Developer Live

JAVA INNOVATIONS

Fast and Efficient Java Microservices

Alina Yurenko
Developer Advocate for GraalVM, Oracle

Java Innovations with GraalVM

High Performance 🚀

Optimize application performance with GraalVM compiler

Fast Startup ☁️

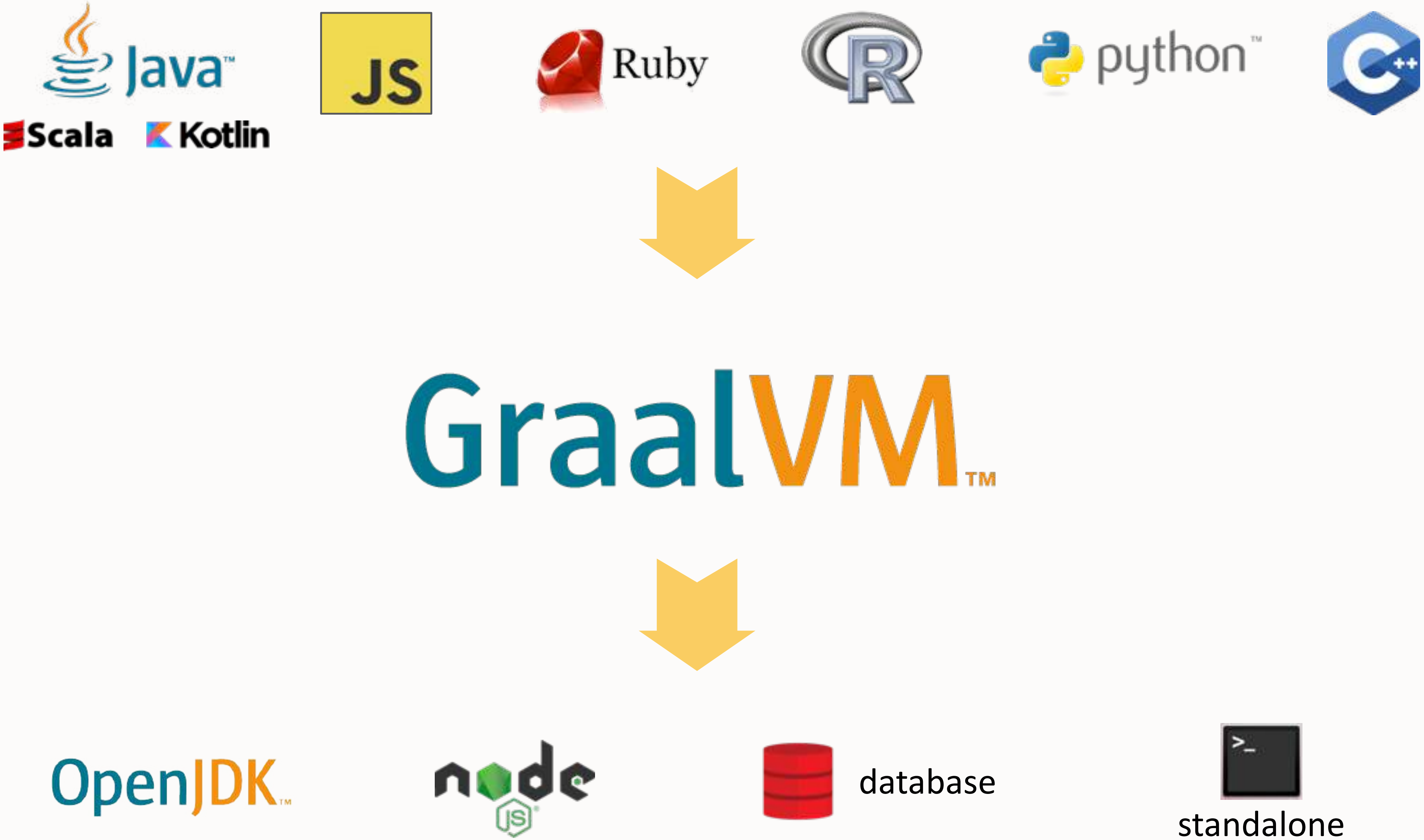
Compile your application AOT and start instantly

