Supercharge your Native Image applications in 5 steps

Alina Yurenko

Developer Advocate for GraalVM
Oracle Labs

Voxxed Days Bucharest

Hari Nandakumar @ Unsplash

About me

- Alina Yurenko / @alina_yurenko
- Developer Advocate for GraalVM at Oracle Labs
- Love open source and communities 🤝
- Love both programming 👩‍💻 & natural languages 🗣️
- Ukrainian

Photo credit: Volkan Vardar @ unsplash.com/photos/9YG4wHcFvjg

What GraalVM offers

More performance with the Graal compiler

- Run your Java application faster
- New JIT compiler optimizations

Fast startup with Native Image

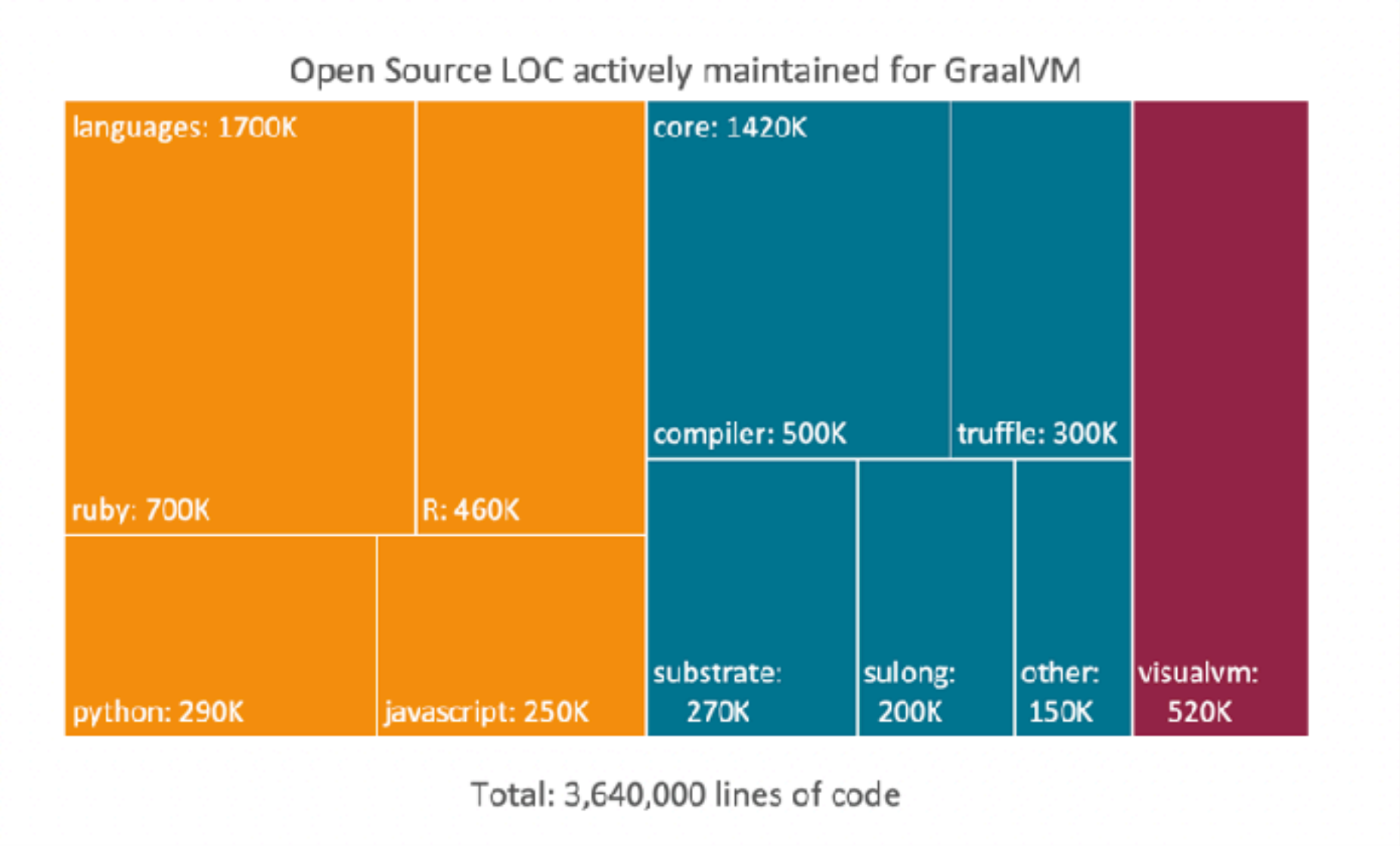
- Create standalone binaries with low footprint
- Instant performance

