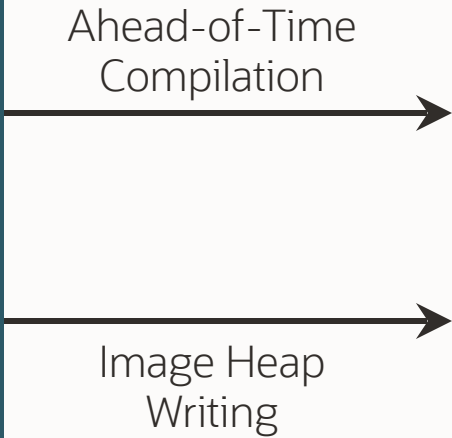
**Input:**  
All classes from application,  
libraries, and VM

- Application
- Libraries
- JDK
- Substrate VM



Iterative analysis until  
fixed point is reached

**Output:**  
Native executable



# AOT vs JIT: Startup Time

## JIT

Load JVM executable

Load classes from file system

Verify bytecodes

Start interpreting

Run static initializers

First tier compilation (C1)

Gather profiling feedback

Second tier compilation (GraalVM or C2)

Finally run with best machine code

## AOT

- Load executable with prepared heap
- Immediately start with optimized machine code

# AOT vs JIT: Memory Footprint

## JIT

Loaded JVM executable

Application data

Loaded bytecodes

Reflection meta-data

Code cache

Profiling data

JIT compiler data structures

## AOT

- Loaded application executable
- Application data

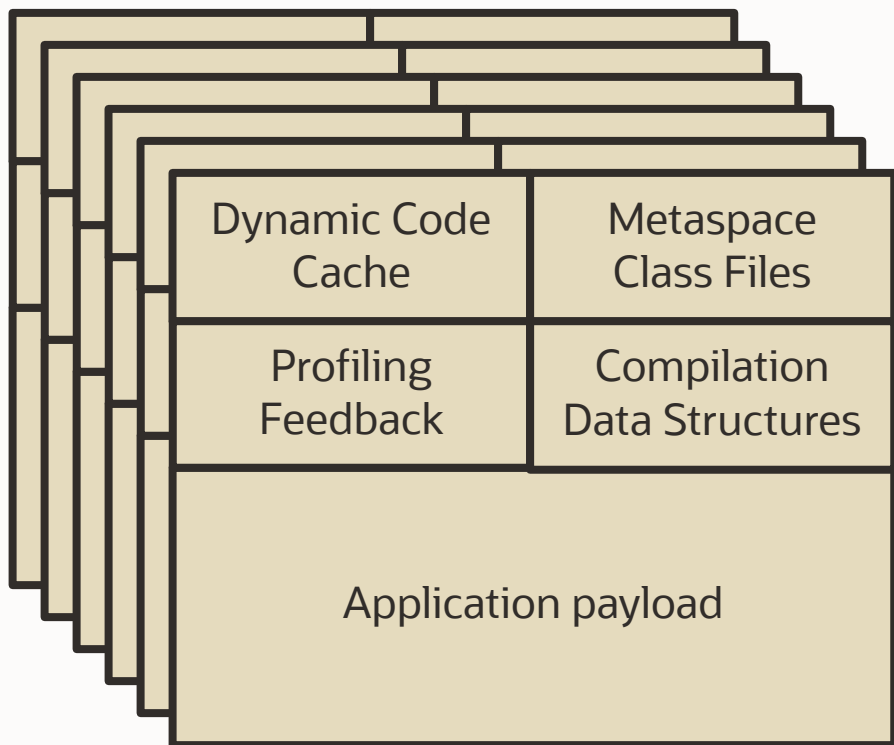
# Memory Scalability

## JIT

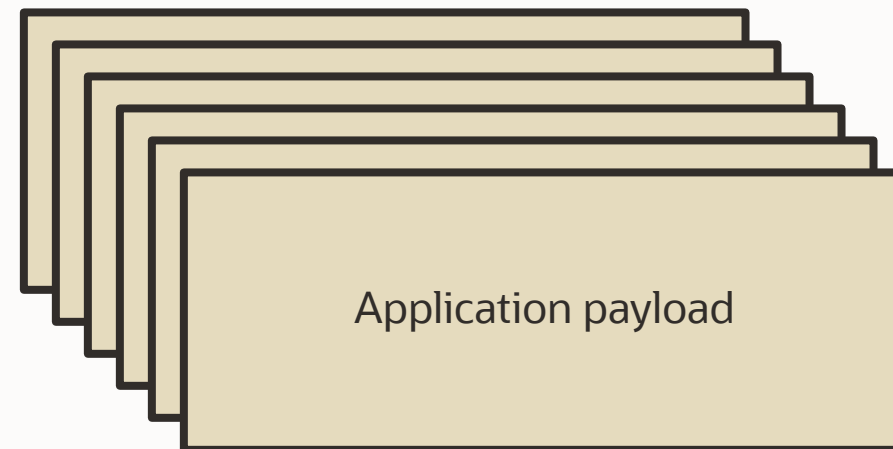
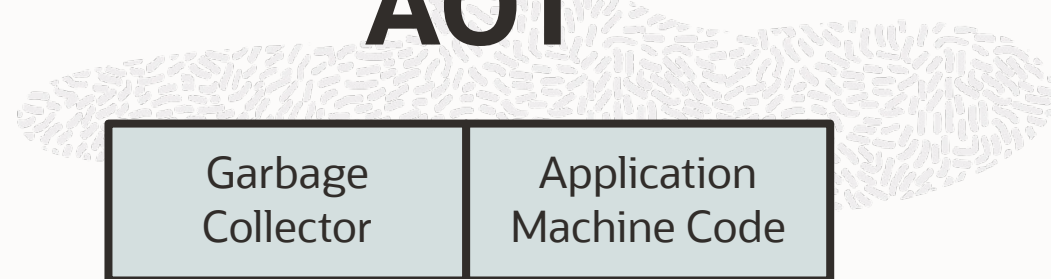