Polyglot 🏗️

Mix & match languages with seamless interop

Open Source 👥

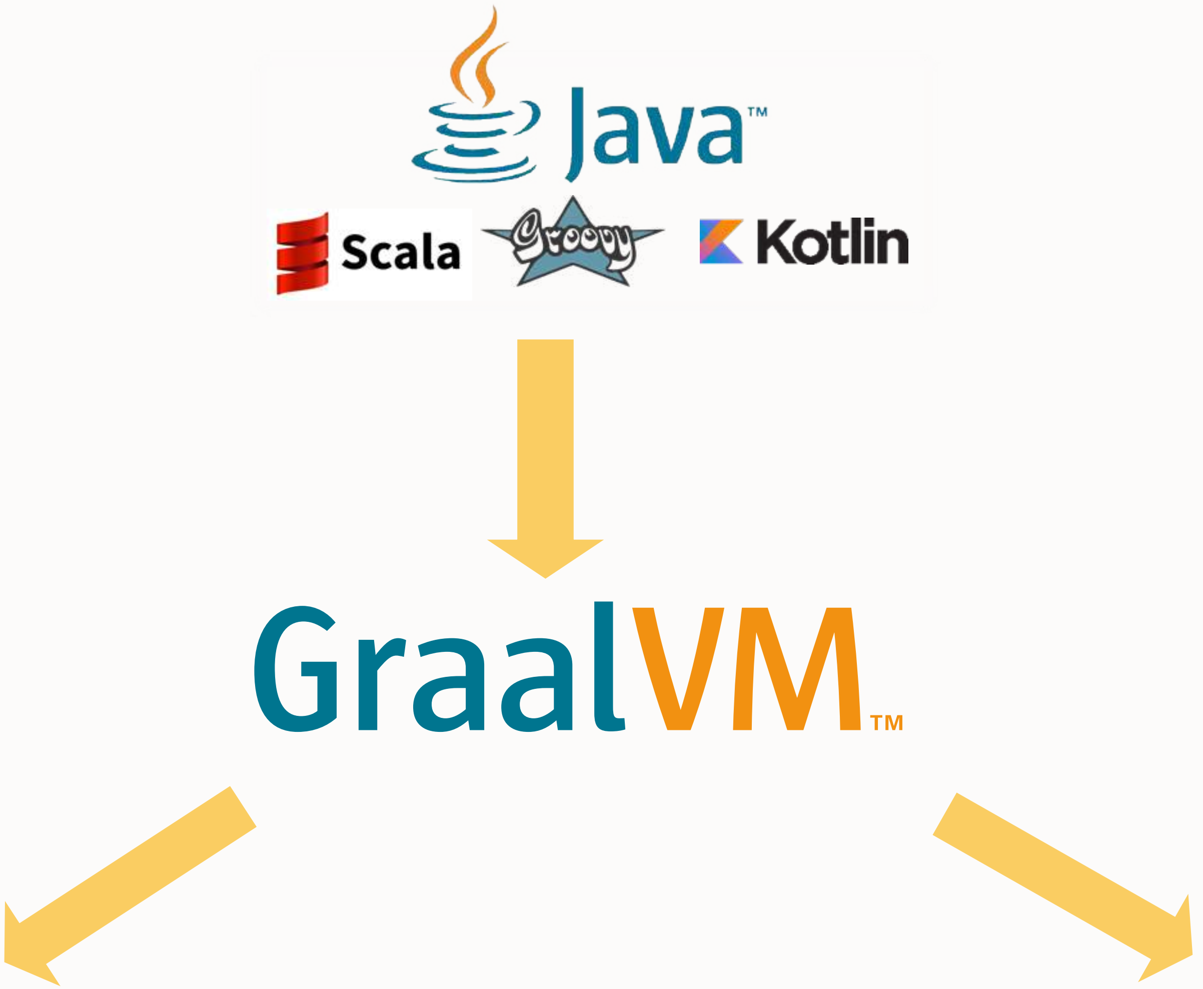
See what's inside, track features progress, contribute