GraalVM™

Polyglot VM

- Interop: extend your Java application with libraries from JavaScript, Python, R…
- High performance for all languages
- Polyglot tooling

Open source on GitHub: 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

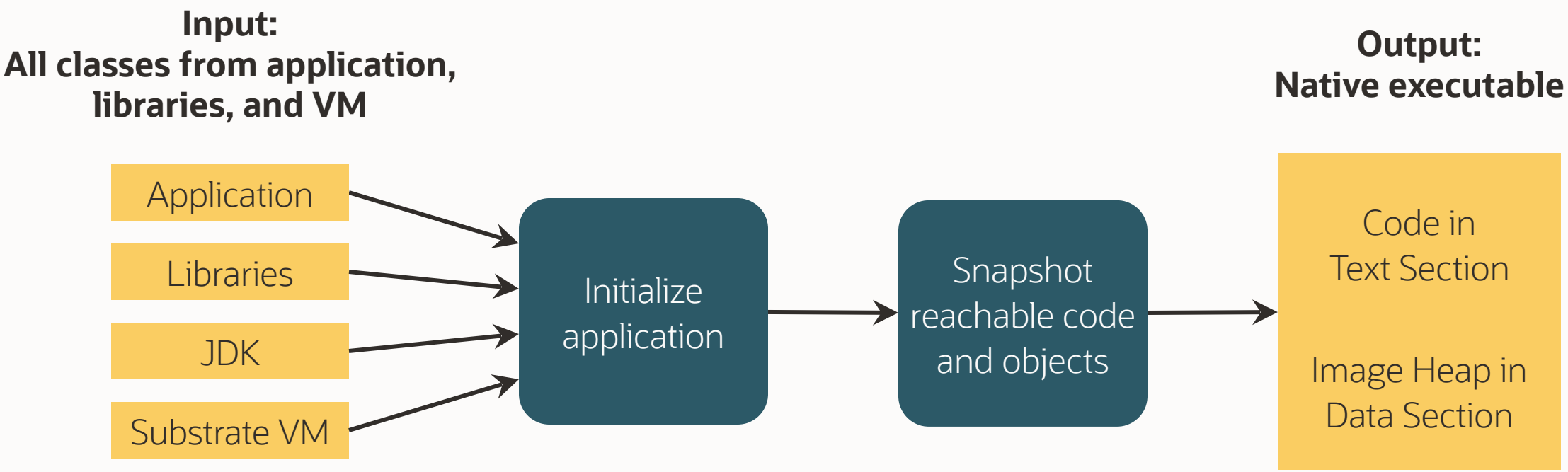
QUARKUS

MICRONAUT

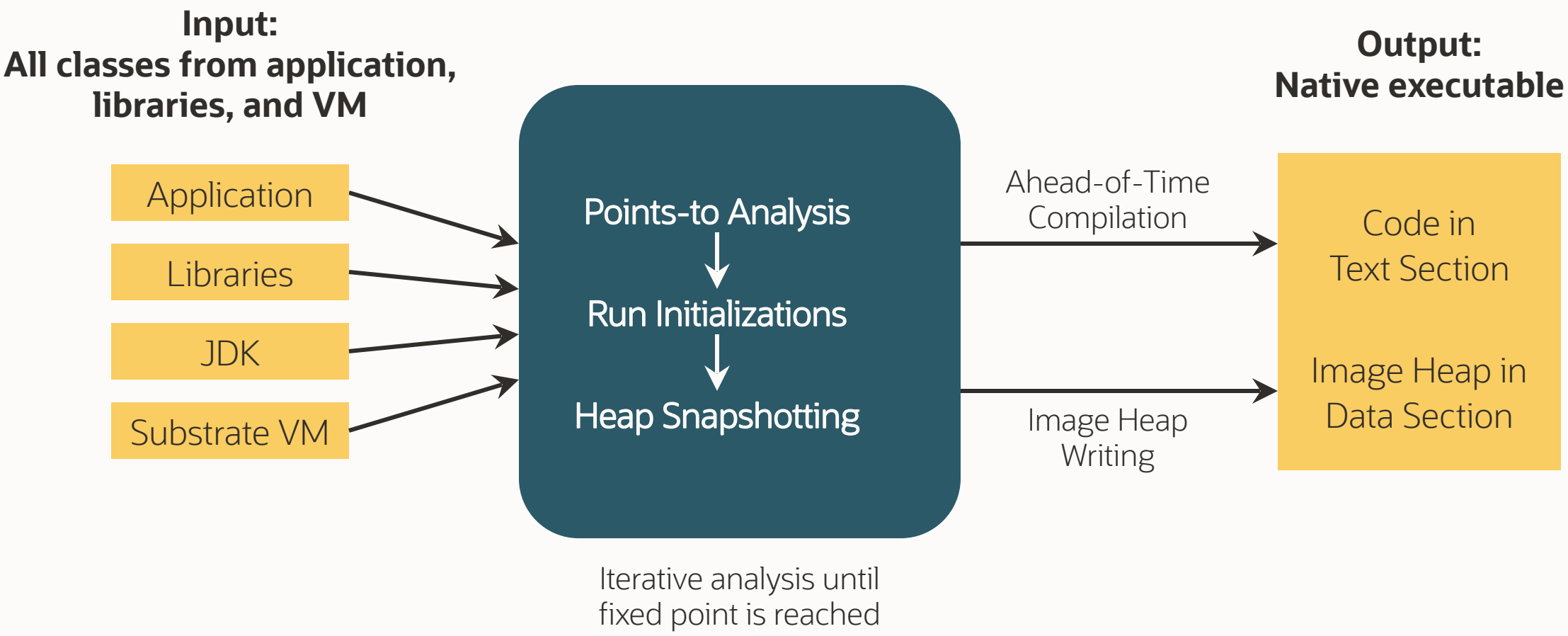
spring

