ESDE Workshop: GraalVM

Made by: Aleks Stamboliyski & Nikolay Kyosev

Contents

1. What is it?
2. Overview, Components & Compiler
3. Polyglot & Native Images
4. Performance & Security
5. Quiz

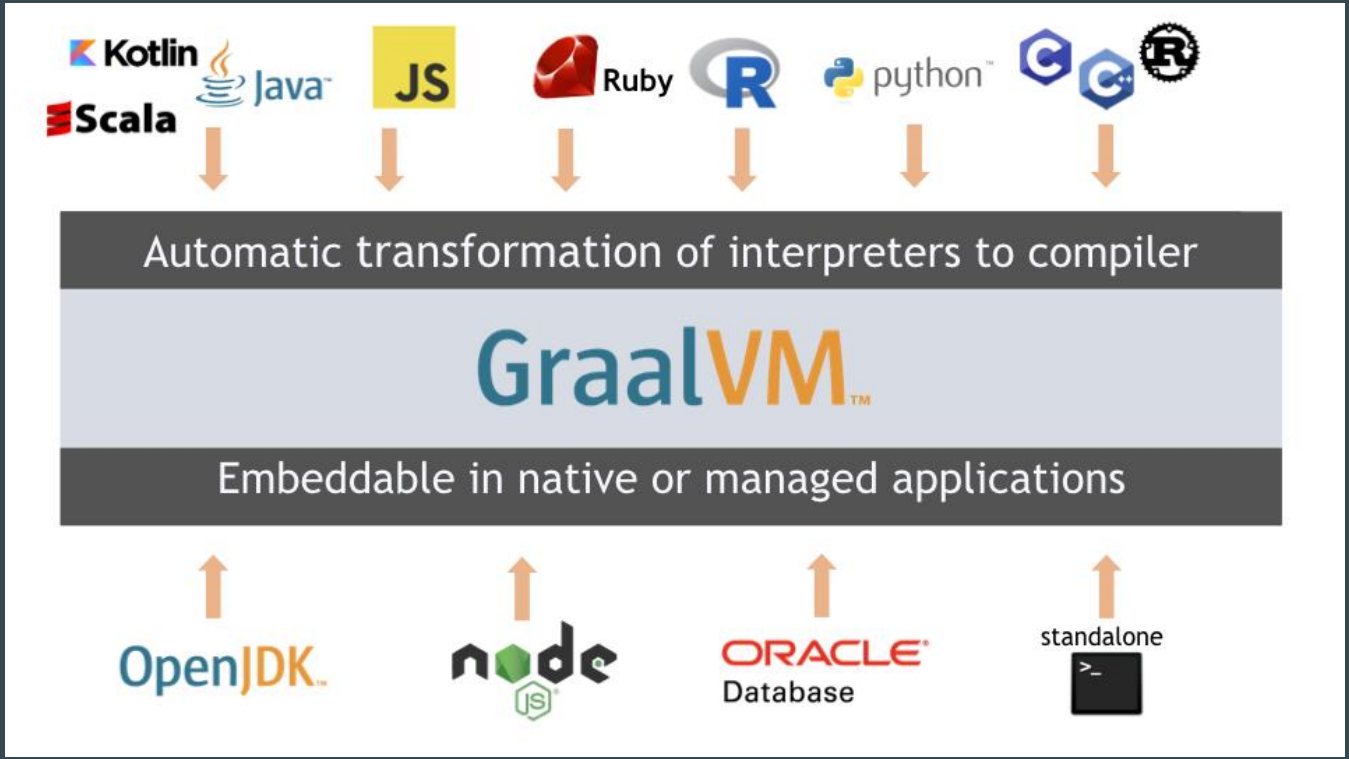
What is it?