database

standalone

GraalVM for Java Microservices



GraalVM™

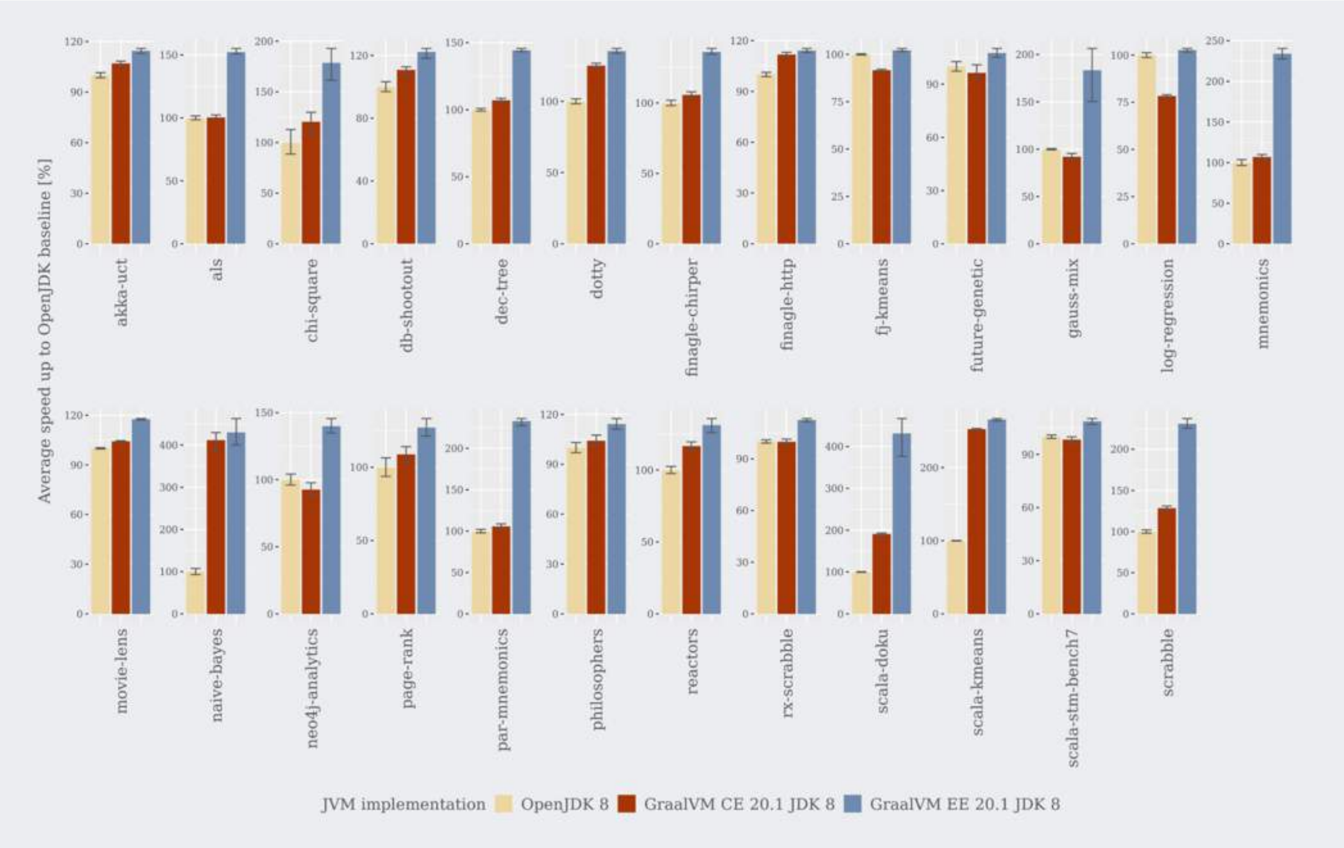
JIT

java MyMainClass

AOT

native-image MyMainClass
./mymainclass

Accelerated performance in JIT mode



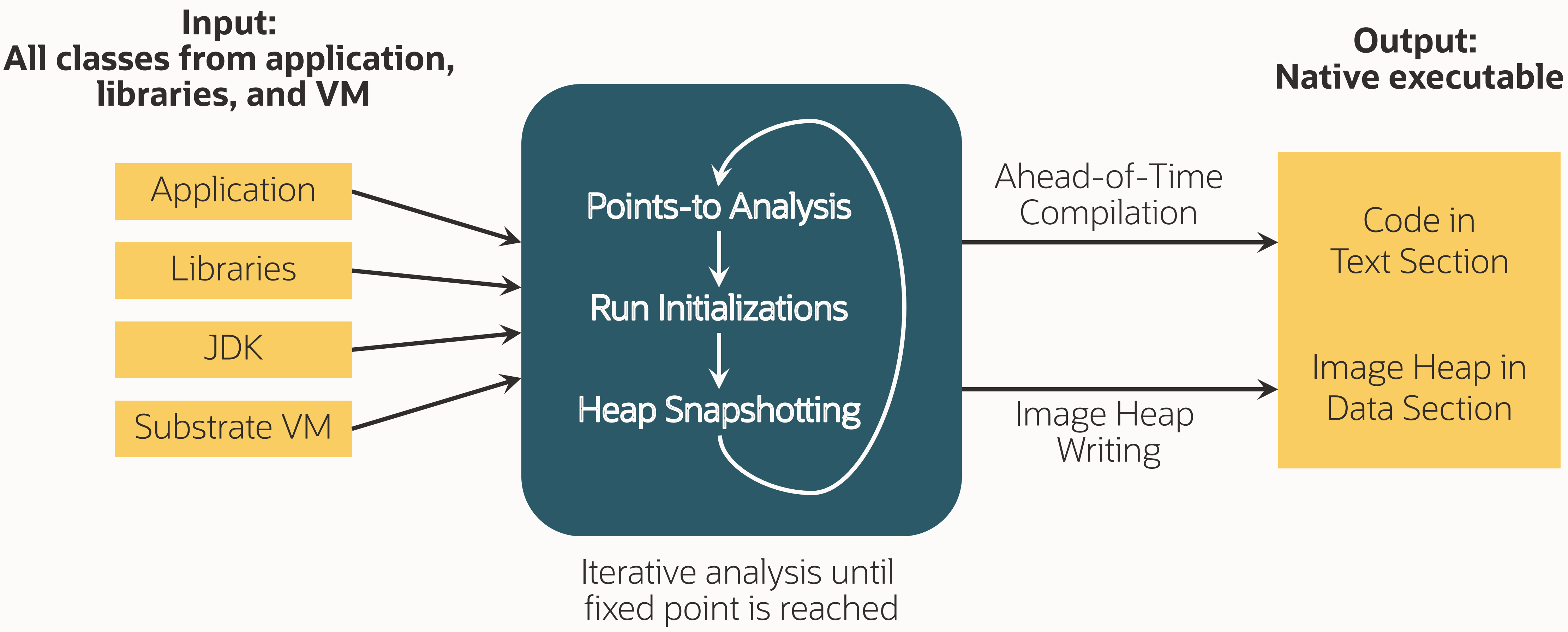
Renaissance benchmark suite: renaissance.dev

Startup performance

GraalVM Native Image

- Allows to compile Java programs into standalone native executables;
- Instant startup;
- Low memory footprint;
- works well with Java microservices frameworks.

