



ORACLE

GraalVM Enterprise 21.1.0

The universal VM

21th of April 2021

Wolfgang Weigend

Master Principal Solution Engineer | global Java Team

Java Technology & GraalVM and Architecture



Safe harbor statement

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, timing, and pricing of any features or functionality described for Oracle's products may change and remains at the sole discretion of Oracle Corporation.

GraalVM Native Image early adopter status

GraalVM Native Image technology (including SubstrateVM) is Early Adopter technology. It is available only under an early adopter license and remains subject to potentially significant further changes, compatibility testing and certification.



Agenda



- GraalVM in the Java SE Subscription
- GraalVM Enterprise Intro
- GraalVM Just-in-Time Compiler
- GraalVM Polyglot support for multiple languages
- GraalVM Enterprise Native Image
- GraalVM Enterprise for cloud native development
 - Java in Containers
- Summary



GraalVM Enterprise with Java SE Subscription

- Oracle Java SE Subscription now entitles customers to use Oracle GraalVM Enterprise at no additional cost
- Key benefits for Java SE Subscribers:
 - Native Image utility to compile Java to native executables that start almost instantly for containerized workloads
 - High-performance Java runtime with optimizing compiler that can improve application performance



ORACLE

GraalVM Enterprise



GraalVM Enterprise

High-performance runtime that provides significant improvements in application performance and efficiency



High-performance optimizing Just-in-Time (JIT) compiler



Multi-language support for the JVM



Ahead-of-Time (AOT) "native image" compiler



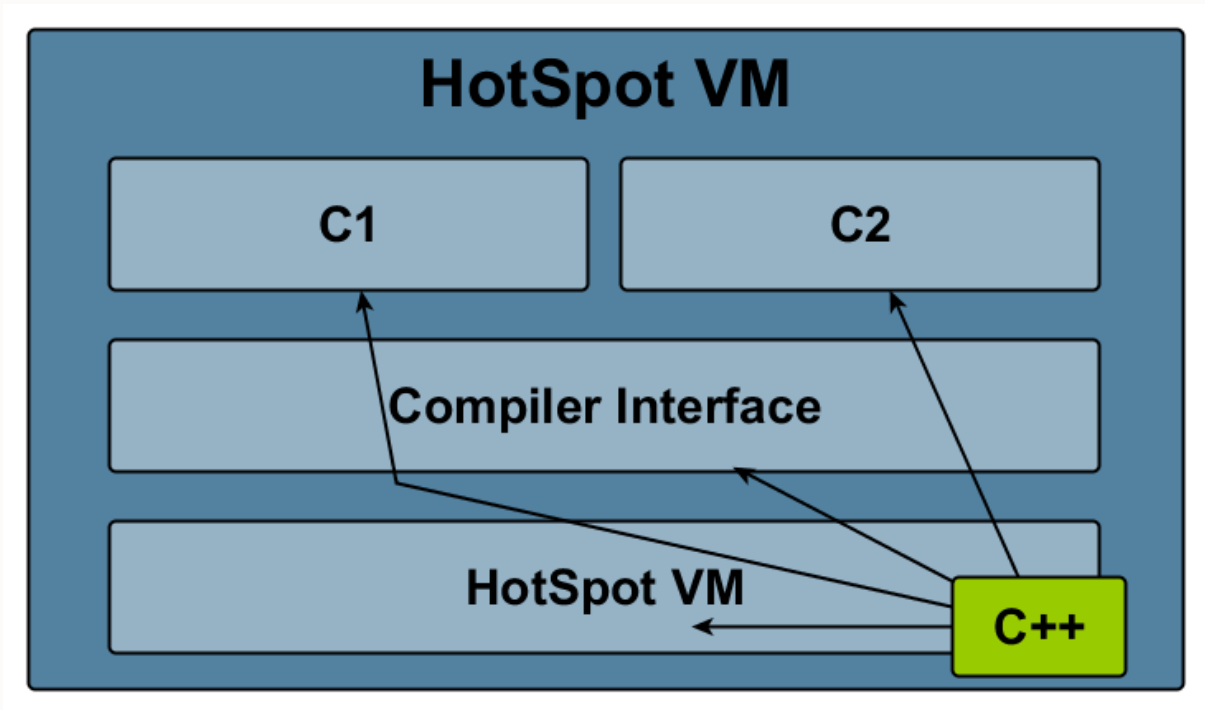
GraalVM JIT Compiler working



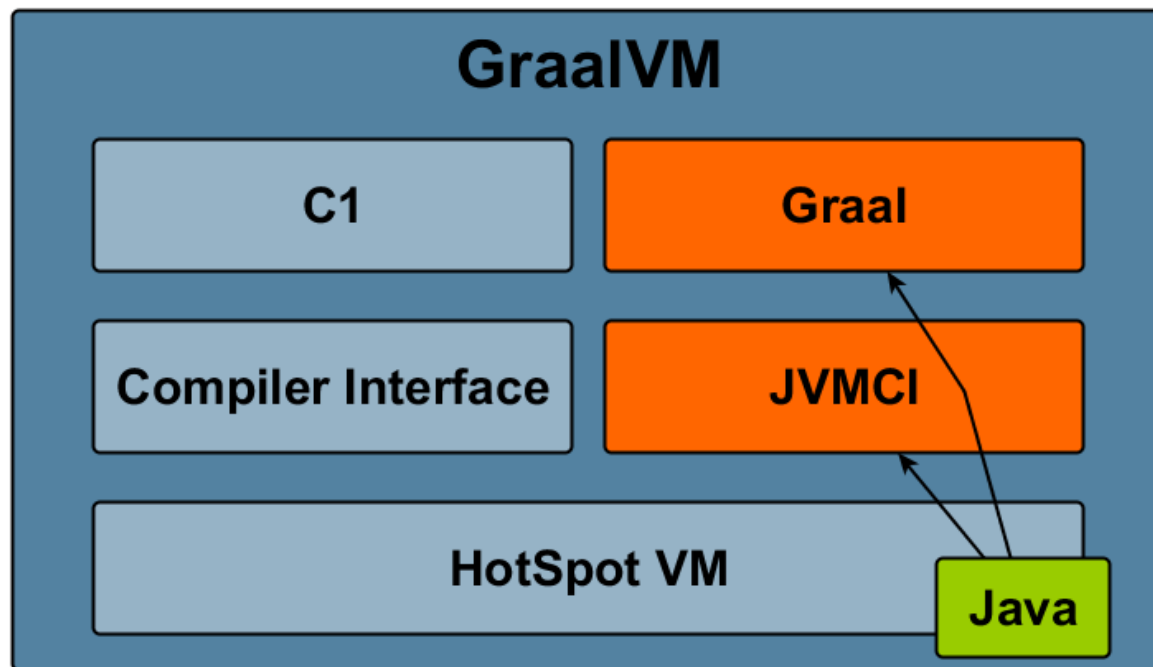
- **Inlining**
 - Code der aufzurufenden Methode/Funktion anstelle des Aufrufs
- **On-Stack Replacement**
 - Loop-Compilation, ohne auf den Methodenaufruf zu warten
- **Escape Analysis**
 - Automatische Stack-Allokation, ohne GC
- **De-Optimierung**
 - Optimierung rückgängig machen