**shared**

**duplicated  
per process**



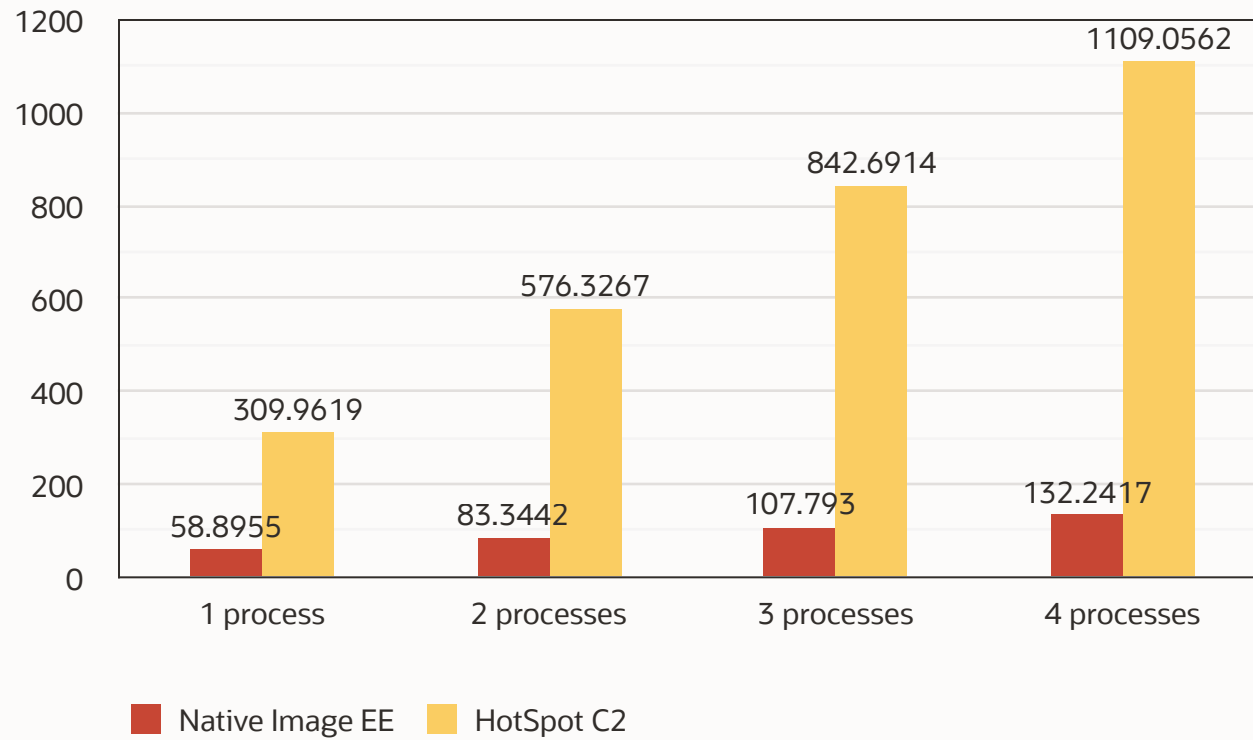
## AOT



# 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



# 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

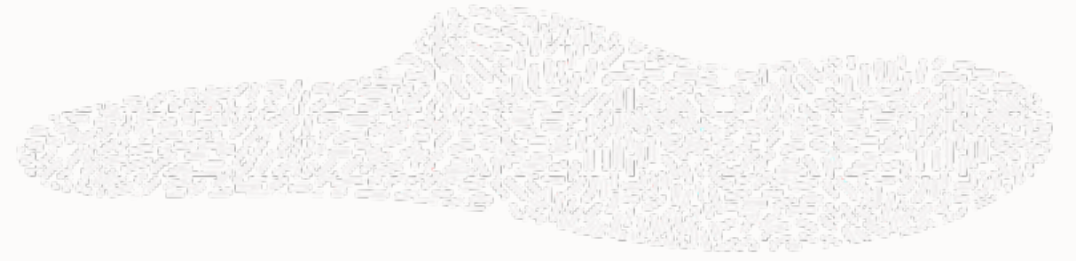
```
<plugin>  
  <groupId>org.graalvm.buildtools</groupId>  
  <artifactId>native-maven-plugin</artifactId>  
</plugin>
```

# Demo: Testing Native Image applications





# 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.

# Native Integration Tests



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

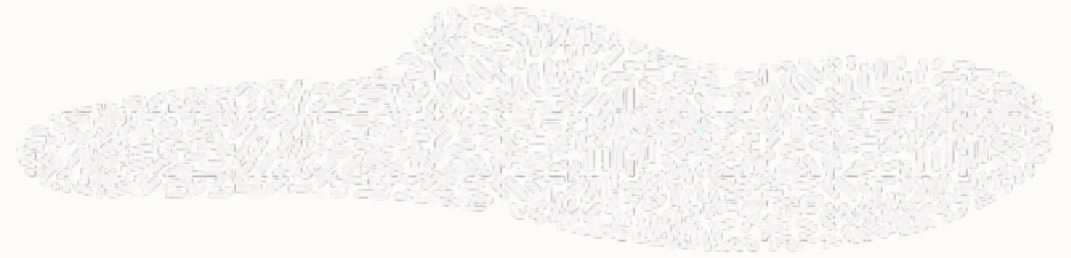
[Micronaut Test Resources](#) simplifies testing of applications which depend on external resources, by handling the provisioning and lifecycle of such resources automatically. For example, if your application requires a MySQL