helidon.io

Native Image Build Process: the idea



Native Image Build Process



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

JIT

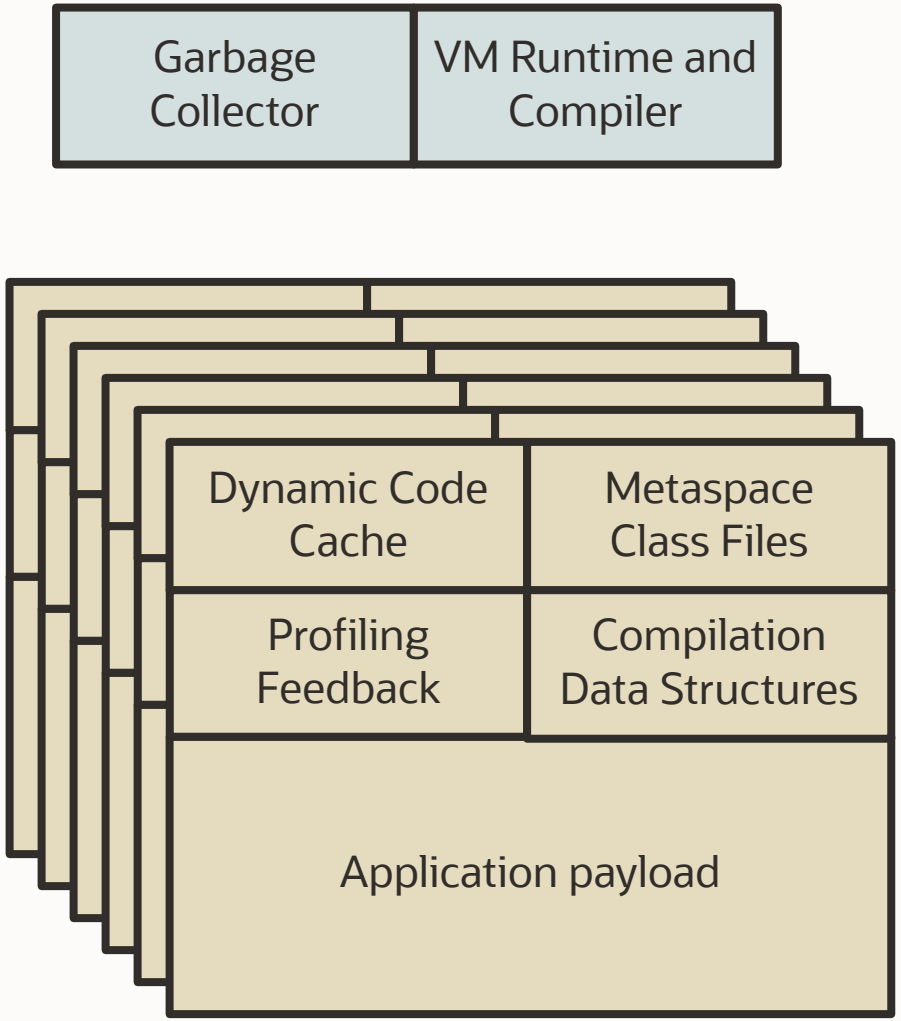
Garbage Collector	Virtual Machine Runtime and Compiler
Dynamic Code Cache	Metaspace Class Files
Profiling Feedback	Compilation Data Structures
Application payload	