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

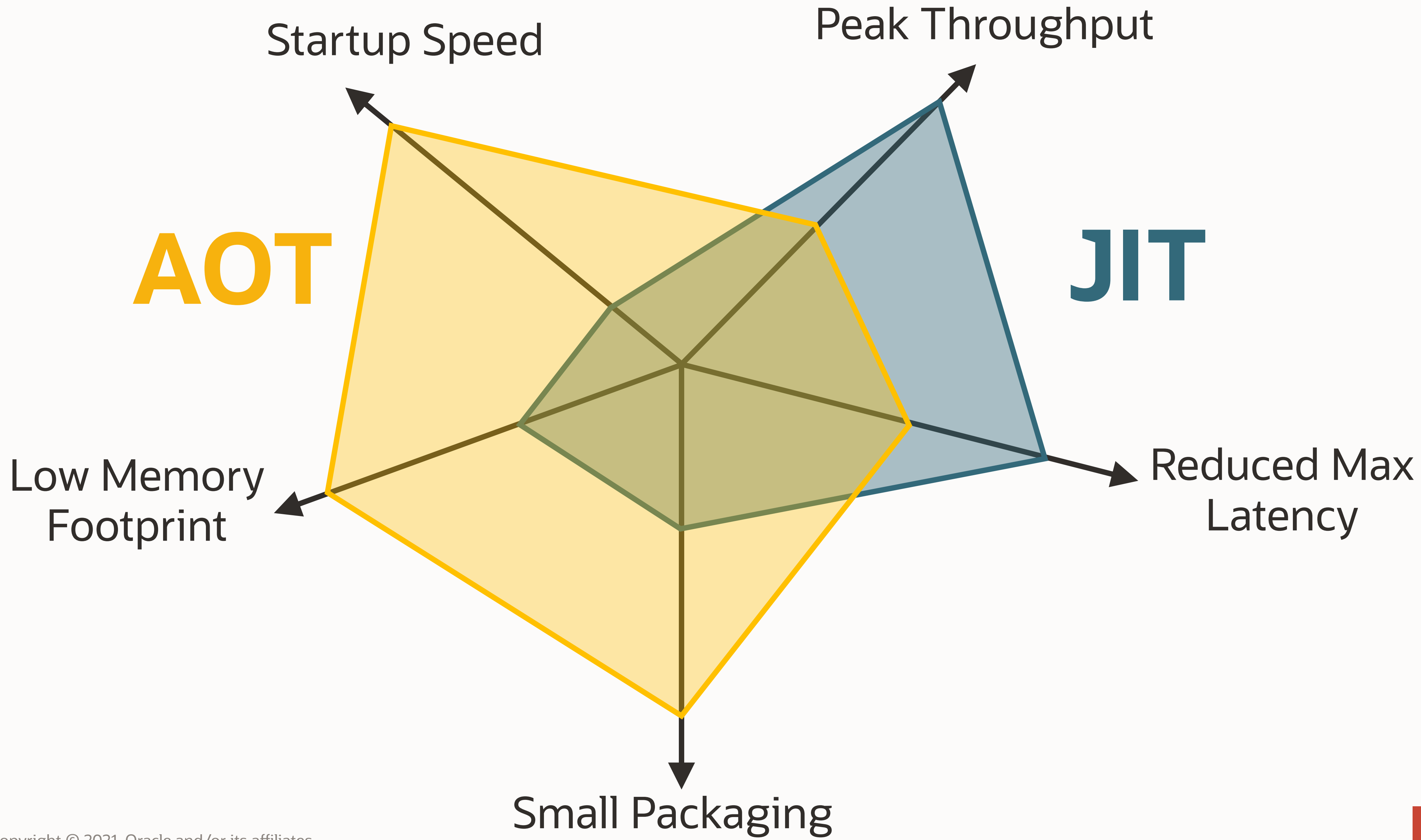
AOT vs JIT: Memory Footprint

JIT

- Loaded JVM executable
- Application data
- Loaded bytecodes
- Reflection meta-data
- Code cache
- Profiling data
- JIT compiler data structures