server, in order to test the application, you need a MySQL database to be 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



# GraalVM & Reflection?



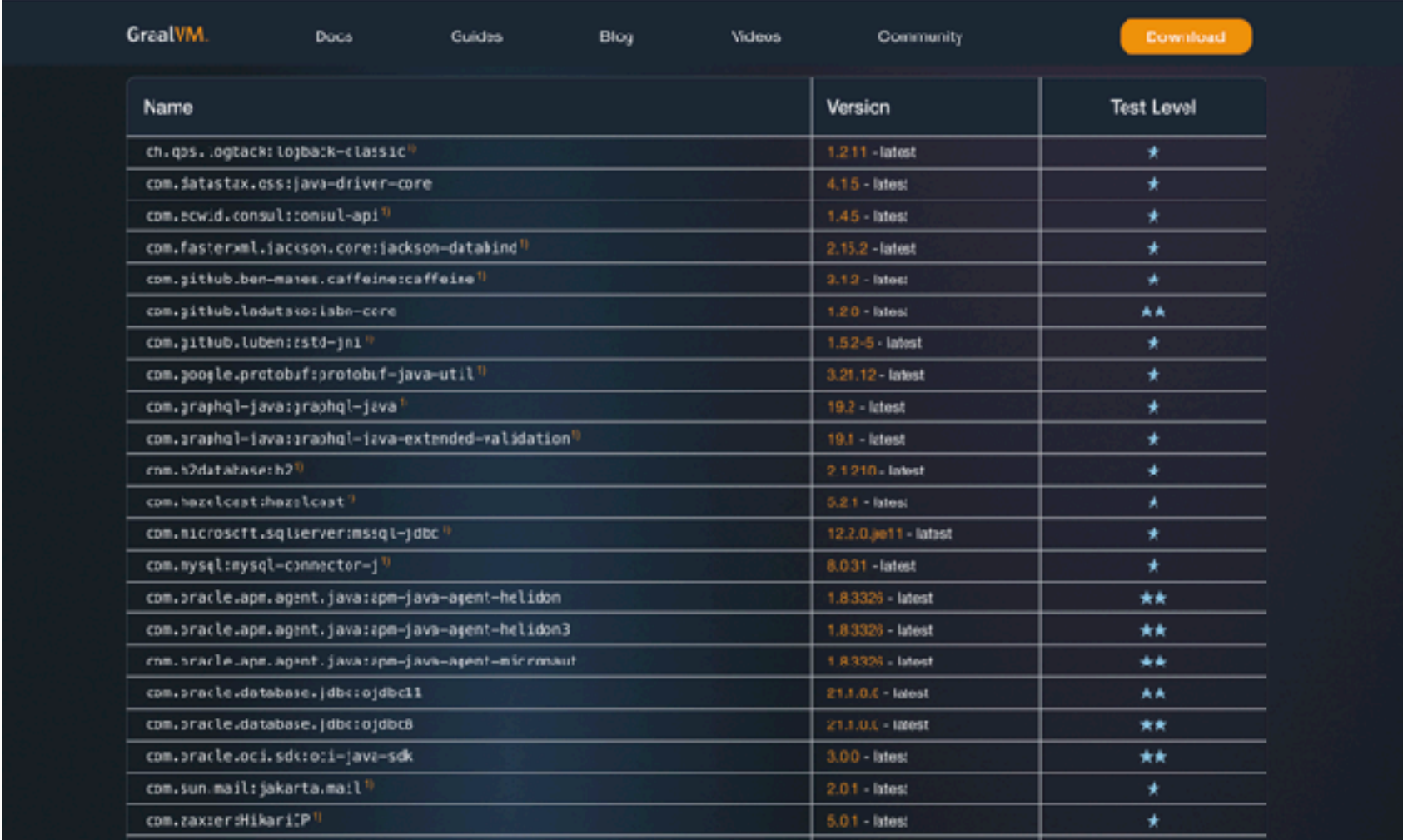
- GraalVM 🤝 Reflection!
- Native Image tries to resolve the target elements through a static analysis that detects calls to the Reflection API
  - If the analysis can not automatically detect your use of reflection, you might need additional configuration
- Trace reflection, JNI, resource usage on the JVM with the tracing agent:
  - Agent to record usage and produce configuration files for native images
    - `java -agentlib:native-image-agent=config-output-dir=META-INF/native-image ...`
  - Manual adjustment / addition might still be necessary
- Many frameworks & libraries ship reflection config that will be automatically picked up