- It’s a JVM
- Polyglot VM (supports multiple languages)
- Open source ecosystem
- Faster than the default JVM

Supported languages

- Java, Scala, Kotlin, etc (default)
- Java Script
- Python
- Ruby
- R
- LLVM (compiled C++, C, Rust)

Overview


Kotlin
Java
Scala
JS
Ruby
R
python
Automatic transformation of interpreters to compiler
GraalVM
Embeddable in native or managed applications
OpenJDK
node
ORACLE
Database
standalone

Core components

Runtimes

- Java HotSpot VM
- Node.js JavaScript runtime environment
- LLVM runtime

Libraries (jar files)

- GraalVM Compiler
- JavaScript interpreter
- LLVM bitcode interpreter
- GraalVM Polyglot API

Utilities

- JavaScript REPL
- LLVM bitcode interpreter command line utility
- GraalVM Updater

Additional components

- Native Image – a technology to compile a JVM-based application ahead-of-time into native code
- LLVM toolchain – a set of tools and APIs for compiling native programs to bitcode that can be executed with the GraalVM LLVM runtime
- Python interpreter – an implementation of the Python language
- Ruby interpreter – an implementation of the Ruby programming language
- R interpreter – an implementation of the R programming language
- GraalWasm – an implementation of the WebAssembly (Wasm) programming language

Compiler

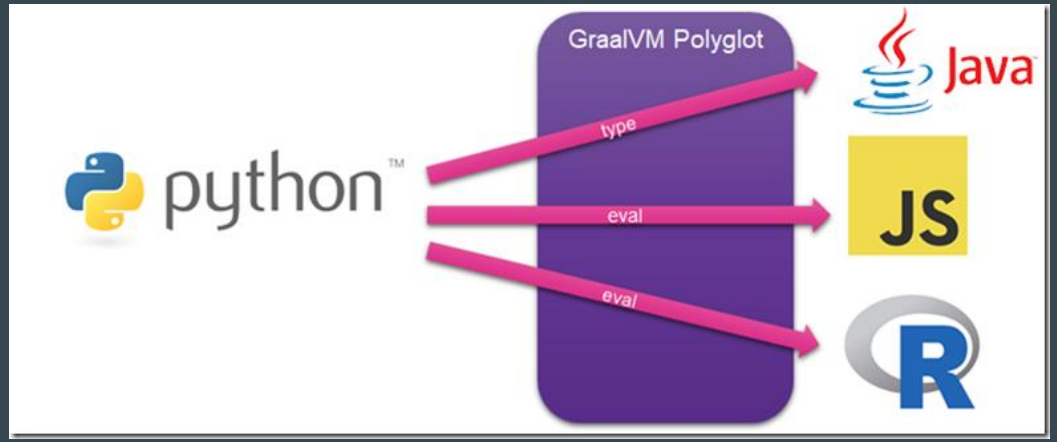
- Highly optimized
- Written in Java
- Integrates with the HotSpot VM
- Replaces the HotSpot JIT C1/C2 compiler

Compiler features

- Advantages - optimization algorithms
- Graph compilation
- AOT compilation
- Compiler operating modes - *libgraal* & *jargraal*
- Diagnostic data

Polyglot capabilities

- Language Implementation Framework
- Host and guest languages run on the same JVM and interchange data
- Polyglot interoperability protocol



Native images

- Compile Java bytecode into platform specific code
- Can build native executables or shared libraries (such as dll)
- Must have a JVM language at its core (Java, Scala, Kotlin, etc)