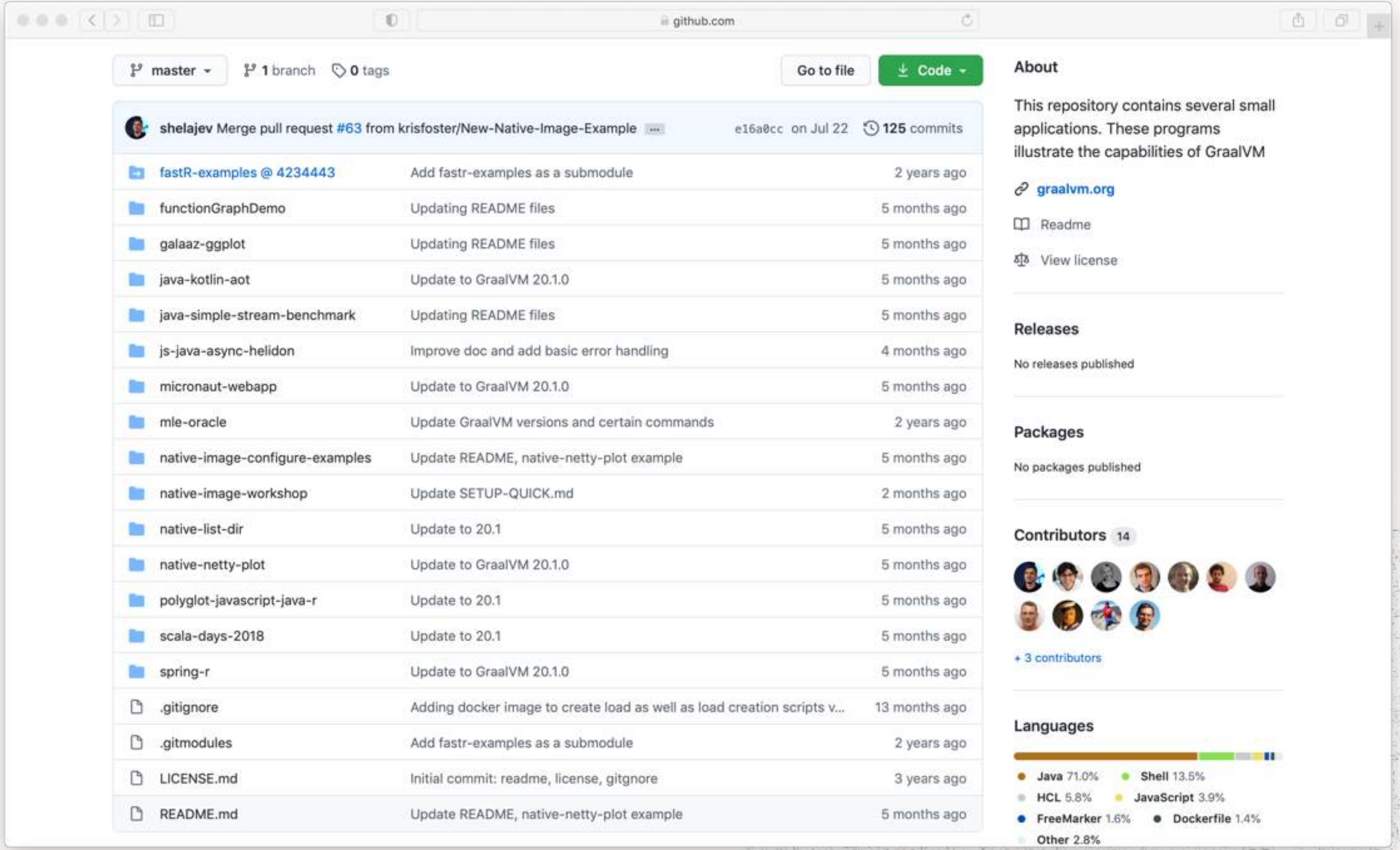
AOT

- Loaded application executable
- Application data


Startup Speed
Peak Throughput
AOT
JIT
Low Memory Footprint
Reduced Max Latency
Small Packaging

Create your first GraalVM application

Try sample apps!



shelajev Merge pull request #63 from krisfoster/New-Native-Image-Example … e16a0cc on Jul 22 125 commits

fastR-examples @ 4234443	Add fastr-examples as a submodule	2 years ago
functionGraphDemo	Updating README files	5 months ago
galaaz-ggplot	Updating README files	5 months ago
java-kotlin-aot	Update to GraalVM 20.1.0	5 months ago
java-simple-stream-benchmark	Updating README files	5 months ago
js-java-async-helidon	Improve doc and add basic error handling	4 months ago
micronaut-webapp	Update to GraalVM 20.1.0	5 months ago
mle-oracle	Update GraalVM versions and certain commands	2 years ago
native-image-configure-examples	Update README, native-netty-plot example	5 months ago
native-image-workshop	Update SETUP-QUICK.md	2 months ago
native-list-dir	Update to 20.1	5 months ago
native-netty-plot	Update to GraalVM 20.1.0	5 months ago