# GraalVM & Reflection: demo



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



Name	Version	Test Level
ch.qos.logback:logback-classic <sup>1)</sup>	1.2.11 - latest	★
com.datastax.oss:java-driver-core	4.15 - latest	★
com.ecwid.consul:consul-api <sup>1)</sup>	1.45 - latest	★
com.fasterxml.jackson.core:jackson-databind <sup>1)</sup>	2.15.2 - latest	★
com.github.ben-manes.caffeine:caffeine <sup>1)</sup>	3.1.2 - latest	★
com.github.lodatos:labo-core	1.2.0 - latest	★★
com.github.luben:zstd-jni <sup>1)</sup>	1.5.2-5 - latest	★
com.google.protobuf:protobuf-java-util <sup>1)</sup>	3.21.12 - latest	★
com.graphql-java:graphql-java <sup>1)</sup>	19.2 - latest	★
com.graphql-java:graphql-java-extended-validation <sup>1)</sup>	19.1 - latest	★
com.h2database:h2 <sup>1)</sup>	2.1.210 - latest	★
com.hazelcast:hazelcast <sup>1)</sup>	5.2.1 - latest	★
com.microsoft.sqlserver:mssql-jdbc <sup>1)</sup>	12.7.0.jre11 - latest	★
com.mysql:mysql-connector-j <sup>1)</sup>	8.0.31 - latest	★
com.oracle.apm.agent.java:apm-java-agent-helidon	1.8.3325 - latest	★★
com.oracle.apm.agent.java:apm-java-agent-helidon3	1.8.3325 - latest	★★
com.oracle.apm.agent.java:apm-java-agent-microsaut	1.8.3325 - latest	★★
com.oracle.database.jdbc:ojdbc11	21.1.0.4 - latest	★★
com.oracle.database.jdbc:ojdbc8	21.1.0.4 - latest	★★
com.oracle.oai.sdk:oci-java-sdk	3.0.0 - latest	★★
com.sun.mail:jakarta.mail <sup>1)</sup>	2.0.1 - latest	★
com.zaxxer:HikariCP <sup>1)</sup>	5.0.1 - latest	★