JIT Compiler written in C++



JIT Compiler written in Java



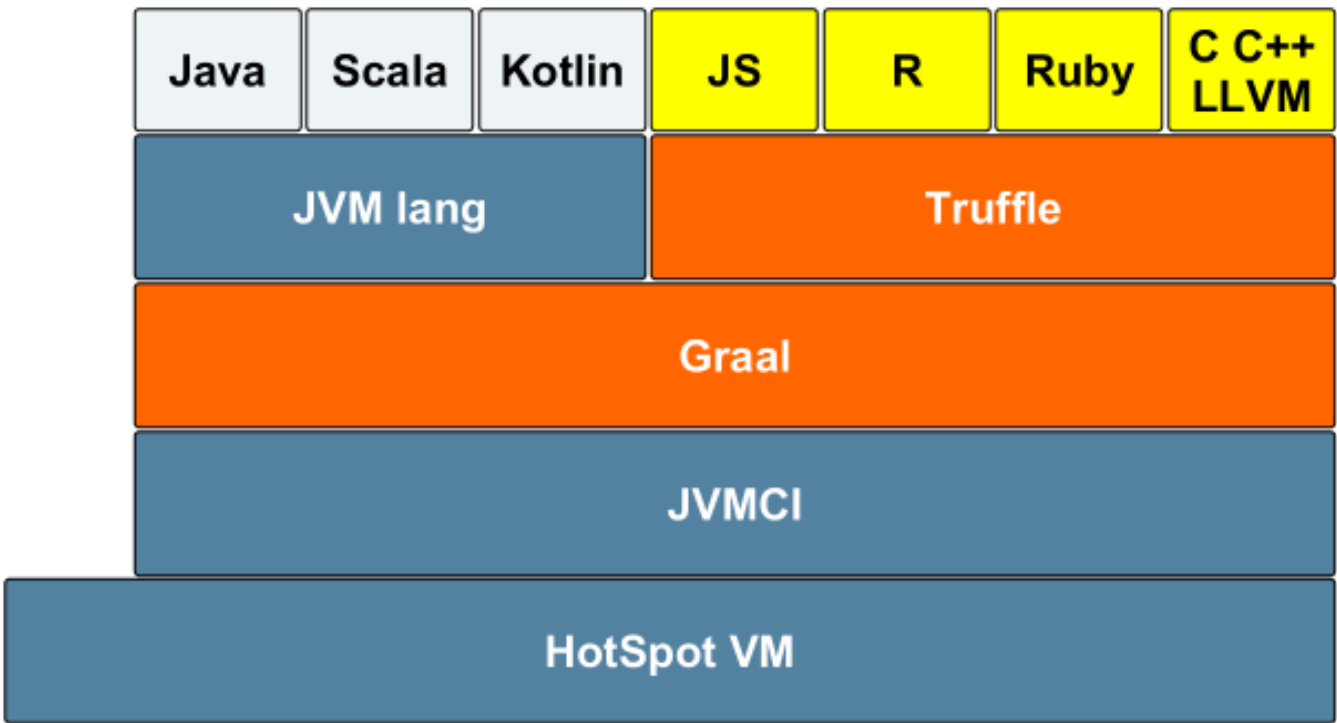
GraalVM



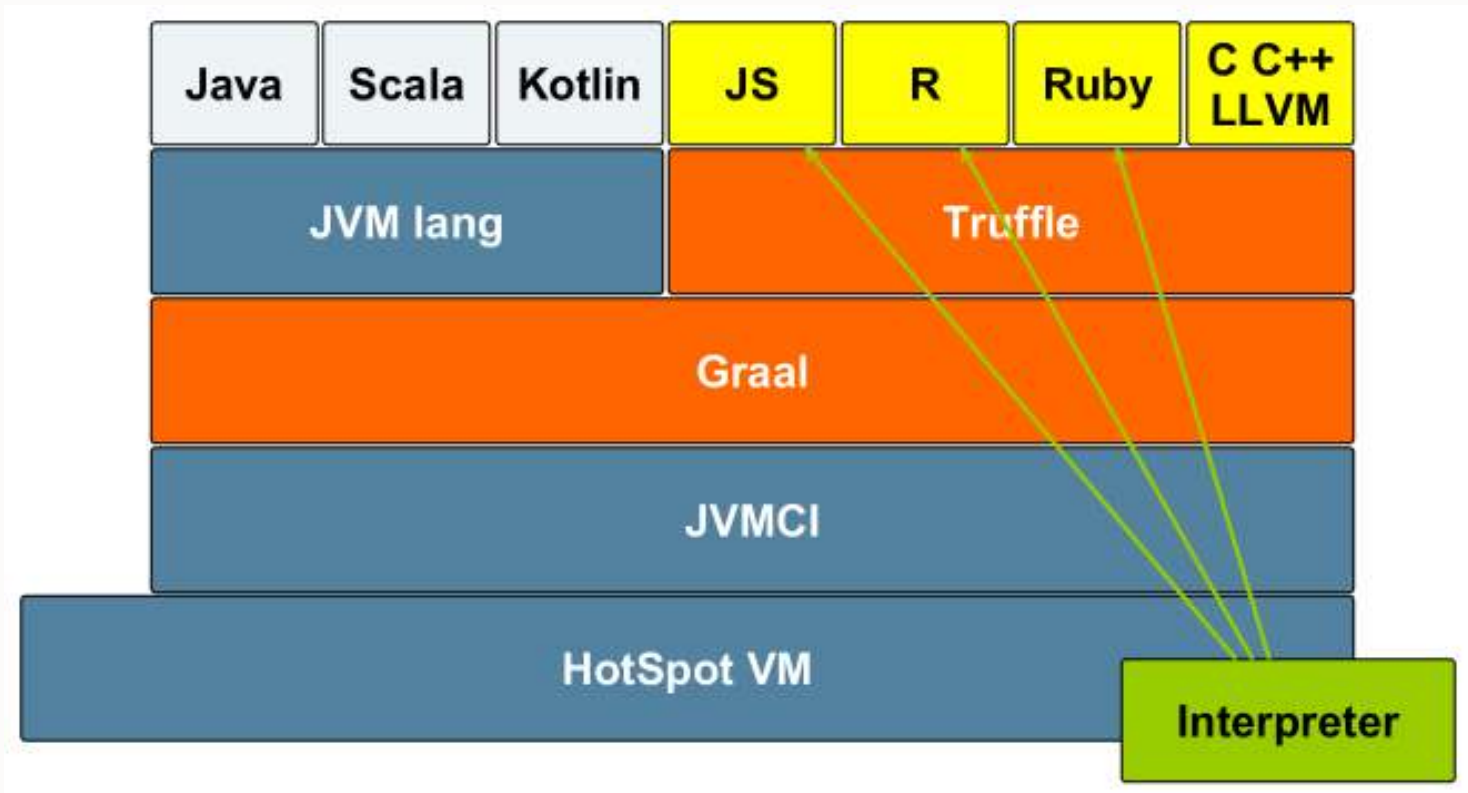
- **Graal**
 - **JIT Compiler**
 - Graal in GraalVM - A new Java JIT Compiler
 - **Graal integrated via Java Virtual Machine Compiler Interface (JVM CI)**
 - **Use a JDK with Graal (jdk.internal.vm.compiler)**
- **Truffle**
 - **Language Implementation Framework**
- **Substrate VM**
 - **Runtime Library and a set of tools for building Java AOT compiled code**



GraalVM - Polyglot (1)

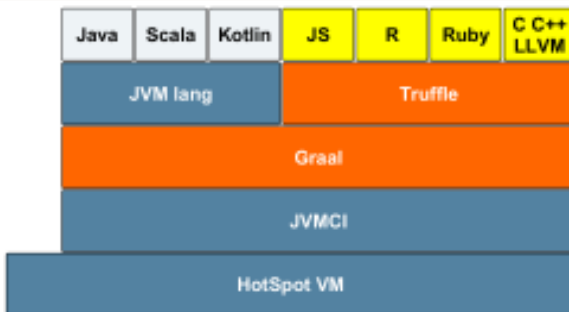


GraalVM - Polyglot (2)

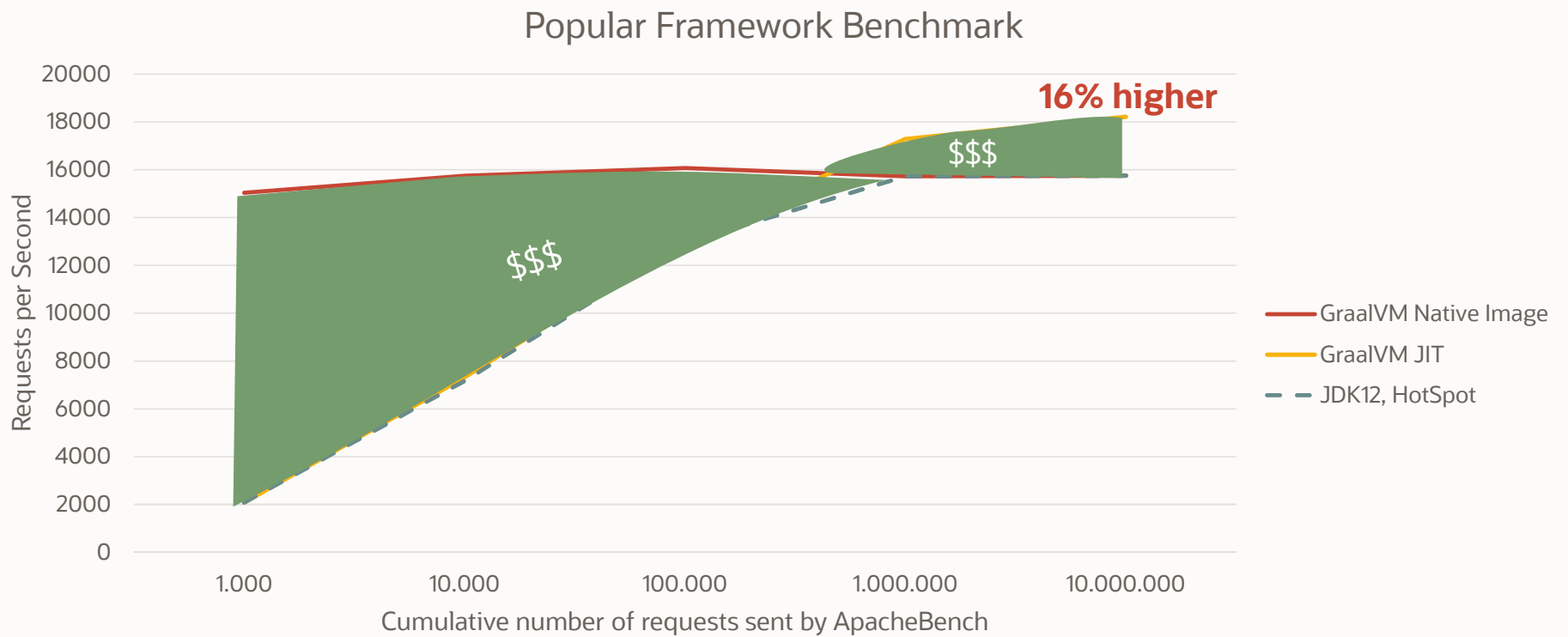


GraalVM - Language Usability

Production-Ready	Experimental	Visionary
Java	Ruby	Python
Scala, Groovy, Kotlin	R	VSCode Plugin
JavaScript	LLVM Tool Chain	GPU Integration
Node.js		WebAssembly
Native Image		LLVM Backend
VisualVM		