AOT

Garbage Collector	Application Machine Code
Application payload	

Memory Scalability

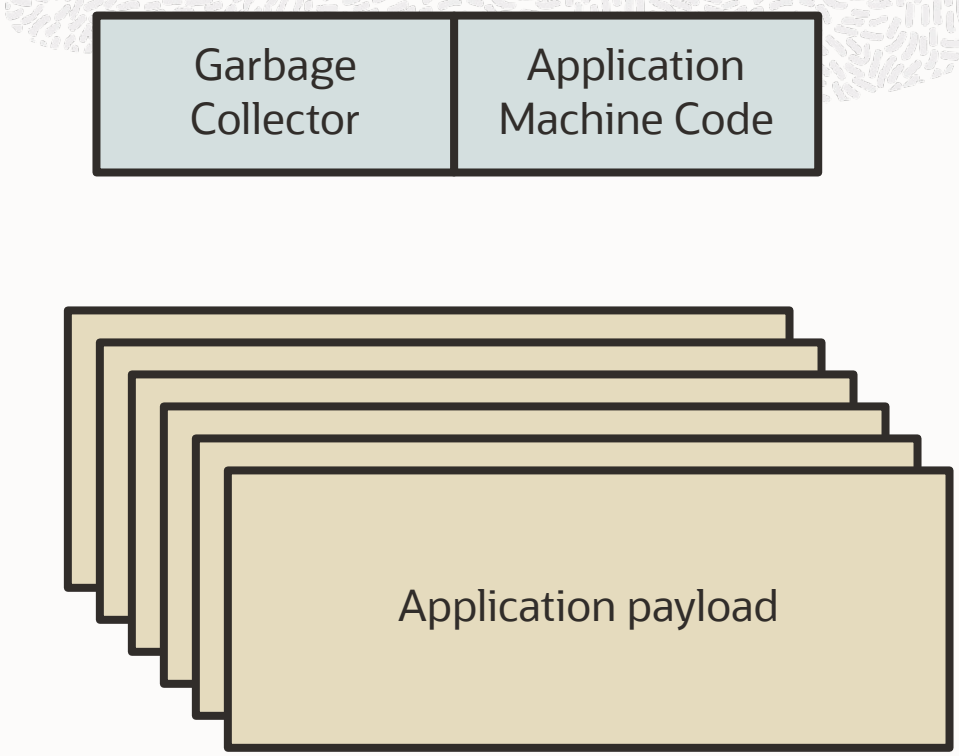
JIT



shared

duplicated per process

AOT



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"
}
```

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.

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

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

GraalVM & Reflection: demo

What about reflection in 3rd-party libraries?

oracle / graalvm-reachability-metadata Public

Edit Pins | Watch 5 | Fork 9 | Starred 96

Code | Issues 11 | Pull requests 5 | Discussions | Actions | Projects | Wiki | Security | Insights | Settings