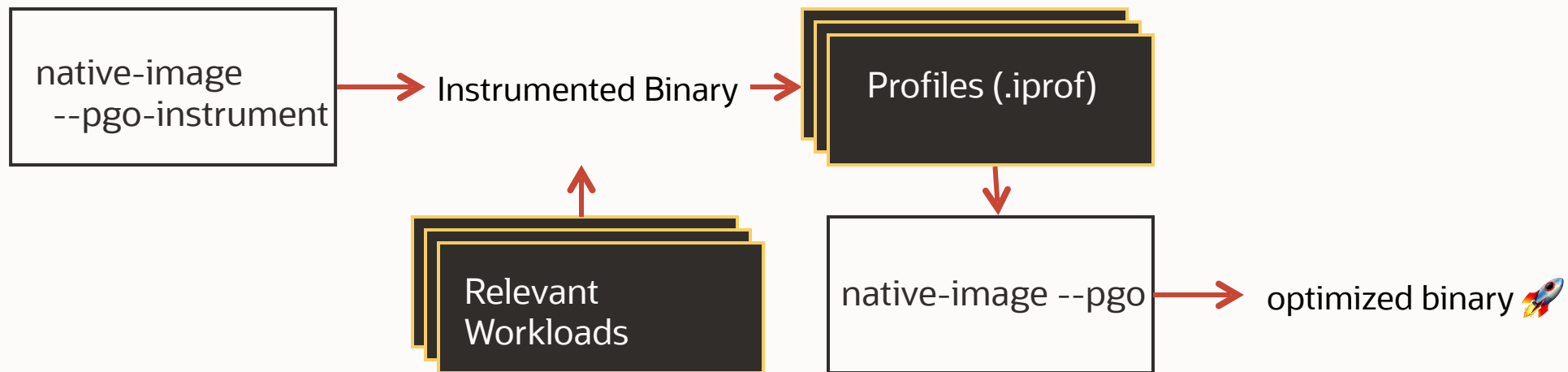
[graalvm.org/native-image/libraries-and-frameworks](https://graalvm.org/native-image/libraries-and-frameworks)

# 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>
```

# Optimizing Performance 🚀

# Optimizing performance of native image



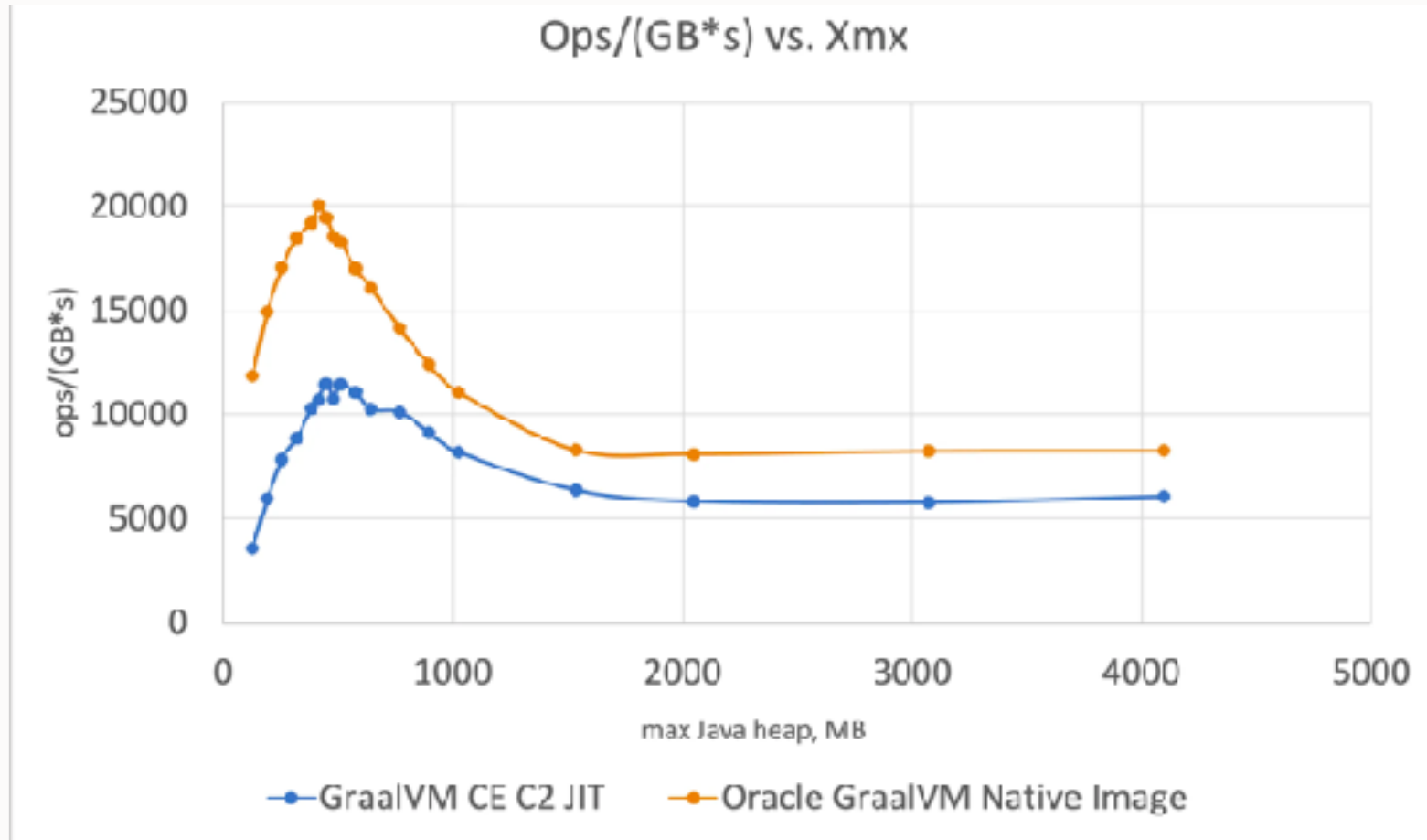


# AOT at the speed of JIT 🚀

Metric/runtime	GraalVM CE with C2 JIT	Oracle GraalVM Native Image	
Memory Usage (max RSS)	1,029 MB	641 MB	<b>-38% lower</b>
Peak throughput	11,066 req/s	11,902 req/s	<b>+8% higher</b>
Throughput per memory	12,488 req/(GB*s)	18,569 req/(GB*s)	<b>+49% better</b>
Tail latency (P99)	7.2ms	5.15ms	<b>-28% lower</b>
Startup	7,090ms	210ms	<b>34x faster</b>

Performance of Spring Petclinic with Oracle GraalVM Native Image, GraalVM CE Native Image, and GraalVM CE with C2 JIT.  
Benchmark details: <https://medium.com/graalvm/graalvm-for-jdk-21-is-here-ee01177dd12d>

# AOT at the speed of JIT 🚀



Performance of Spring Petclinic with Oracle GraalVM Native Image, GraalVM CE Native Image, and GraalVM CE with C2 JIT.  
Benchmark details: <https://medium.com/graalvm/graalvm-for-jdk-21-is-here-ee01177dd12d>



# Compressing native images with UPX

**Julien Dubois** @julienubois

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 🤖