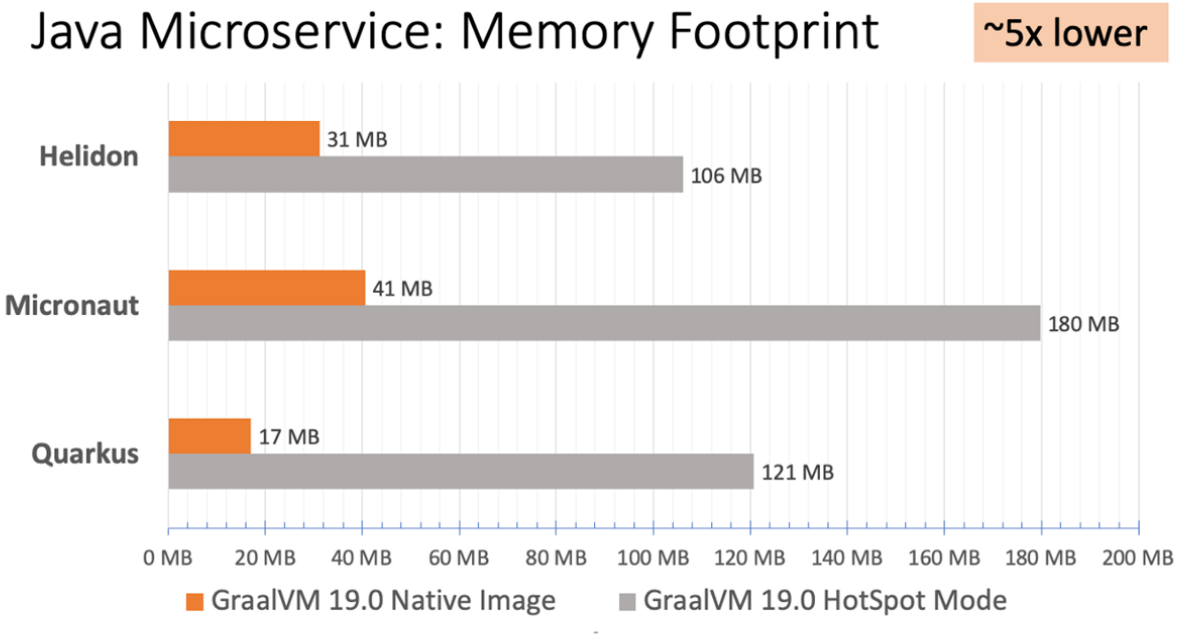
Pros

- Faster application startup time
- Smaller memory footprint
- Plug and run, does not need external dependencies.