master | 5 branches | 4 tags | Go to file | Add file | Code

dnestore Merge pull request #65 from sdeleeze/metadata-version-fix ... a6bd8f3 2 hours ago 158 commits

.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

About

Repository which contains community-driven collection of GraalVM reachability metadata for open-source libraries.

- Readme
- CC0-1.0 license
- Code of conduct
- 96 stars
- 5 watching
- 9 forks

Releases 4

0.2.0 Latest
9 days ago

+ 3 releases

Contributors 6

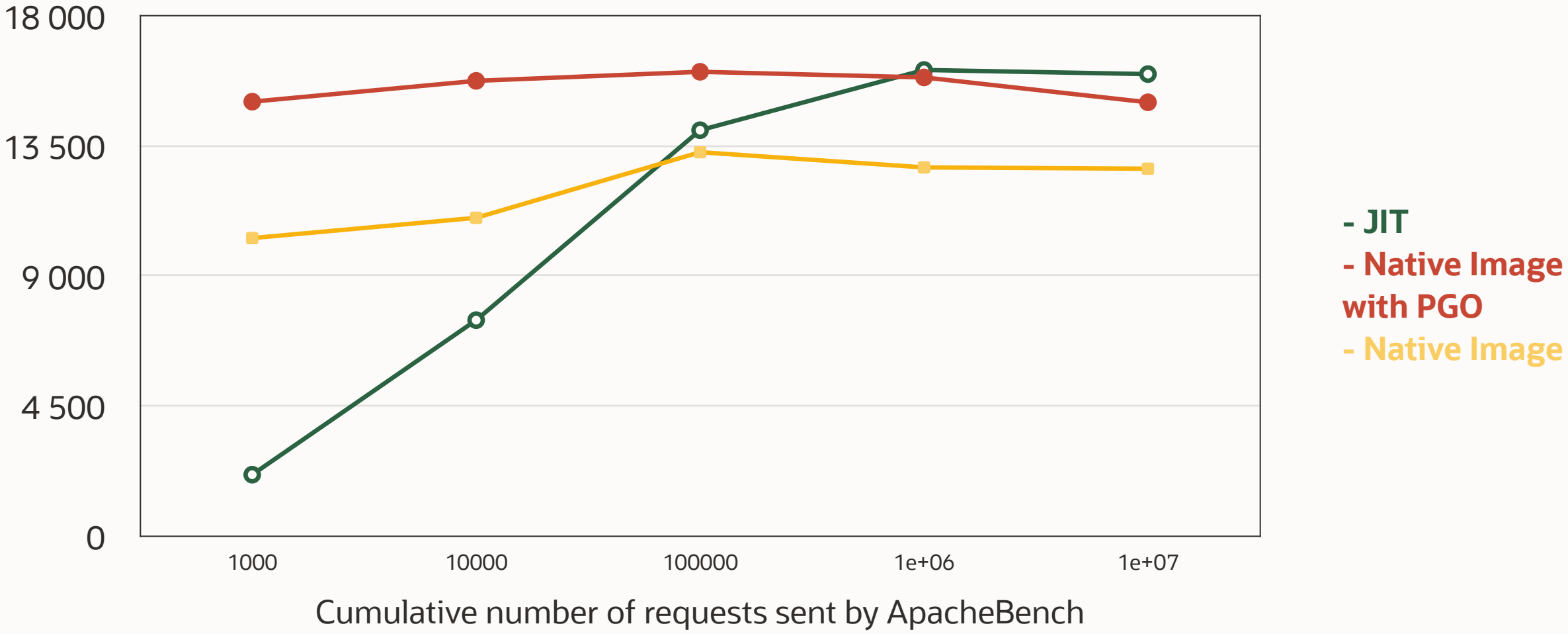
Is there an easier way to handle reflection? Yes!