```
Thu Dec 10 12:46:17 CET 2020
└─ Downloads mkdir original
└─ Downloads mv Functionapp_20201230150.zip original
└─ Downloads cd original
└─ original time unzip Functionapp_20201230150.zip
Archive: Functionapp_20201230150.zip
  creating: foobar/
  inflating: host.json
  inflating: spring-native-image
  inflating: foobar/function.json
unzip Functionapp_20201230150.zip  0.41s user 0.93s system 98% cpu 0.444 total
└─ original ll spring-native-image
-mor-@-@ 1 julien  staff   624 Dec  3 08:34 spring-native-image
└─ original ll

$ cd Downloads
└─ cd Downloads
└─ Downloads mkdir upx
└─ Downloads mv Functionapp_202012101848.zip upx
└─ Downloads cd upx
└─ upx time unzip Functionapp_202012101848.zip
Archive: Functionapp_202012101848.zip
  extracting: host.json
  extracting: spring-native-image
  creating: foobar/
  extracting: foobar/function.json
unzip Functionapp_202012101848.zip  0.06s user 0.01s system 98% cpu 0.077 total
└─ upx ll spring-native-image
-mor-@-@ 1 julien  staff   174 Dec 10 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)

UPX Compression levels	Time (ms) to run eragen -help [Linux]
none	21,9
-j	277,6
-best	262,1
-lma	1250
java-jar	1153

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

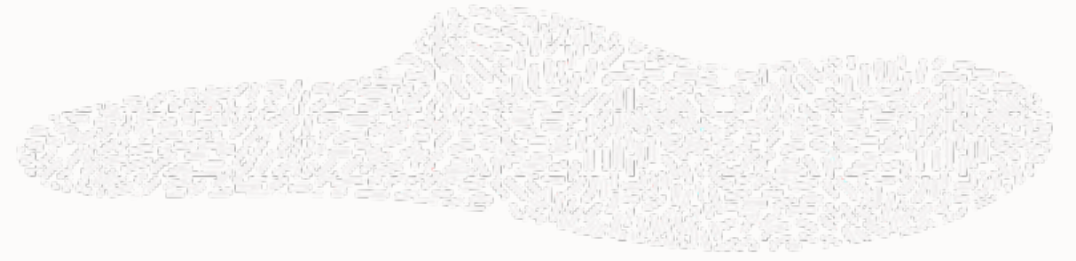
- statically link against all libraries except libc
- great for deploying such native images on distroless container images