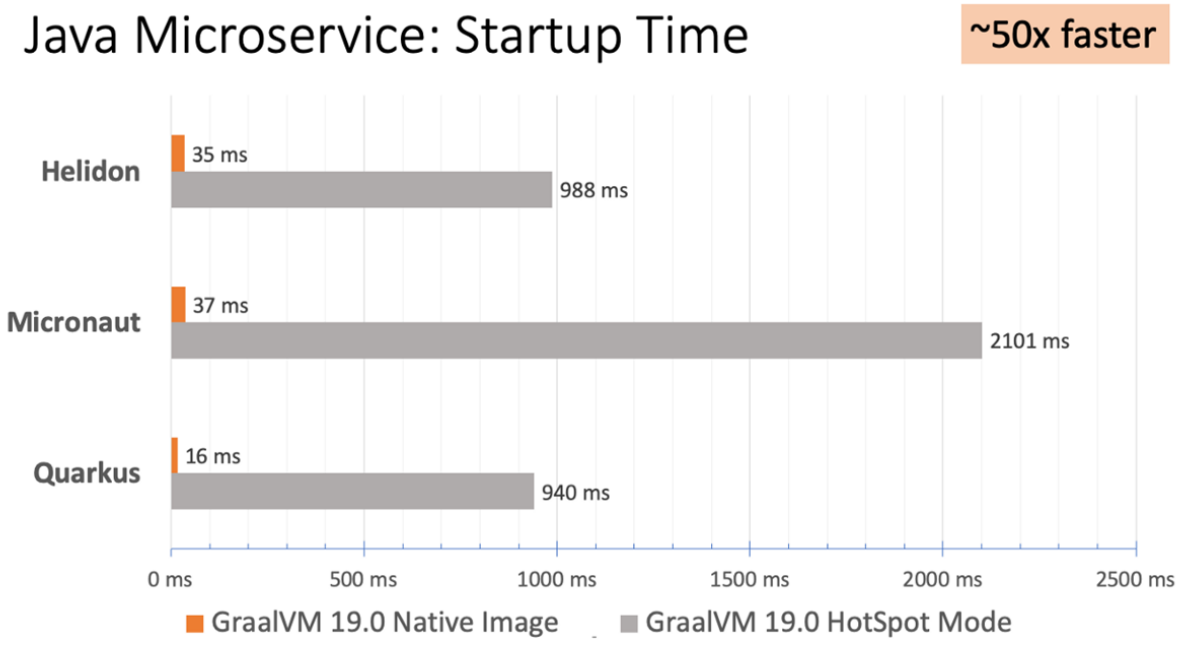
Cons

- Take a while to compile
- Not included by default in graalvm
- Platform specific
- Can have worse performance

Native image memory footprint



Native image startup time

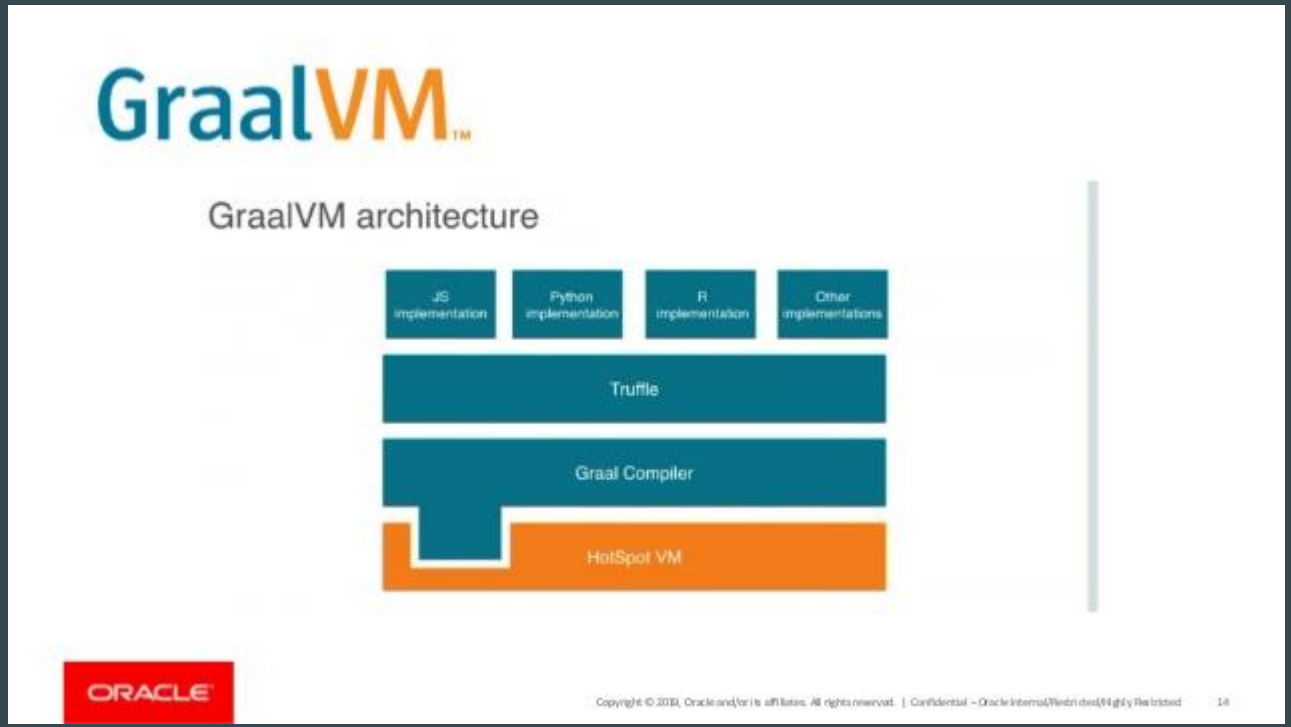


GraalVM as a platform

As mentioned earlier, GraalVM is more of a platform than a JVM, and ideally will evolve into that direction in the future. To move in this direction, GraalVM gives users the ability to:

- Implement their own language
  - Using the Language API, offered by the Language Implementation Framework
- Implement their own tooling
  - Using the Instrument API, offered by the Language Implementation Framework

GraalVM structure

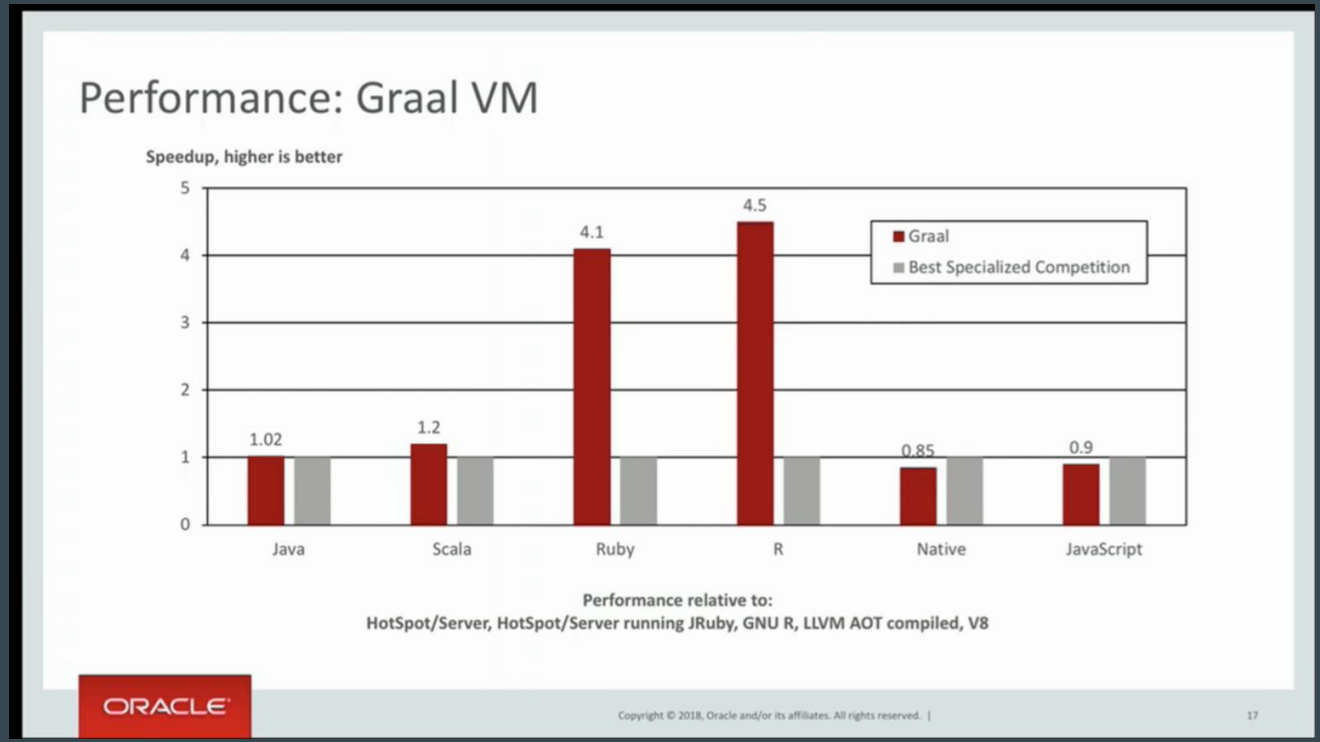


Performance

GraalVM is better optimized than the default JVM when it comes to:

- High abstraction applications (many abstract classes, interfaces, inner classes)
- Applications which make use of newer, more functional, java features (Streams, Lambdas).

Performance



Security

- Security model
- Security manager
- Untrusted code

Thank you for your attention!