polyglot-javascript-java-r	Update to 20.1	5 months ago
scala-days-2018	Update to 20.1	5 months ago
spring-r	Update to GraalVM 20.1.0	5 months ago
.gitignore	Adding docker image to create load as well as load creation scripts v...	13 months ago
.gitmodules	Add fastr-examples as a submodule	2 years ago
LICENSE.md	Initial commit: readme, license, gitgnore	3 years ago
README.md	Update README, native-netty-plot example	5 months ago

About

This repository contains several small applications. These programs illustrate the capabilities of GraalVM

graalvm.org

Readme

View license

Releases

No releases published

Packages

No packages published

Contributors 14

+ 3 contributors

Languages

- Java 71.0%
- Shell 13.5%
- HCL 5.8%
- JavaScript 3.9%
- FreeMarker 1.6%
- Dockerfile 1.4%
- Other 2.8%

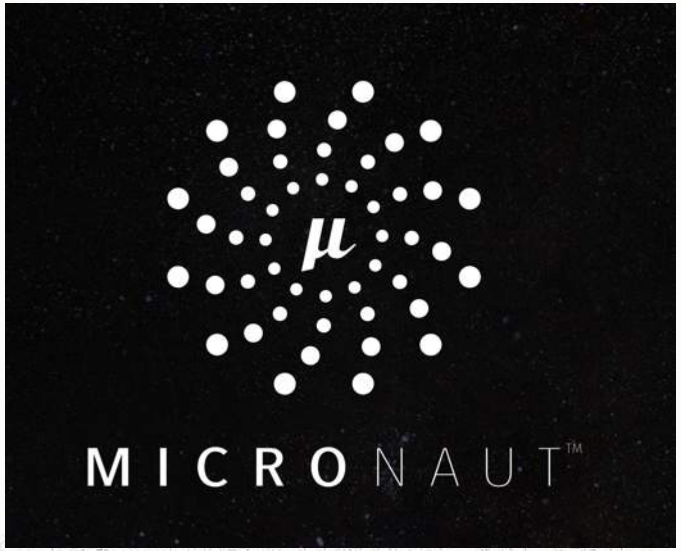
github.com/graalvm/graalvm-demos

Micronaut: how to

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: micronaut.io/
- Documentation: micronaut.io/documentation.html
- Micronaut Launch (Project Generator): micronaut.io/launch/