## Reduced Attack Surface



- 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

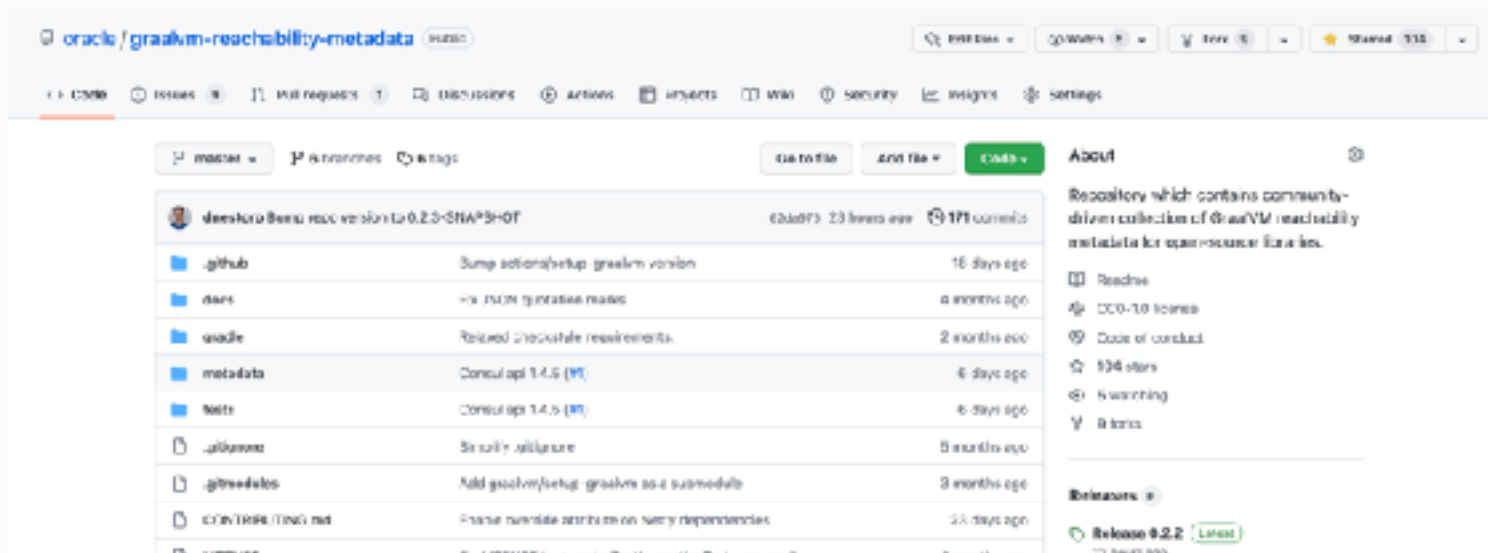


# What's the catch?

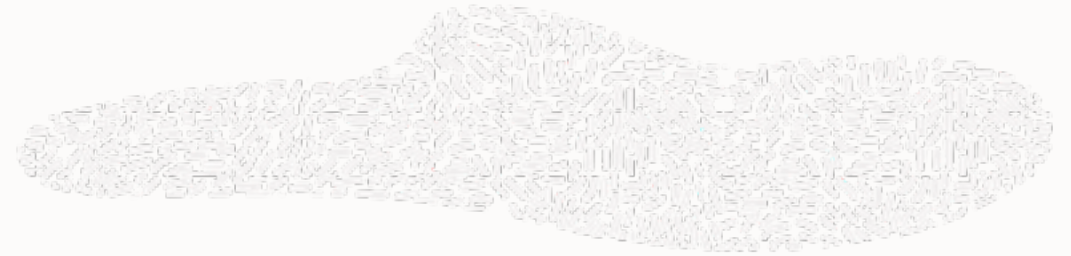
# GraalVM & Reflection?



- 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

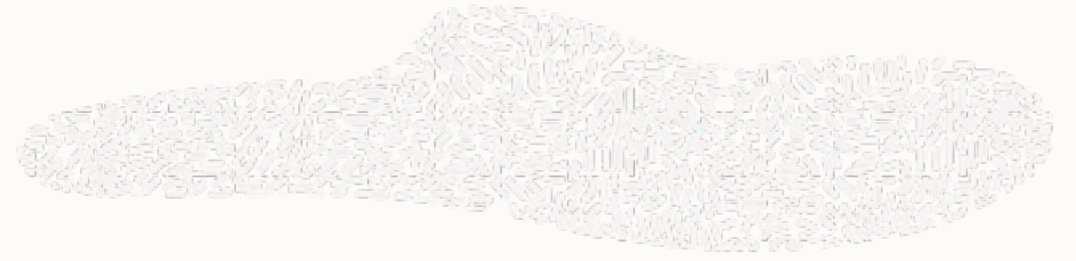


# Required Build Time Step



- Computational effort necessary at build time
- Need a powerful machine with the same target architecture & OS
  - Use GraalVM with GitHub Actions: [github.com/marketplace/actions/github-action-for-graalvm](https://github.com/marketplace/actions/github-action-for-graalvm)
  - Many larger apps can build with 2 GB of memory
- Develop in JIT mode for fast development, only use AOT for final deployment
- For best throughput, use profile-guided optimizations





# What's new in GraalVM

# GraalVM for JDK 21



GraalVM™

# Introducing GraalOS



## Layered Native Images #7626

Open christianwimmer opened this issue 5 days ago · 4 comments



christianwimmer commented 5 days ago · edited by friephaus · Member

Allow a native image (the application image) to depend on multiple base images that can be shared between multiple applications.