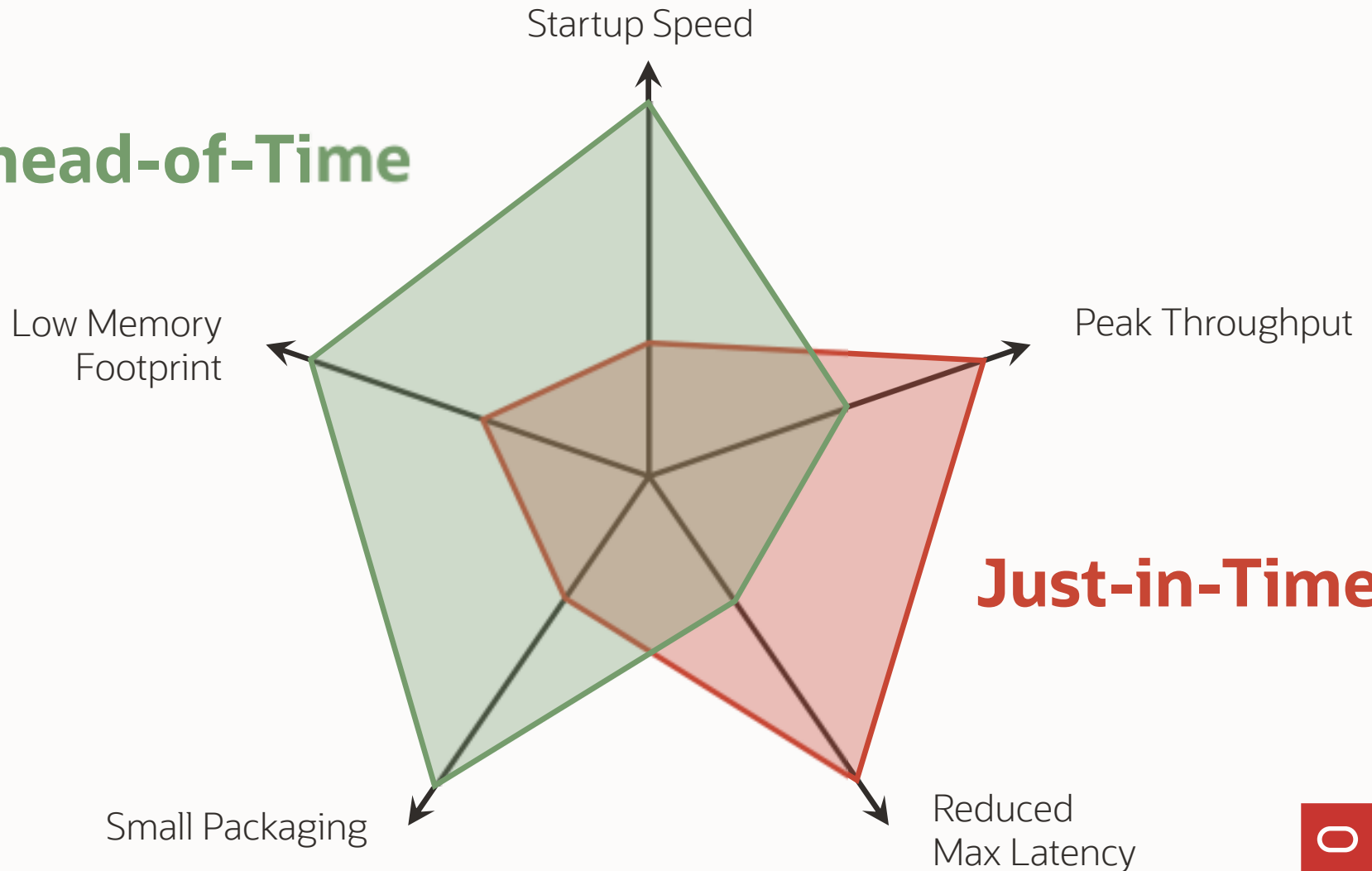


GraalVM Enterprise throughput



GraalVM Enterprise compilation performance characteristics

Ahead-of-Time



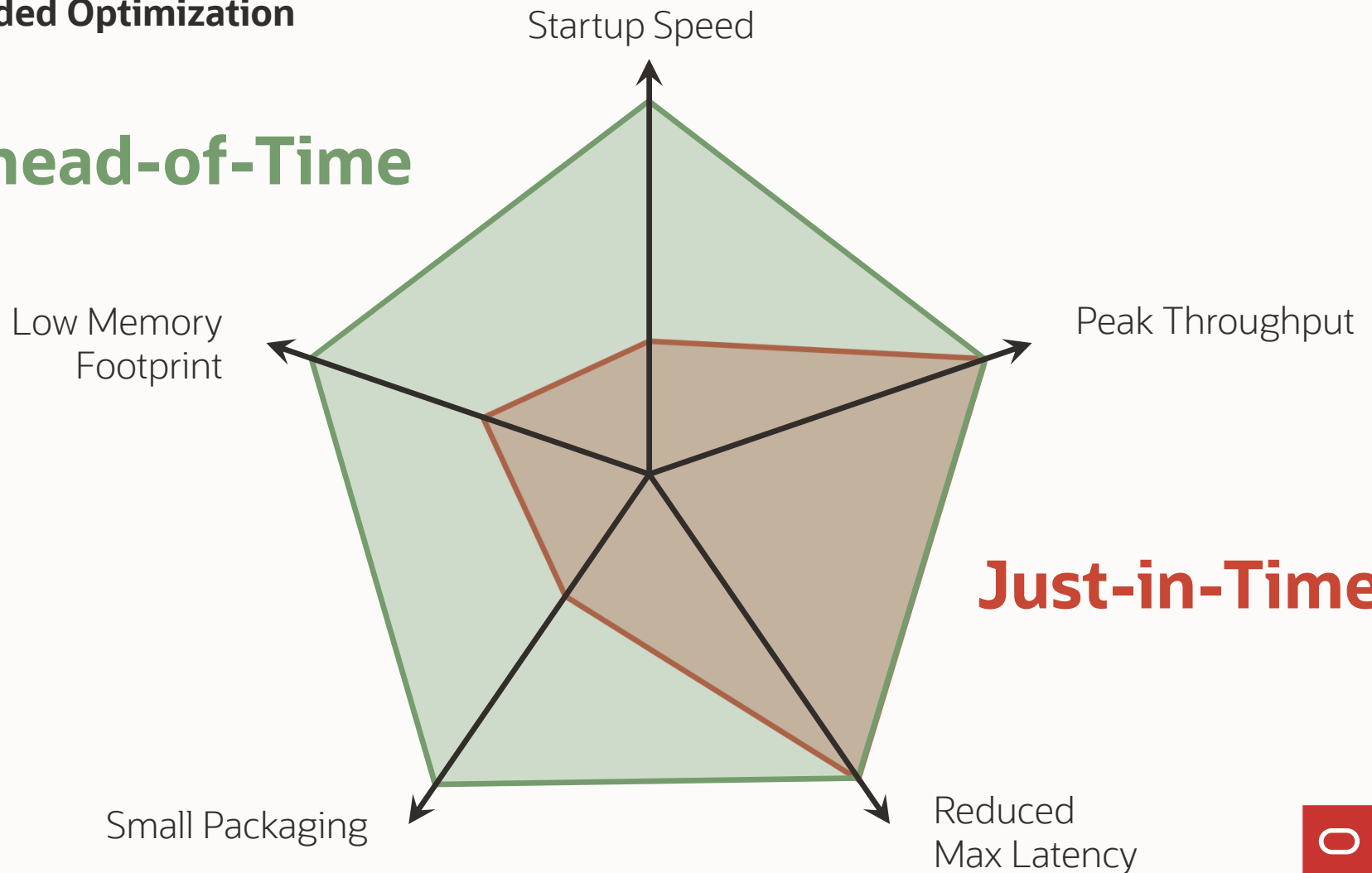
Just-in-Time



GraalVM Enterprise compilation performance characteristics

Profile Guided Optimization

Ahead-of-Time



Just-in-Time

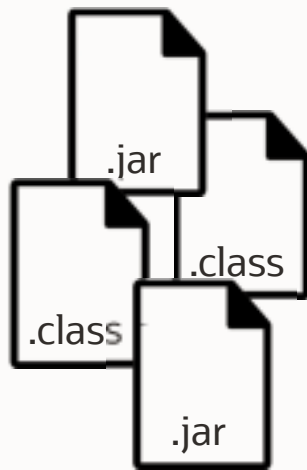


ORACLE

GraalVM Native Image

GraalVM Enterprise Native Image—Ahead-of-time compiler & runtime

Microservices and Containers



Up to 5x less memory
100x faster startup



Closed World Assumption

- **The points-to analysis needs to see all bytecode**
 - Otherwise aggressive AOT optimizations are not possible
 - Otherwise unused classes, methods, and fields cannot be removed
 - Otherwise a class loader / bytecode interpreter is necessary at run time
- **Dynamic parts of Java require configuration at build time**
 - Reflection, JNI, Proxy, resources, ...
- **No loading of new classes at run time**

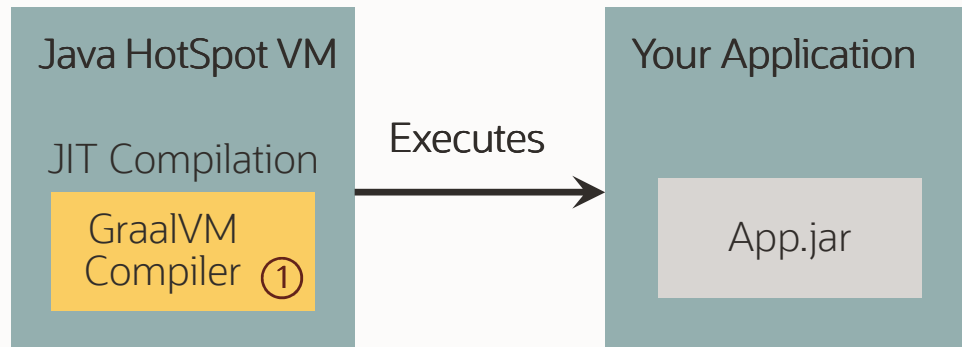


Image Heap

- **Execution at run time starts with an initial heap: the “image heap”**
 - Objects are allocated in the Java VM that runs the image generator
 - Heap snapshotting gathers all objects that are reachable at run time
- **Do things once at build time instead at every application startup**
 - Class initializers, initializers for static and static final fields
 - Explicit code that is part of a so-called “Feature”
- **Examples for objects in the image heap**
 - `java.lang.Class` objects, Enum constants



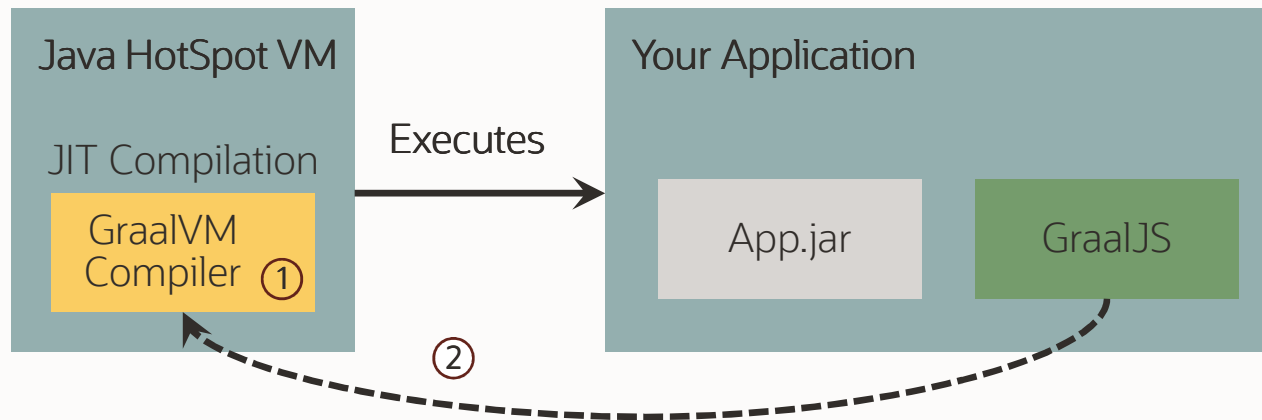
One Compiler, Many Configurations



① Compiler configured for just-in-time compilation inside the Java HotSpot VM

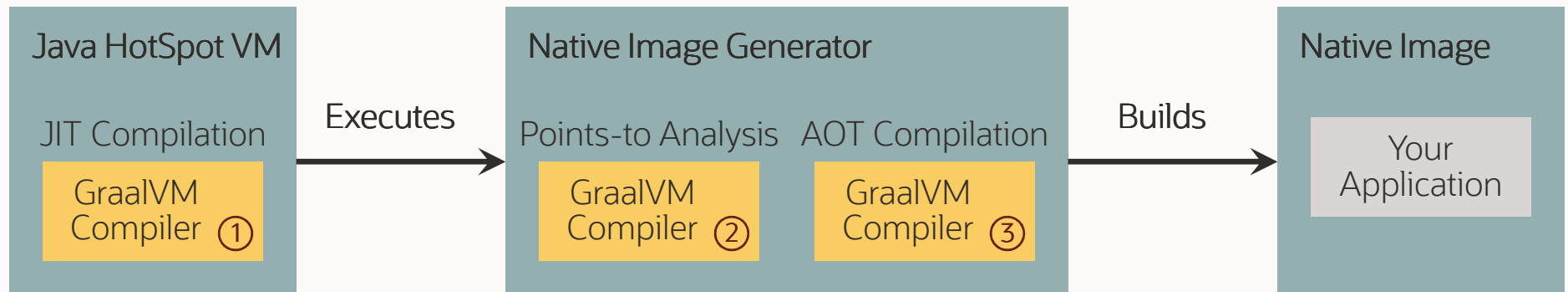


One Compiler, Many Configurations



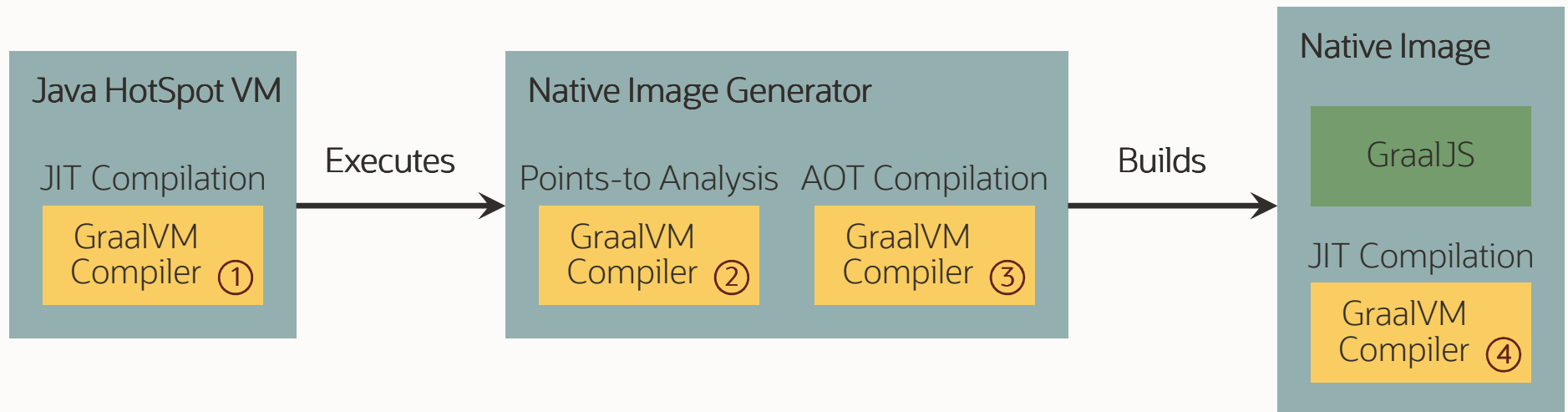
- ① Compiler configured for just-in-time compilation inside the Java HotSpot VM
- ② Compiler also used for just-in-time compilation of JavaScript code

One Compiler, Many Configurations



- ① Compiler configured for just-in-time compilation inside the Java HotSpot VM
- ② Compiler configured for static points-to analysis
- ③ Compiler configured for ahead-of-time compilation

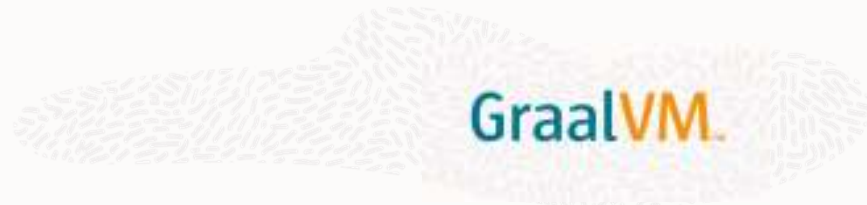
One Compiler, Many Configurations



- ① Compiler configured for just-in-time compilation inside the Java HotSpot VM
- ② Compiler configured for static points-to analysis
- ③ Compiler configured for ahead-of-time compilation
- ④ Compiler configured for just-in-time compilation inside a Native Image