```
<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>
```

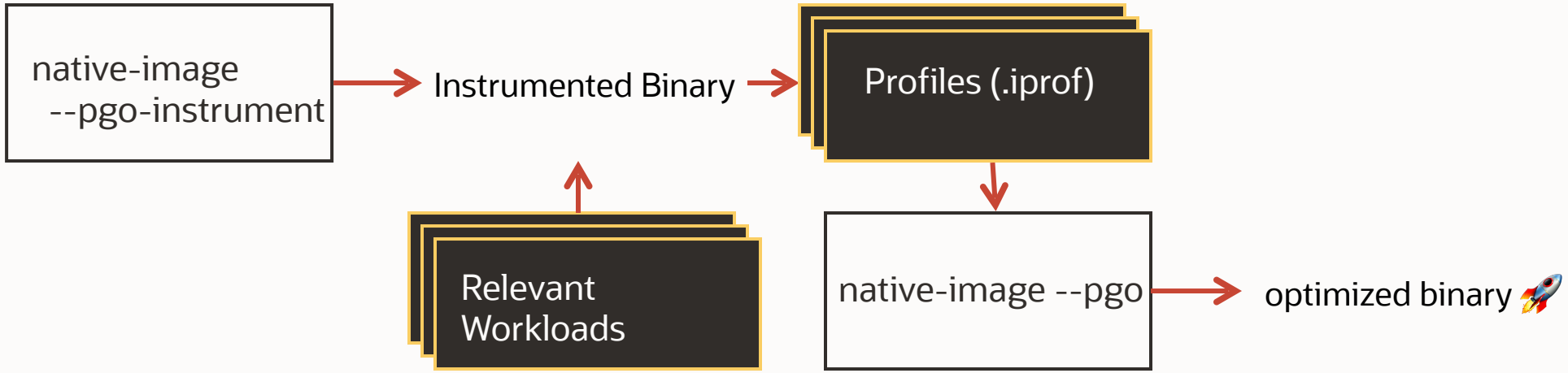
Demo: using metadata

Optimizing Performance 🚀

AOT vs JIT: Throughput



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

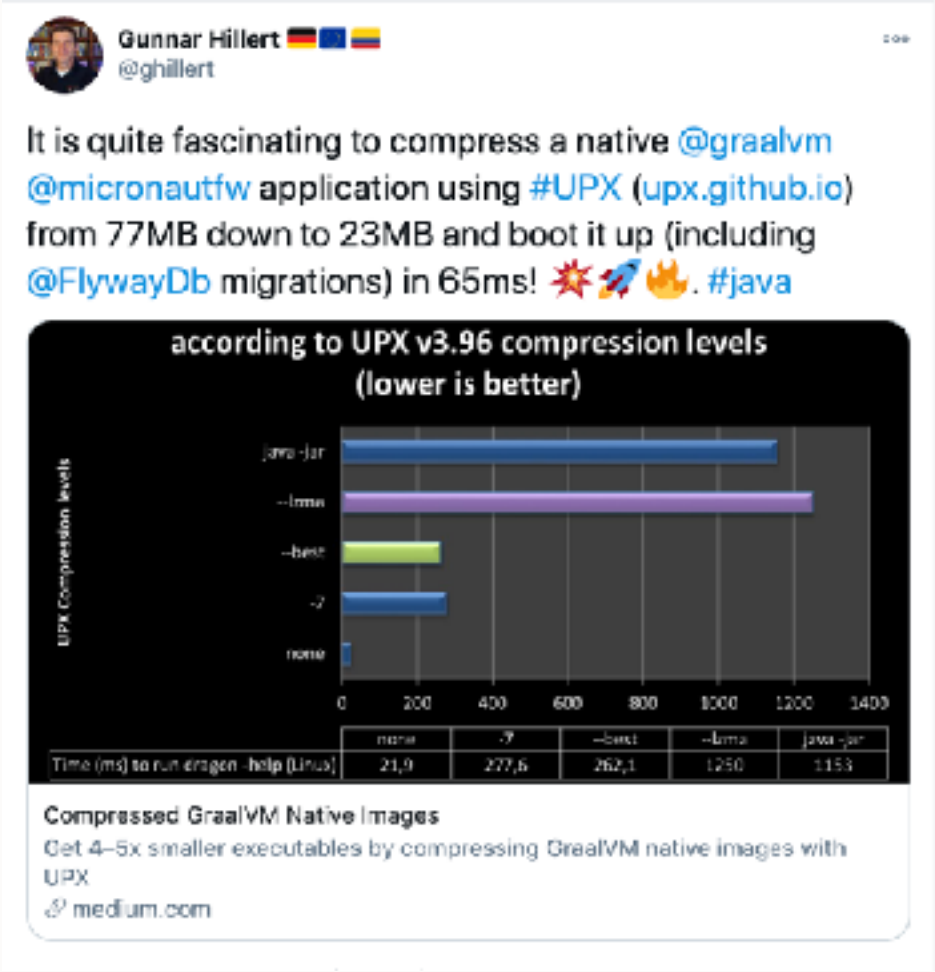
- Monitor and optimize performance of native images in production deployments
- include JFR at image build time:

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