### Goals

- Reduce the combined RSS memory size when many different applications that share common base frameworks and libraries run on the same machine.
- Reduce image build time for the application (not for the layered images themselves, or the total time necessary to build all layer dependencies): Allow incremental building of native images where a base image for the core dependencies of the application are pre-built and for application code changes, only the application code needs to be considered.
- Provide a memory efficient and secure integration into [GraalOS](#), but also work outside of GraalOS (at first on Linux, eventually also on MacOS and Windows).
- Guarantee the same programming model restrictions as standalone images. We want to be able to "upgrade" a layered application image to a standalone image in GraalOS when that application is frequently used.

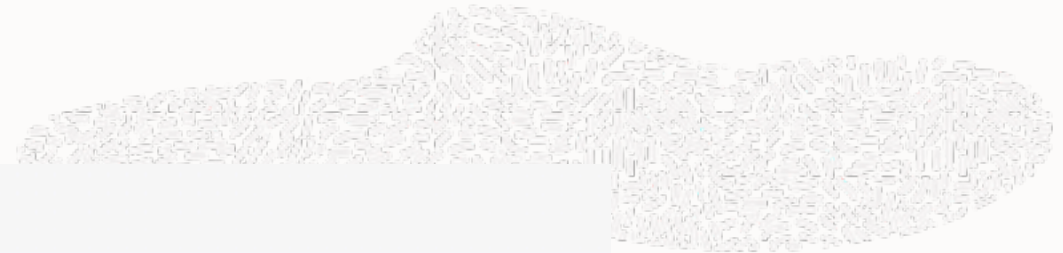
### Non-Goals

- Layered images are no general purpose standard shared libraries that can be used from other languages (like from C).
- No interchangeability of images: We do not allow building different base images and mix / match different such images at run time. When building an application, the base images it depends on are fixed at build time. When a base image is rebuilt, all applications that depend on the base image need to be rebuilt too. This includes minor bugfixing releases of a framework or the JDK.
- No overall build time improvements: We accept that the sum of the image build time of the base images and the

<https://github.com/oracle/graal/issues/7626>



# GraalVM Community roadmap on GitHub



**GraalVM Community Roadmap**

Native Image + Compiler | Language Runtimes | + New View

focus-areas: "Native Image + Compiler"

Title	Assignees	Status	Labels	Notes
<b>0 GraalVM for JDK 21 (September 19, 2023)</b>				
18 Adapt support for JDK 21 #3854	wilhi	In Progress		
19 Deprecate the GraalVM Updater #3855	frlepheus	In Progress	bug	
20 Reporting missing metadata in Native Image by throwing spe... #5171	leicorbet and vjovanov	In Progress	feature	
21 Safe Composition of Metadata #5173	leicorbet and vjovanov	In Progress	feature	
22 [66-47647] Explicit Experimental Option Handling #7126	frlepheus	Done	native-image	
+ Cannot add items when grouped by milestone				
<b>Planned for the Future (18)</b>				
23 Support for AArch64 (darwin-aarch64) #2669	giles-dubocsq and frl...	In Progress	feature platform-darwin	
24 Advance the Programming Model for Application Initialization... #4922	christianwimmer	In Progress	feature	
25 Improve AWT Support #4921	christianwimmer	In Progress	feature	
26 Parse bytecode only once when building a native image #4923	christianwimmer	In Progress	feature	
27 Always run the native image generator on the module path #4924	christianwimmer	In Progress	feature	
28 Remove scanning and loading of all classes at beginning of ... #2599	christianwimmer and rjc...	In Progress	feature native-image	
29 Planned Upgrades to Debug Info Support #5047	edinn and frlepheus	Done	feature native-image-debuginfo	Contributions from Red Hat
30 Windows Debug Information for local variables. #5335	stooke	In Progress	feature redhat-interest	Contributions from Red Hat
31 JDWP-based Debugging Support #5648	giles-dubocsq	In Progress	feature	
32 JFR Support in Native Image #5470	christianwimmer, frlep...	In Progress	feature native-image-jfr redhat-interest	Contributions from Red Hat

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



# Get started with GraalVM

## Get started with GraalVM

[graalvm.org](https://graalvm.org)

or

```
sdk install java 21-graal
```

## Questions & let's connect!



## GraalVM resources





# Thank you!

—  
**Alina Yurenko**

[@alina\\_yurenko](https://twitter.com/alina_yurenko)