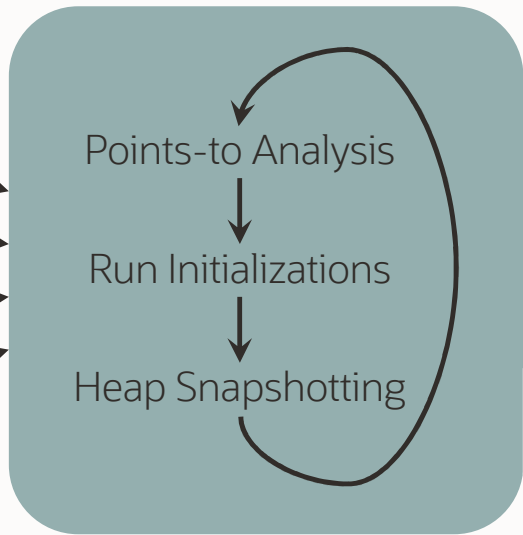


Native Image - Details



Input:
All classes from application,
libraries, and VM

- Application
- Libraries
- JDK
- Substrate VM



Iterative analysis until
fixed point is reached

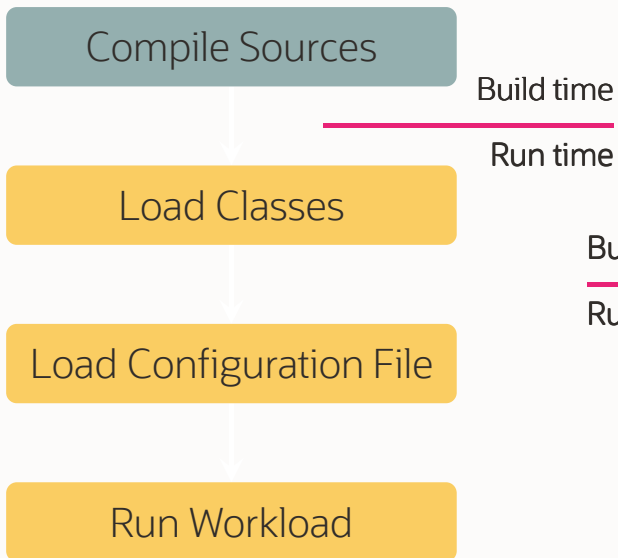
Output:
Native executable

- Code in Text Section
- Image Heap in Data Section

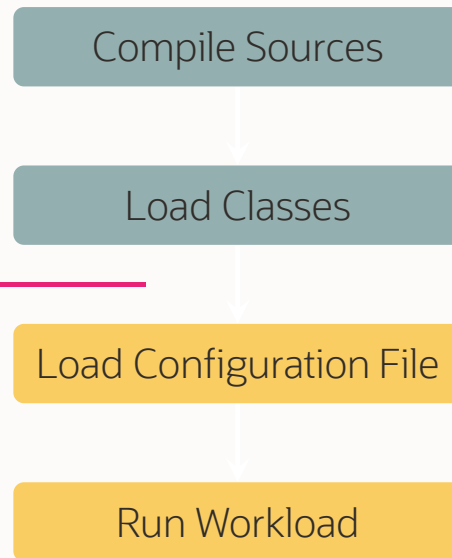


Benefits of the Image Heap

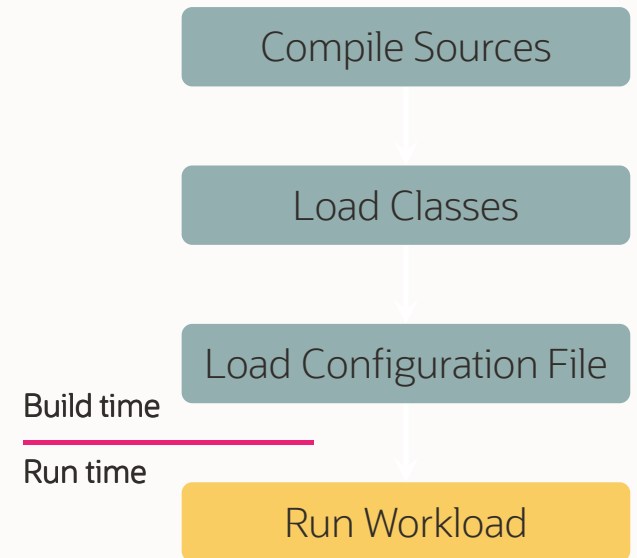
Without GraalVM Native Image



GraalVM Native Image (default)



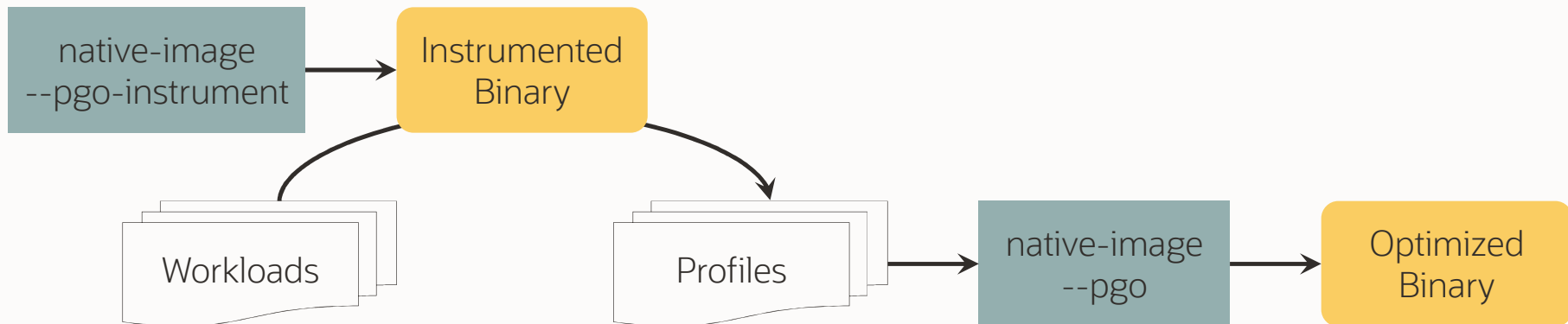
GraalVM Native Image: Load configuration file at build time



Profile-Guided Optimizations (PGO)

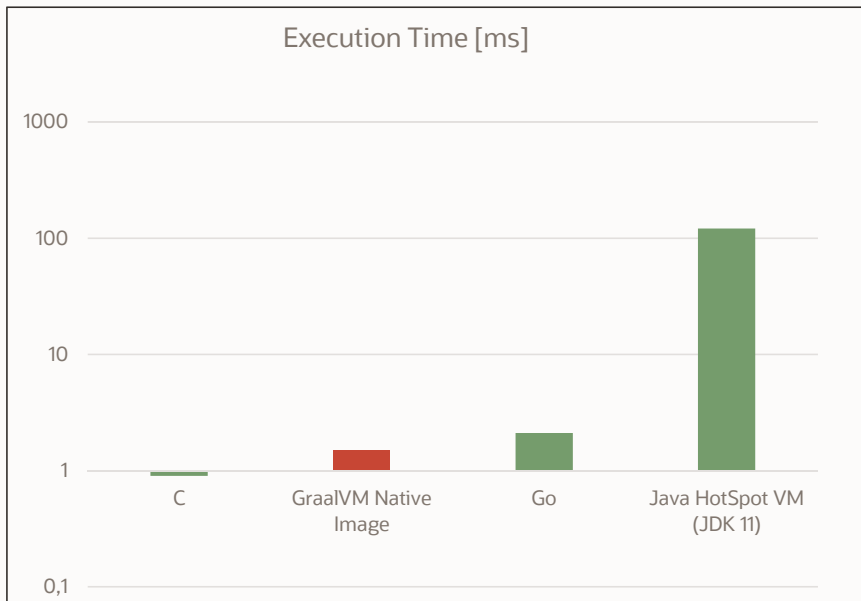
Out of Band Optimization

- AOT compiled code cannot optimize itself at run time (no “hot spot” compilation)
- PGO requires representative workloads
- Optimized code runs immediately at startup, no “warmup” curve

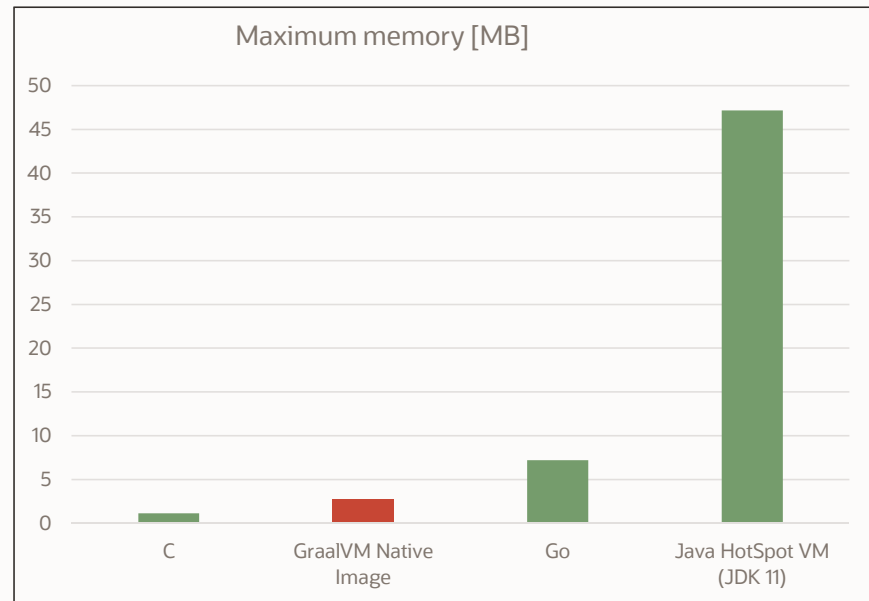


GraalVM Enterprise Native Image

Lower cloud costs for containerized workloads, and microservices



Competitive startup time



Significantly reduced memory requirements



GraalVM Enterprise Native Image

Supported by leading frameworks



GraalVM Enterprise Native Image - Spring Native

Which version of Spring Boot is certified or official supported with GraalVM EE 21 native image?

- We don't currently offer certification of Spring Boot but we are discussing it.
- But Spring will declare Spring Native 1.0, which is essentially “certification” for Native Image.

When do we expect the declaration of Spring Native 1.0, which is the essentially “certification” for the GraalVM native image?

- Spring Native is in beta now, as the latest 0.9.1
- Spring Native 0.9.0 supports Spring Boot 2.4.3,
- Spring Native 0.9.1 will support Spring Boot 2.4.4, etc.
- <https://spring.io/blog/2021/03/11/announcing-spring-native-beta>

Spring Native should be GA in the next few months/weeks.