- To enable JFR and start a recording:

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

Compressing native images with UPX



** more aggressive compression algorithms can have runtime impact*

Static and Mostly Static Images

Static native images

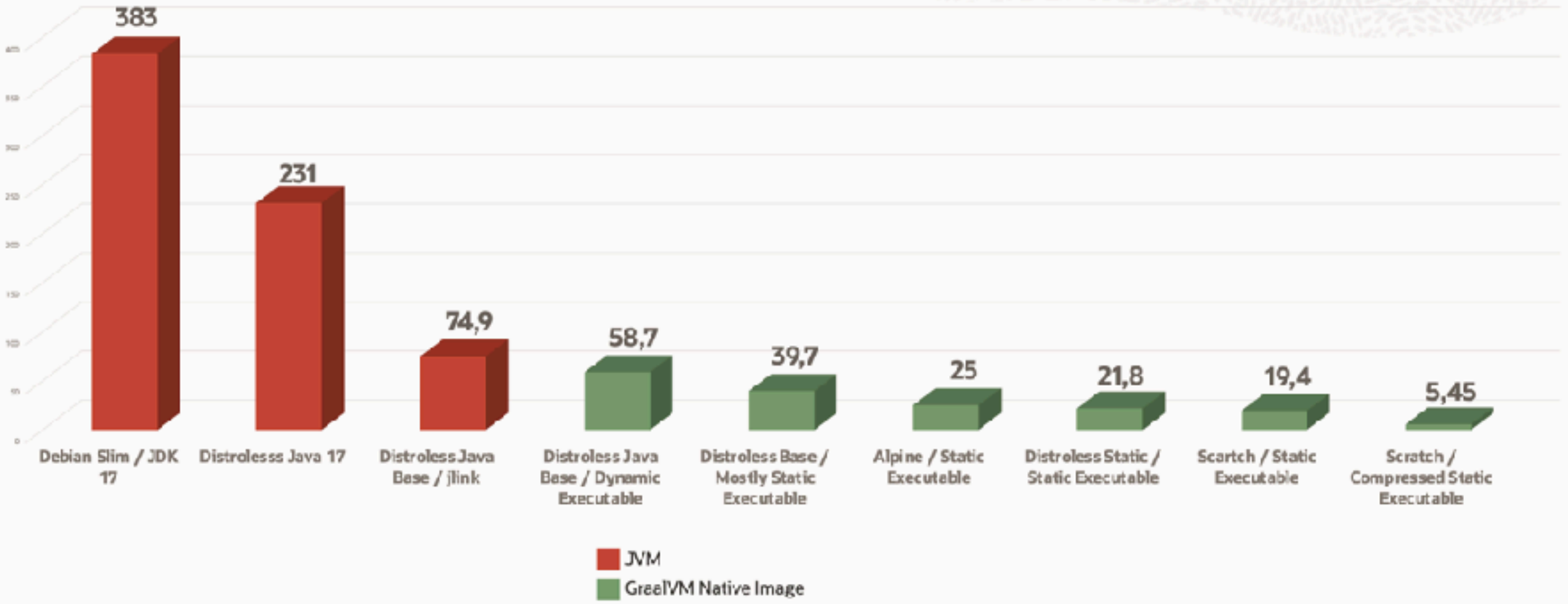
- statically linked against musl-libc , which can be used without any additional library dependencies
- great for deploying on slim or distroless container images

```
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

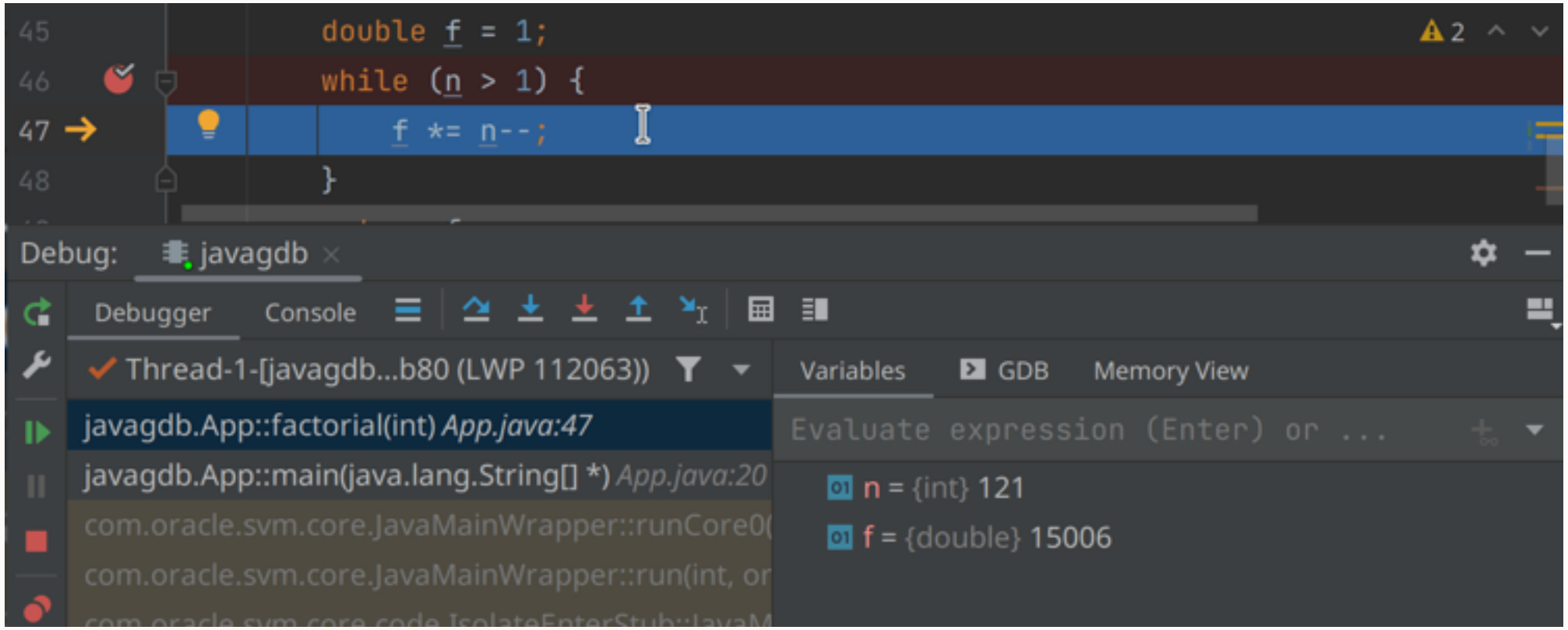
Reduced Attack Surface

- 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 new in GraalVM

Developer experience improvements: Debugging in IntelliJ IDEA 2022.2 EAP 5!🎉

- Attach Debugger action
- Stepping
- Local variables view
- Integration with Gradle and Maven build



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 (thanks to Red Hat for their contributions!)

Demo: monitoring Native Image applications

GraalVM™ JDK 20

dev builds

GraalVM Community roadmap on GitHub

GraalVM Community Roadmap

Native Image + Compiler | Language Runtimes | + New view

focus-area:"Native Image + Compiler" 19 ×

	Title	Assignees	Status	Labels	Notes
	22.3.0 Release (October 25, 2022) 7				
1	GraalVM JDK19 Builds #4957	giles-duboscq	In Progress	feature	
2	Support Virtual Threads (Project Loom) in Native Image #4920	peter-hofer	Done	feature	
3	Complete the Third-Party API in Native Image #4919	christianwimmer and olp...	In Progress	feature	
4	[GR-40674] Perf support #4811	adinn and olpaw	Done	native-image-debuginfo	Contributions from Red Ha
5	[GR-40264] Introduce `--enable-monitoring` option. #4823	fniephaus	Done	OCA Verified	
6	[GR-39475] [GR-40095] Initial jvmstat support in GraalVM Na #4803	christianhaeubl	Done	OCA Verified	
8	[GR-39497] Add support for JSON build output to GraalVM N: #4685	jerboaa	Done	oca-signed	Contributions from Red Ha
	Cannot add items when grouped by milestone				
	23.0.0 Release (January 24, 2023) 3				
15	[GR-40463] Add initial support for remote management over J #4732	roberttoyonaga	In Progress	feature native-image	Contributions from Red Ha
18	Add support for ZGC on HotSpot #5050	tkrodriguez	In Progress	compiler feature	
19	Add support for JDK19 and retire JDK11 support #5063		In Progress	feature	

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

What’s next for GraalVM

Add support for ZGC on HotSpot #5050

Open tknodriguez opened this issue on Sep 22, 2022 · 0 comments

tknodriguez 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

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

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
```

What GraalVM offers

More performance in JIT mode

- Run your Java application faster
- New JIT compiler optimizations

Fast startup with Native Image

- Create standalone binaries with low footprint
- Instant performance

GraalVM™

Polyglot VM

- Interop: extend your Java application with libraries from JavaScript, Python, R…
- high performance for all languages
- polyglot tooling

Thank you!

Presentation & resources:

Got questions?

SCAN ME