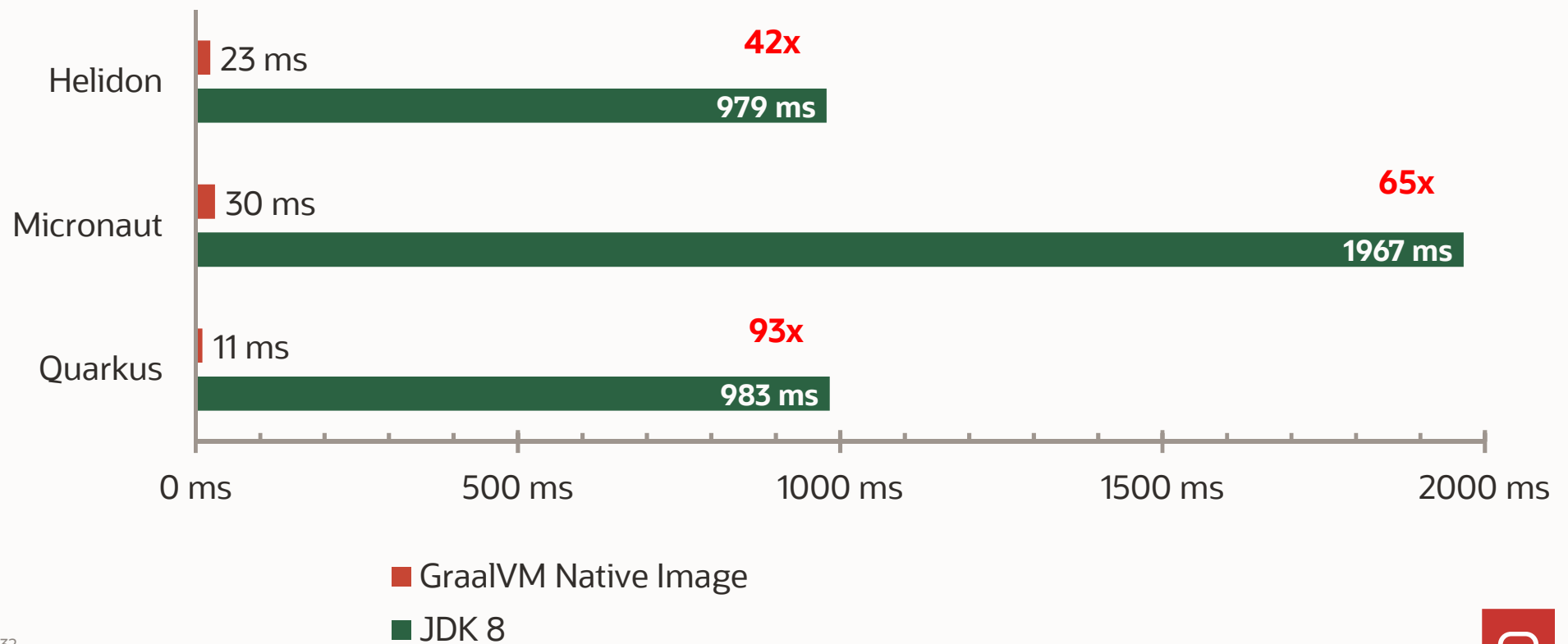


What GraalVM is for Microservices and Cloud Runtime

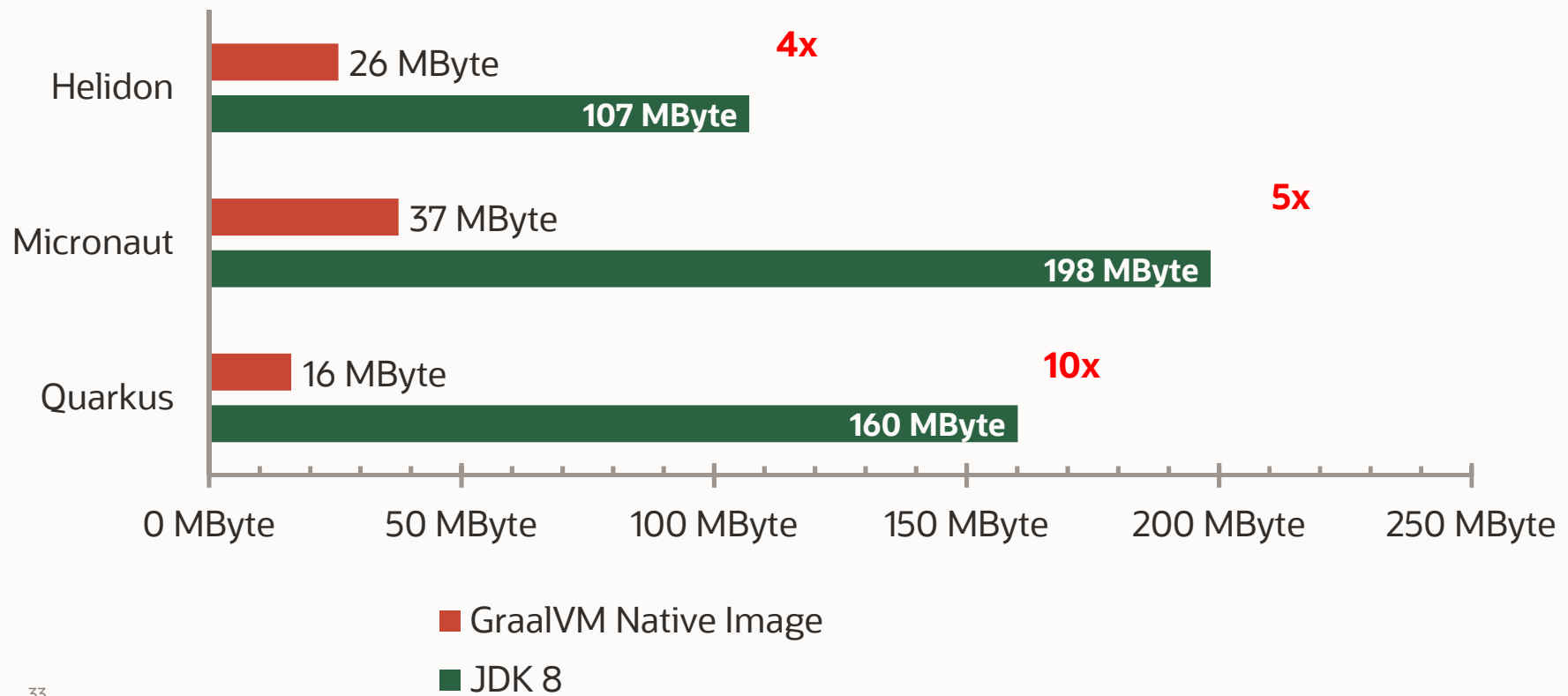
Up to 5x Less Memory
100x Faster Startup



Cloud Services – Startup Time



Cloud Services – Memory Footprint



GraalVM Enterprise Native Image based on JDK 11 with microservices frameworks

Build Profiles

1. Executable Jar

- Hollow jar
- All third-party dependencies are stored separately to take advantage of Docker layering

2. Jlink image

- Jlink optimized JRE + your application
- Faster startup time and smaller image size with no code restrictions

3. GraalVM native-image

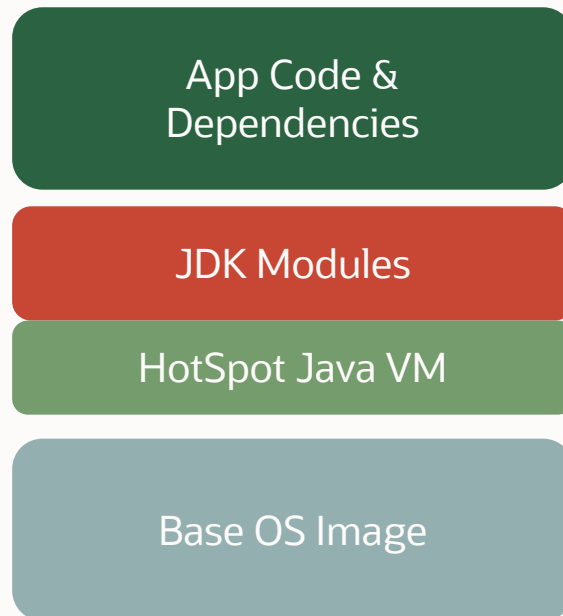
- Fastest startup time and smallest memory consumption
- Introduces some code restrictions related to usage of runtime operations



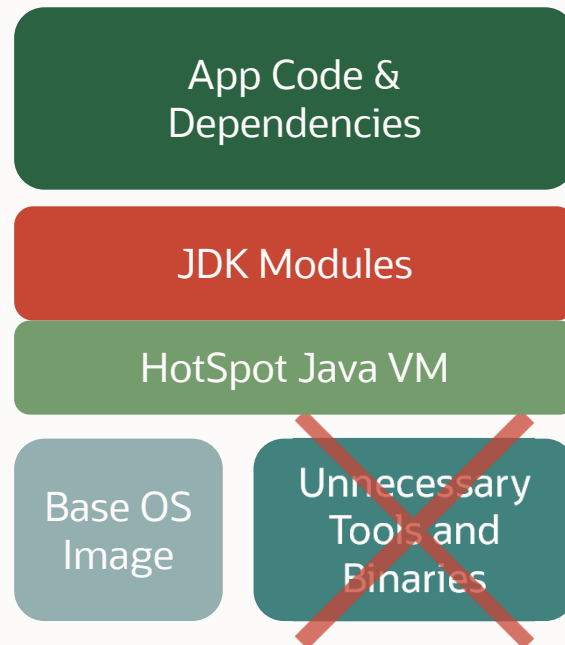
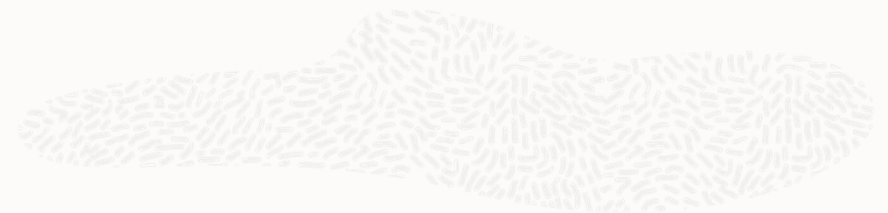
ORACLE

Java in Containers

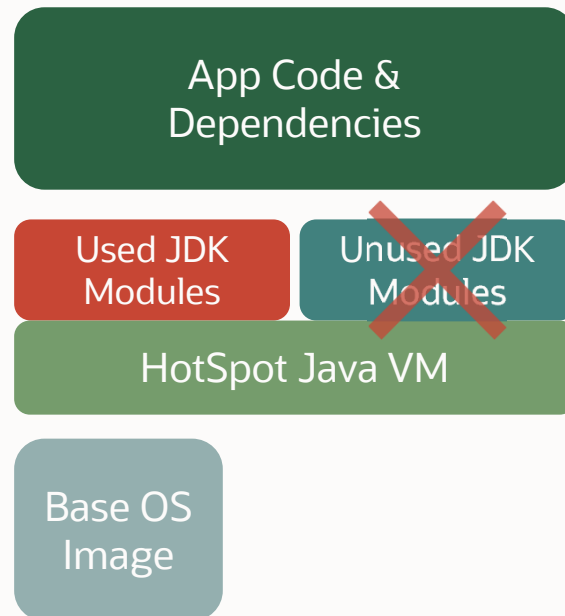
Java in Container



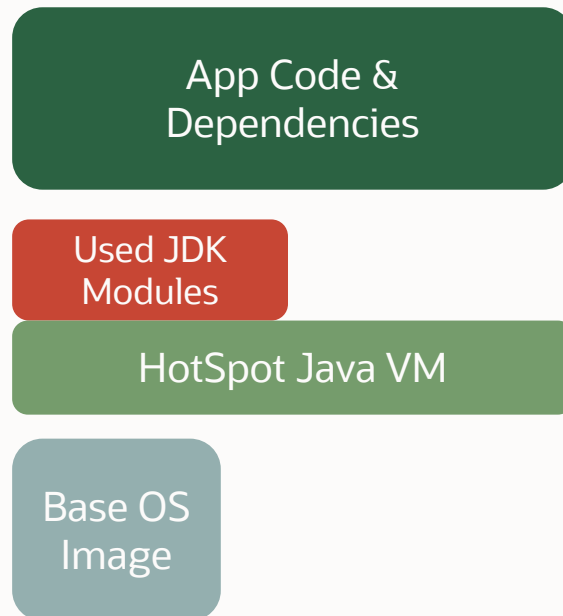
Java in a Slim/Distroless Container



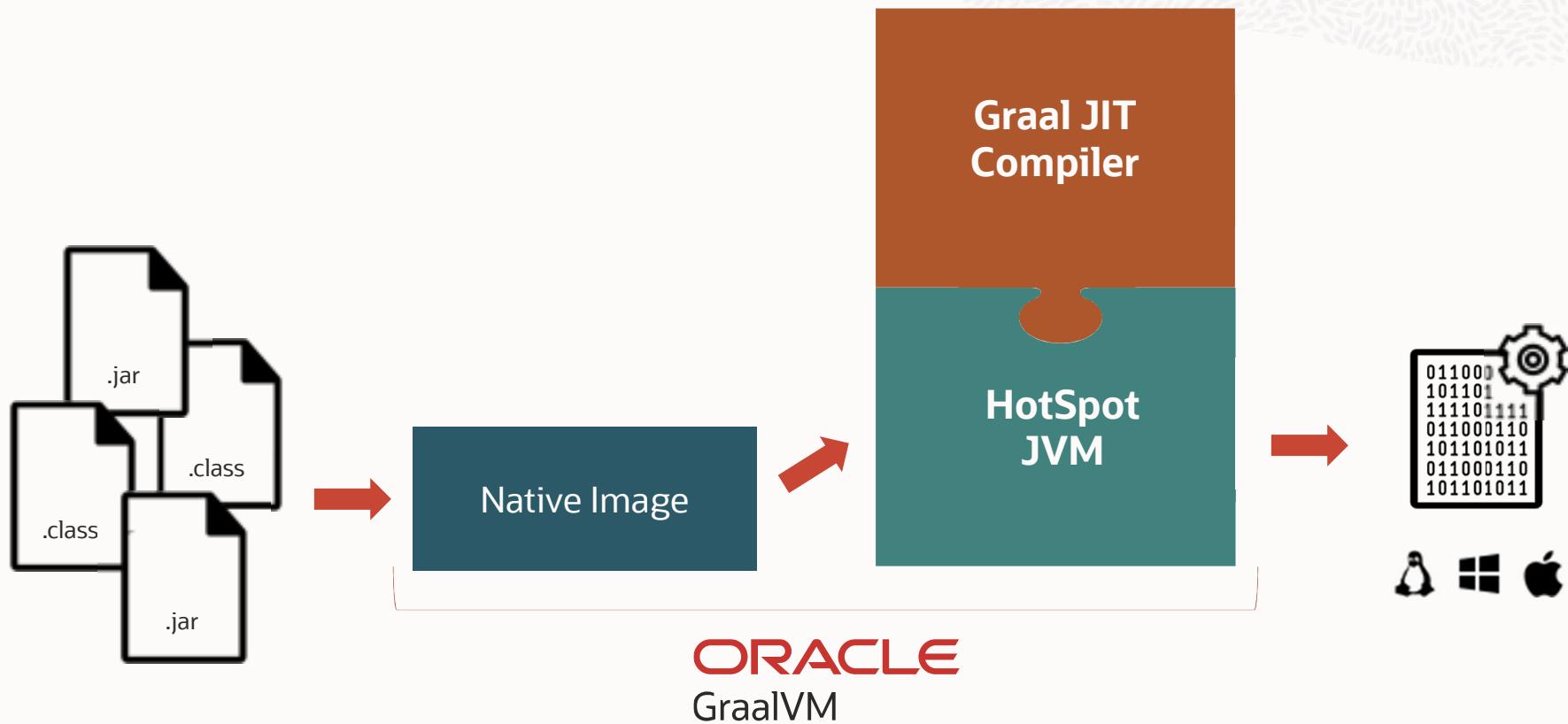
Java using `jlink` in a Slim/Distroless Container



Java using `jlink` in a Slim/Distroless Container



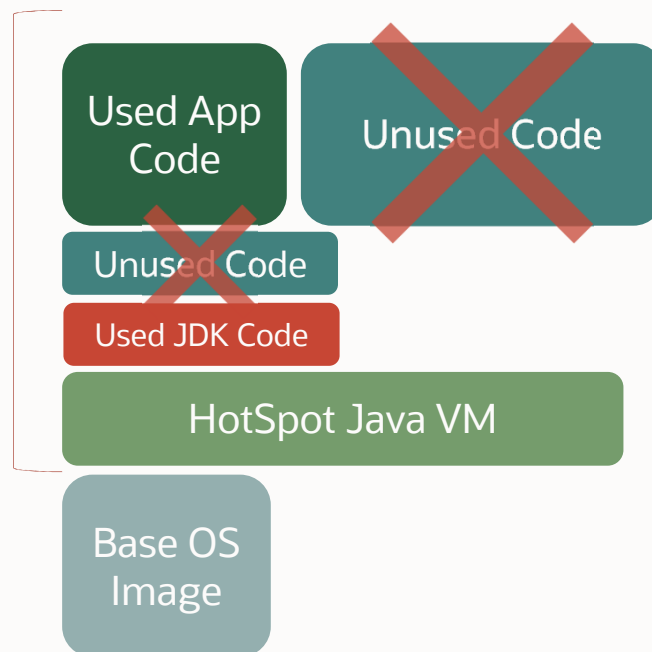
GraalVM Native Image — built on optimizing Graal compiler technology



Java Native Executable in Scratch Container w/ GraalVM Native Image

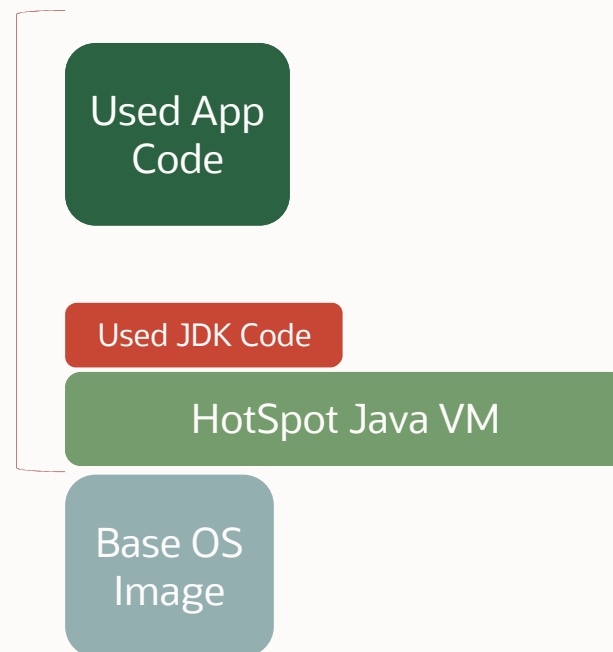
```
native-image --static -jar <jar> <app name>
```

ORACLE
GraalVM



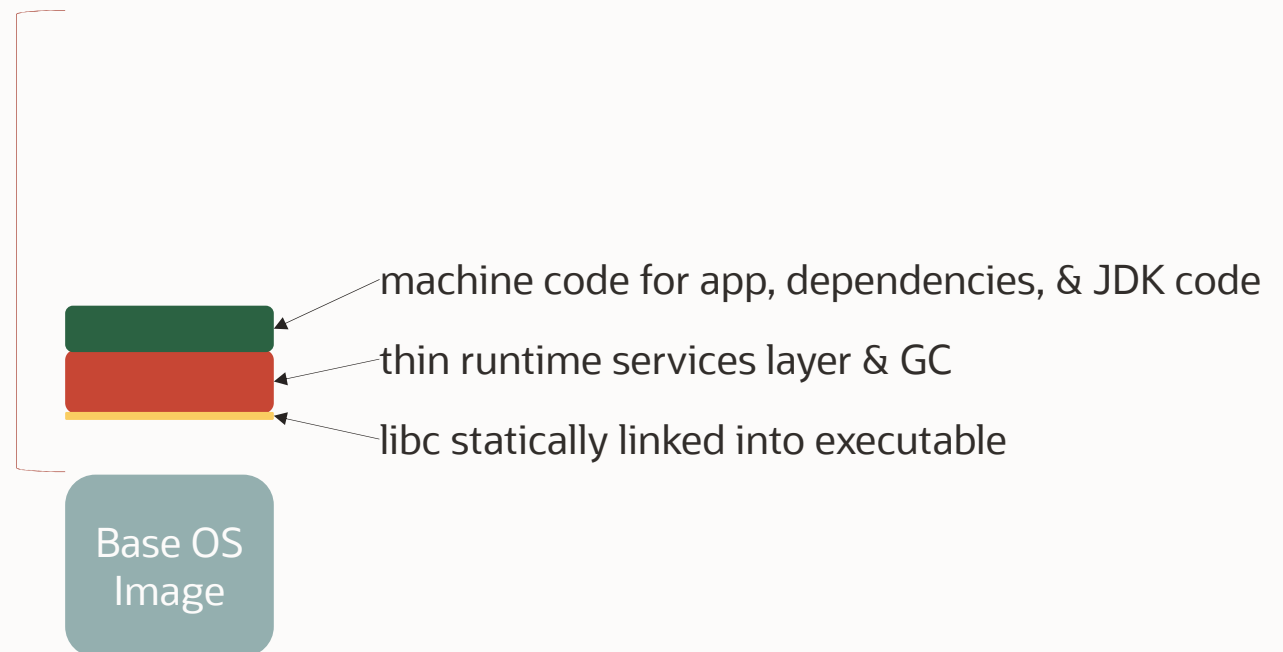
Java Native Executable in Scratch Container w/ GraalVM Native Image

ORACLE
GraalVM



Java Native Executable in Scratch Container w/ GraalVM Native Image

ORACLE
GraalVM



Java Native Executable in Scratch Container w/ GraalVM Native Image

```
FROM scratch  
COPY helloworld app  
ENTRYPOINT ["/app"]
```



Java Native Executable in Scratch Container w/ GraalVM Native Image

Compile, generate executable, build Container

```
$ javac HelloWorld.java
```

```
$ native-image --static HelloWorld hello
-rwxrwxr-x. 1 opc opc 11M Jan 26 18:54 hello
```

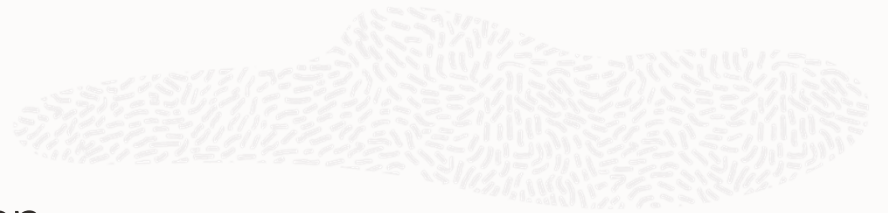
```
$ docker build . -t hello:scratch
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
hello	static	7afc946a849e	About a minute ago	10.7MB



Java Hello World container image size ~ **11 MB**

JVMs in Containers — recap



- JVM behaves as a good (Container) citizen
- Reduce “latency”
 - Container Startup
 - Application Startup
- **All OpenJDK investments “leaks” into containers!**
 - New Java languages
 - New JDK Features
 - Performance improvements
 - Footprint improvements
 - Etc.
- GraalVM offers unparalleled startup and container size reductions

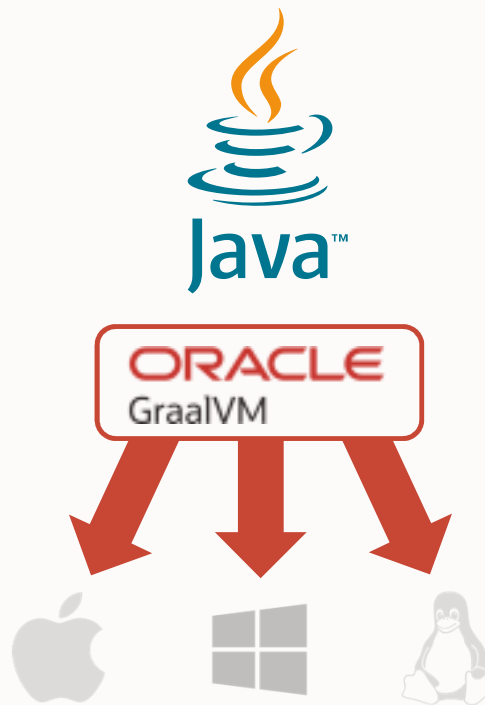


GraalVM Enterprise — Summary

High-performance optimizing
Just-in-Time (JIT) compiler

Ahead-of-Time (AOT)
“native image” compiler

Multilingual Virtual Machine



- **Test your applications with GraalVM**
 - *Documentation and downloads*
- **Connect your technology with GraalVM**
 - *Integrate GraalVM into your application*

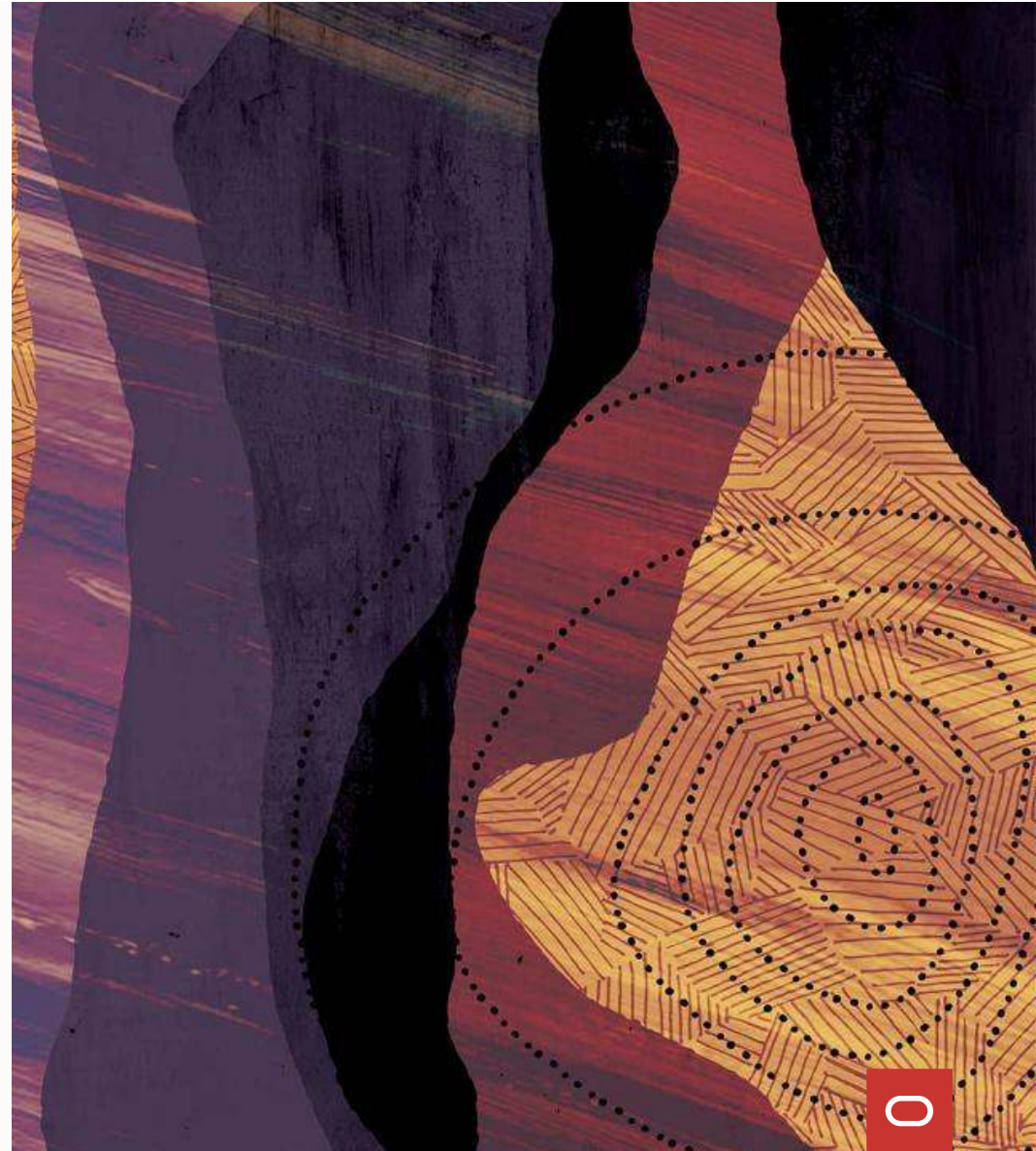


Thanks!

GraalVM Enterprise

—
Wolfgang.Weigend@oracle.com

Twitter: [@wolflook](https://twitter.com/wolflook)





ORACLE

GraalVM 21

Native Image Quick Reference



GraalVM Native Image Quick Reference

BUILD A NATIVE IMAGE

Build a native image from a JAR file with all dependencies:

```
native-image [options] -jar myApp.jar [imagenam]
```

Specify classes search path for directories, JARs, ZIPs:

```
-cp jar:com/package/**/myApp.jar
```

Specify the custom main class:

```
-H:Class=MyApp
```

Control classes initialization at build or run time:

```
--initialize-at-build/run-time= com.example.MyClass,org.package
```

Include resources matching Java RegEx:

```
-H:IncludeResource=/com/package/**/file.xml
```

Enable HTTPS support:

```
--enable-https
```

Install exit handlers:

```
--install-exit-handlers
```

Include all security service classes:

```
--enable-all-security-services
```

Add all charsets support:

```
-H:+AddAllCharsets
```

Include all timezones pre-initialized:

```
-H:+IncludeAllTimeZones
```

Build a statically linked image with libc implementation:

```
--static --libc=glibc/musl
```

Build a statically linked image with libc dynamically linked (distroless):

```
-H:+StaticExecutableWithDynamicLibC
```

Enable polyjot support:

```
--language:java|js|python|ruby|llvm|wasm
```

Attach a debugger to the build process:

```
--debug-attach=[port]
```

BUILD AN OPTIMIZED NATIVE IMAGE

Build an image with profile-guided optimizations:

```
native-image --pgo-instrument MyApp
```

Run the image to record the profile: ./myapp

```
native-image --pgo default.iprof MyApp
```

Select GraalVM's garbage collector implementation:

```
--gc=G1
```

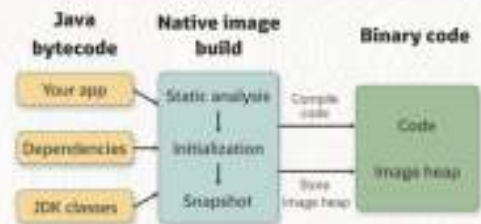
BUILD A SHARED LIBRARY

```
native-image -jar jarfile [libraryname]
```

```
--shared
```

Annotate the entry point method(s) with the `@EntryPoint` annotation. Entry point methods must be static, have non-object parameters and return types.

```
@EntryPoint  
static int add(InitiateThread thread, int a, int b) {  
    return a + b;  
}
```



More info at www.graalvm.org

CONFIGURE A NATIVE IMAGE

Static analysis requires configuration for some language features: accessing resources, serialization, reflection, JNI, etc.

Run a Java process with tracing agent to generate the configuration:

```
java -agentlib:native-image-agent= config-output-dir=/path/to/config-dir/ -jar MyApp.jar
```

Specify the configuration to use for building a native image:

```
-H:ConfigurationFileDirectories=/path/to/config-dir/
```

Configuration files in META-INF/native-image on the classpath are included automatically.

Configure memory at run time:

```
./imagenam -Xmx<n> -Xmn<n>
```

Configure default heap settings at build time:

```
-R:MaxHeapSize=<n> -R:MaxNewSize=<n>
```

DEBUG A NATIVE IMAGE

Build a native image with debug information:

```
-g
```

Print garbage collection logs:

```
./imagenam -XX:+PrintGC -XX:+VerboseGC
```

Trace the initialization path for a certain class:

```
-H:+TraceClassInitialization= package.class.Name
```

Print classes initialized detected by the static analysis:

```
-H:+PrintClassInitialization
```

Gather the diagnostic data for GraalVM Dashboard:

```
-H:+DashboardAll -H:DashboardDump=<path>
```

© 2020



ORACLE

GraalVM 21.1.0

Release Notes



GraalVM Enterprise Edition 21.1.0 Release Notes (1)

- **Java**

The Oracle JDK release that GraalVM Enterprise Edition is built on was updated to:

- 8u291 for Java 8 based GraalVM Enterprise, please see Java SE 8 release notes
- 11.0.11 for Java 11 based GraalVM Enterprise, please see Java SE 11 release notes
- 16.0.1 for Java 16 based GraalVM Enterprise, please see Java SE 16 release notes

- **Platform Updates**

- **Java 16 (experimental) support:**
 - The GraalVM Enterprise distribution based on Oracle Java 16 is available for download with several known limitations.
- **MacOS platform support:**
 - GraalVM Enterprise builds for macOS based on Oracle JDK 8 continue to be available, unlike builds of GraalVM Community for macOS based on OpenJDK 8 which are no longer being produced.
- **Linux AArch64 platform compatibility:**
 - The GraalVM Enterprise distributions for Linux AArch64 architecture remain experimental in this release. Supported features include the GraalVM compiler, the gu tool, the Node.js JavaScript runtime, Native Image, some developer tools.



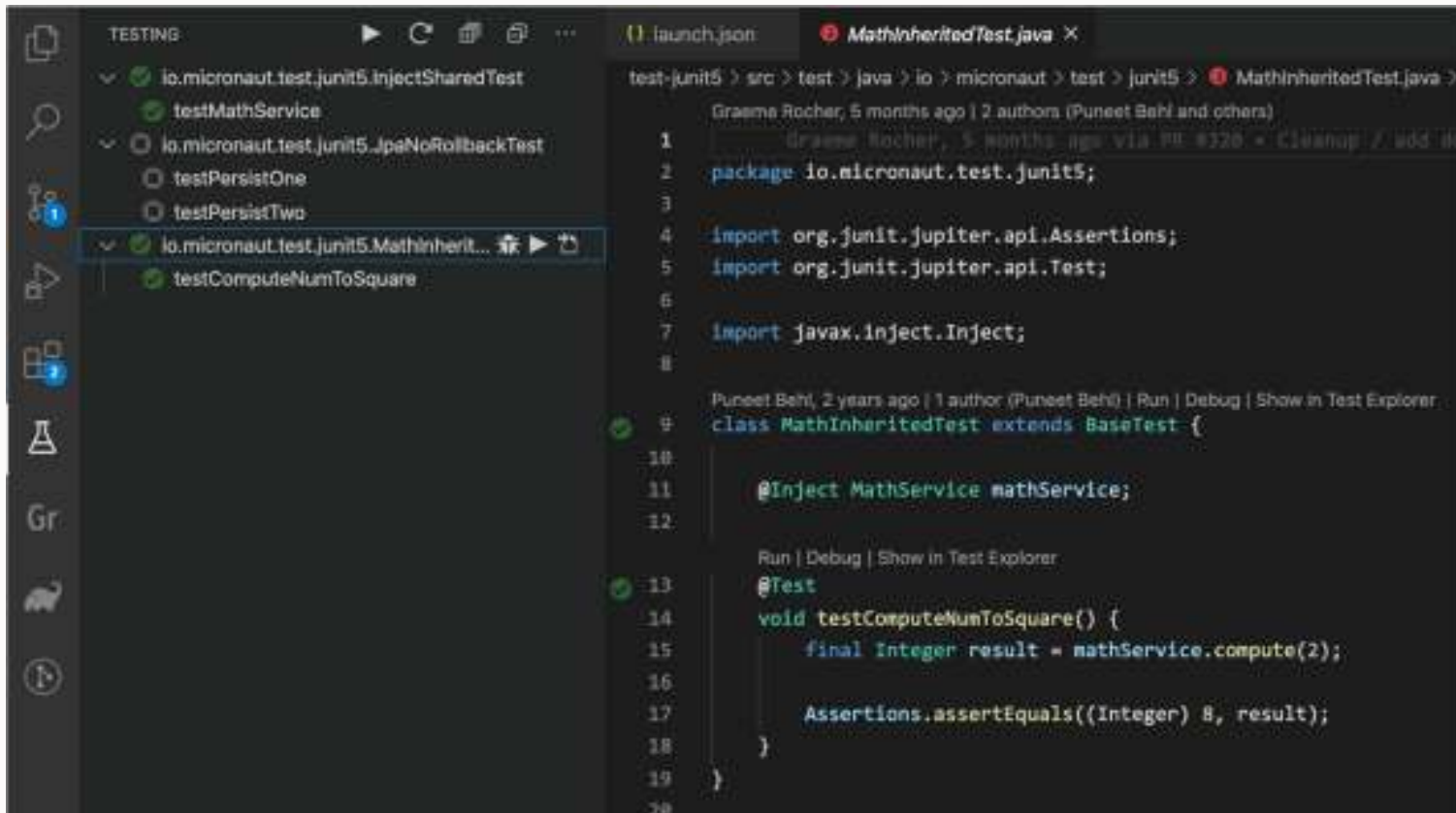
GraalVM Enterprise Edition 21.1.0 Release Notes (2)

- **Compiler**
- **Native Image**
- **Truffle**
- **Java on Truffle**
- **JavaScript**
- **LLVM Runtime (Sulong)**
- **Ruby**
- **Python**
- **R**
- **WebAssembly (GraalWasm)**
- **Truffle Language Contexts**
- **Truffle Language and Tool Implementations**



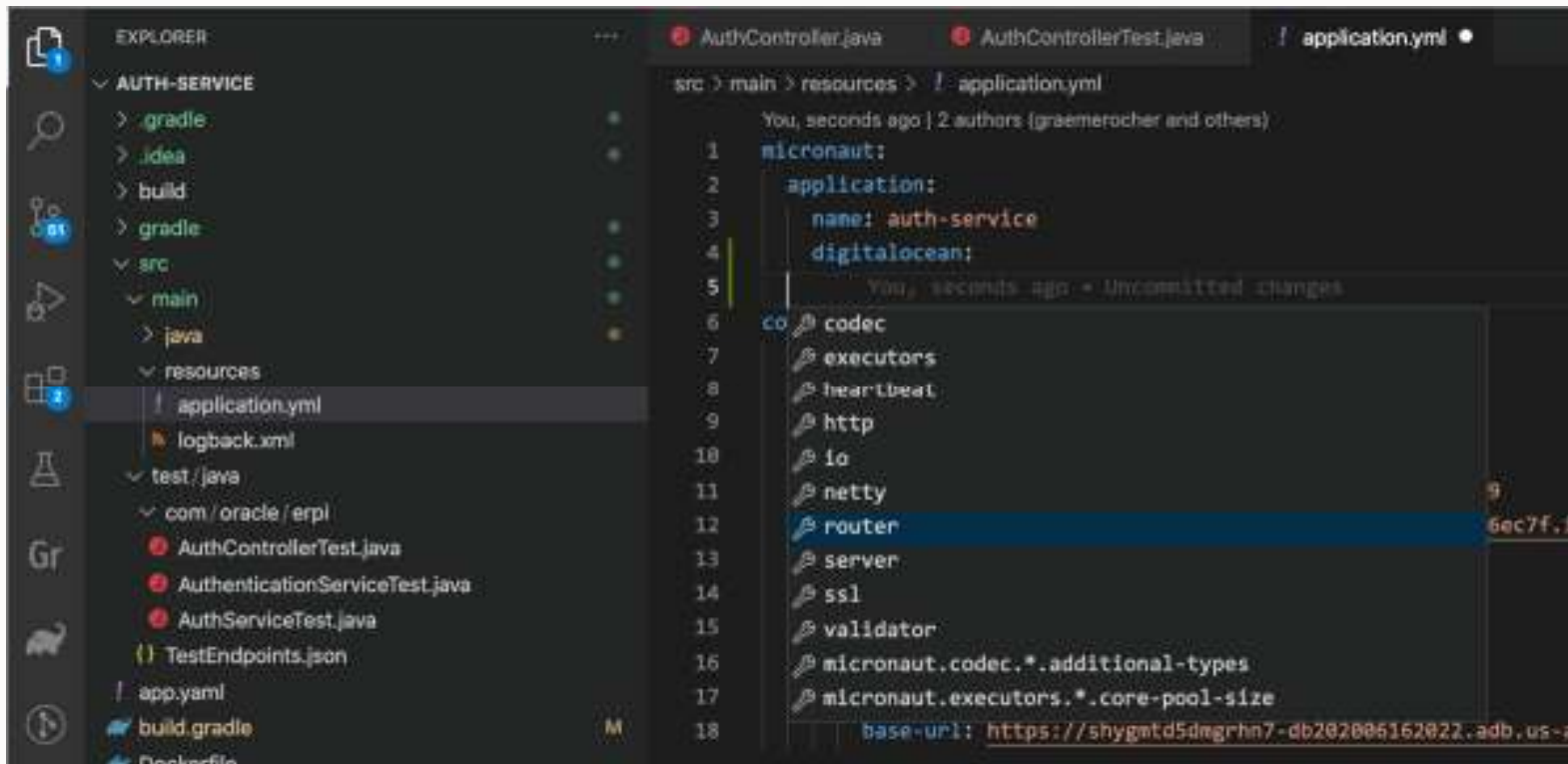
GraalVM Enterprise Edition 21.1.0 Release Notes (3) - Tools

- Visual Studio Code Extensions - Added results visualization for unit tests:



GraalVM Enterprise Edition 21.1.0 Release Notes (4) - Tools

- Visual Studio Code Extensions - Improved Micronaut support by adding YAML <=> Java code editing features:



GraalVM Enterprise Edition 21.1.0 Release Notes (5) - Tools

- **Visual Studio Code Extensions - Summary**

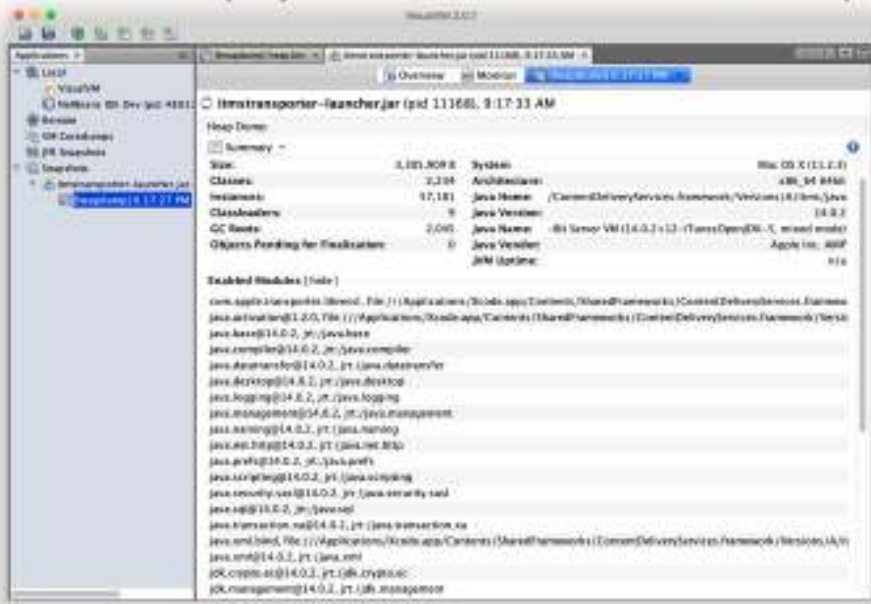
- Added results visualization for unit tests
- Improved Micronaut support by adding YAML <> Java code editing features
- Added a number of refactorings.
- Included Micronaut projects Docker build commands.
- Improved support for Maven and Gradle multi-project builds.



GraalVM Enterprise Edition 21.1.0 Release Notes (6) - Tools

- **VisualVM**

- Added support for upcoming JDK 16
- Added support for the new Apple M1 processor (aarch64)
- Added support for the importation of plugins from the previous release
- Added a display list of enabled modules in heap dumps feature, taken from JDK 9+ created by jlink



ORACLE

GraalVM 21.0.0

Espresso



GraalVM Enterprise Edition 21.0.0 – Espresso (1)

- A meta-circular Java bytecode interpreter for the GraalVM
- Espresso is a fully meta-circular implementation of *The Java Virtual Machine Specification, Java SE 8 and 11 Edition*, written in Java, capable of running non-trivial programs at speed
- A Java bytecode interpreter at its core, turned Just-In-Time (JIT) compiler by leveraging Truffle and the Graal compiler on the GraalVM
- It highlights the sublime potential of the GraalVM as a platform for implementing high-performance languages and runtimes

GraalVM Enterprise Edition 21.0.0 – Espresso (2) - Status

- Espresso is still an early prototype, but it already passes the Java Compatibility Kit (a.k.a. the JCK or TCK for Java SE) 8c and 11 runtime suite
- Espresso can compile itself with both javac and (the Eclipse Java Compiler) ecj
It features complete meta-circularity: it can run itself any amount of layers deep, preserving all the capabilities (Unsafe, JNI, Reflection...) of the base layer. Running HelloWorld on three nested layers of Espresso takes ~15 minutes
- Espresso is similar to HotSpot Express, the same codebase can run either an 8 or 11 guest JVM, on either an 8 or 11 host JVM
- The development of Espresso happens mostly on HotSpot, but this configuration (Espresso on HotSpot) is only supported on Linux
- Espresso's native image runs on Linux, MacOS and Windows



ORACLE

GraalVM Downloads



GraalVM Downloads

Community Edition

GraalVM Community is available for free for any use. It is built from the GraalVM sources available on GitHub. We provide pre-built binaries for Linux/X86, Linux/ARM, macOS, and Windows platforms on x86 64-bit systems. Support for the Windows and Linux/ARM platforms, and the Python, Ruby and R languages is experimental.

Note

GraalVM Community Edition contains significant technology from other projects including OpenJDK and Node.js which are not maintained by the GraalVM community. GraalVM Enterprise Edition is recommended for production applications.

Enterprise Edition

GraalVM Enterprise provides additional performance, security, and scalability relevant for running applications in production. **You can get a version of GraalVM Enterprise that is free for evaluation and developing new applications via the Oracle Technology Network (OTN), or a commercially licensed version for production use via the Oracle Store.**

We provide binaries for Linux, macOS, and Windows platforms on x86 64-bit systems. Support for the Windows and Linux/ARM platforms, and the Python, Ruby and R languages is experimental.



GraalVM – Downloads

GraalVM is distributed as **Community Edition** and **Enterprise Edition**. Listed below are bundles available:

- **GraalVM EE 21.1.0 based on Oracle Java 8u291**
- **GraalVM EE 21.1.0 based on Oracle Java 11.0.11**
- **GraalVM EE 21.1.0 based on Oracle Java 16.0.1**

- **OS: Linux, macOS, Windows**
- <https://www.graalvm.org/downloads/>



GraalVM Community 21.1.0	GraalVM Enterprise 21.1.0
Details →	Details →
<ul style="list-style-type: none">• Free for all purposes• Runs any program that runs on GraalVM Enterprise• Based on OpenJDK 8u292, 11.0.11 and 16.0.1	<ul style="list-style-type: none">• Free for evaluation and development• Additional performance, scalability and security• Based on Oracle JDK 8u291, 11.0.11 and 16.0.1• Included in Oracle Cloud and Java SE Subscription
DOWNLOAD FROM GITHUB	ORACLE GRAALVM DOWNLOADS
 macOS  Linux  Windows	 macOS  Linux  Windows

Oracle GraalVM Enterprise Edition 21

Release Version: Java Version: OS: Architecture:

GraalVM Enterprise is part of Oracle Java SE Subscription
GraalVM Enterprise 21 is the latest release containing new features.

Oracle GraalVM Enterprise Edition 21.1.0 Linux x86 for Java 16 Downloads

 Oracle GraalVM Enterprise Edition Core SHA256: ...1715f show copy The core components of Oracle GraalVM Enterprise Edition. Native Image and optional language packs are not included.	Status: Experimental (Not recommended for production use) Installation Guide
 Oracle GraalVM Enterprise Edition Native Image SHA256: ...f335a show copy GraalVM Enterprise Native Image is an ahead-of-time compiler.	Status: Experimental (Not recommended for production use) See know issues. Installation Guide







GraalVM – Downloads

Oracle GraalVM Enterprise Edition 21

Release Version: Java Version: OS: Architecture:

GraalVM Enterprise is part of Oracle Java SE Subscription
GraalVM Enterprise 21 is the latest release containing new features.

 Ideal Graph Visualizer SHA256: ...64725 show copy Ideal Graph Visualizer (IGV) allows GraalVM language developers to analyze compilation graphs. It is also useful for guest script developers if they need to optimize their scripts performance on top of GraalVM. Learn more about Ideal Graph Visualizer.	Installation Guide
 GraalVM LLVM Toolchain Plugin SHA256: ...9afd9 show copy GraalVM component containing the LLVM Toolchain 10.0.0.	Status: Experimental (Not recommended for production use) Installation Guide
 Oracle GraalVM Enterprise Edition WebAssembly Language Plugin SHA256: ...3a218 show copy GraalVM implementation of WebAssembly optimized for GraalVM Enterprise.	Status: Experimental (Not recommended for production use) Installation Guide
 Oracle GraalVM Enterprise Edition Node.js Runtime Plugin SHA256: ...41af0 show copy Node.js version using the GraalVM Enterprise Edition implementation of JavaScript.	Status: Experimental (Not recommended for production use) Installation Guide