Helidon & GraalVM: how to

Quick example of using Helidon and GraalVM:
https://helidon.io/docs/latest/#/guides/36_graalnative

Video tutorial: Helidon MP and GraalVM:
https://www.youtube.com/watch?v=-y_MUgGyiW4

Helidon & GraalVM: results

Startup time improvement

Helidon SE Quickstart on the JVM	0.921 seconds
Helidon SE Quickstart as GraalVM Native image	0.026 seconds

```
REPOSITORY                TAG       IMAGE ID        CREATED          SIZE
quickstart-se-native      latest   1227ac82d199     5 days ago       21.4MB
```

Quarkus and GraalVM: how to and results

How to get started:

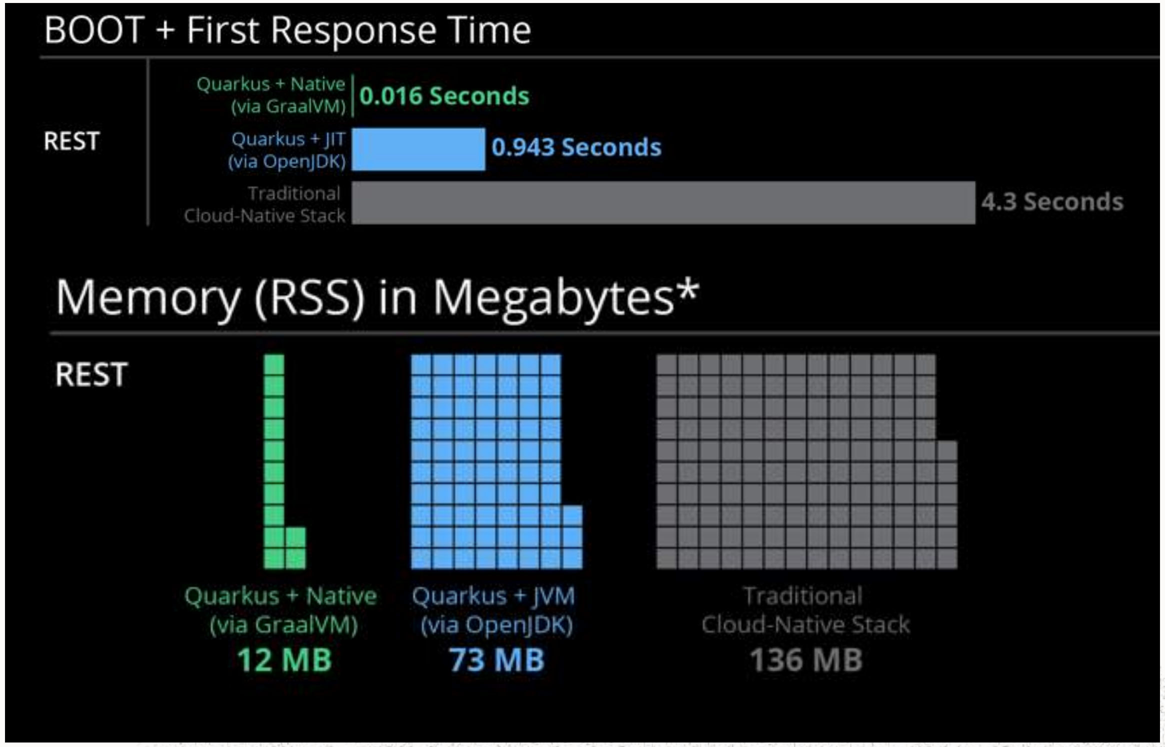
https://quarkus.io/get-started/

Build a native image:

quarkus.io/guides/building-native-image

Quickstarts:

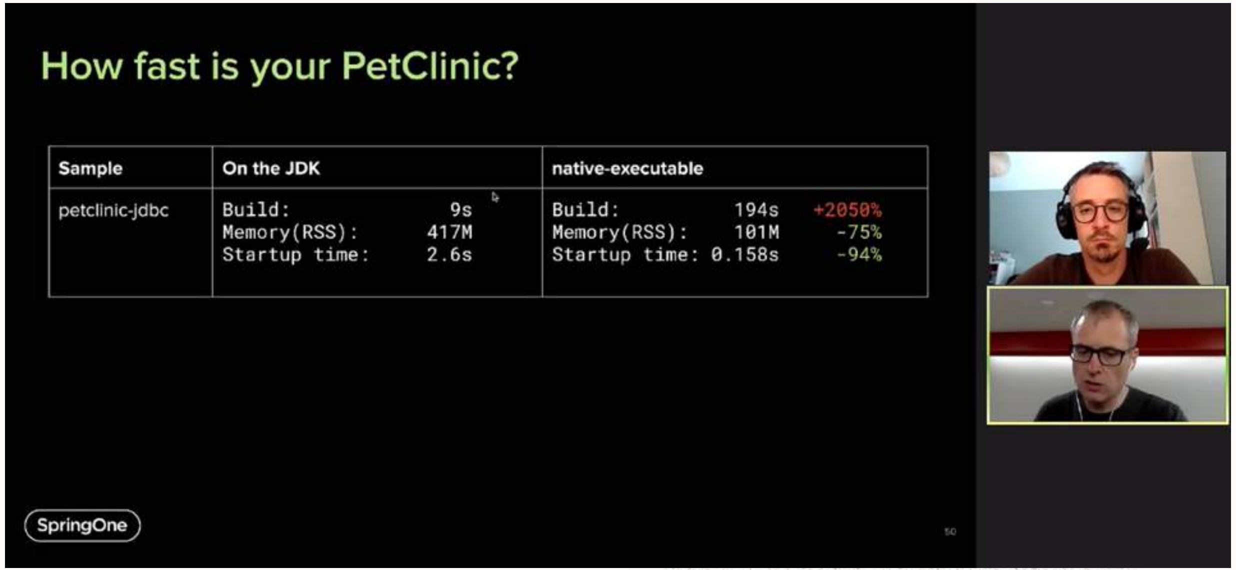
https://github.com/quarkusio/quarkus-quickstarts



Spring Boot & GraalVM: how to

State of the Spring GraalVM Native project:

github.com/spring-projects-experimental/spring-graalvm-native

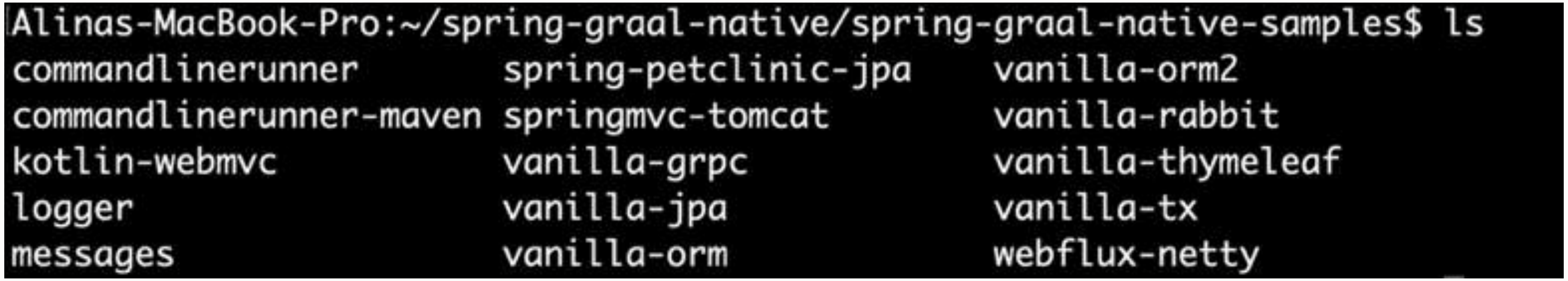


How fast is your PetClinic?

Sample	On the JDK	native-executable
petclinic-jdbc	Build: 9s Memory(RSS): 417M Startup time: 2.6s	Build: 194s +2050% Memory(RSS): 101M -75% Startup time: 0.158s -94%

https://www.youtube.com/watch?v=Um9djPTtPe0

Spring Boot & GraalVM: how to

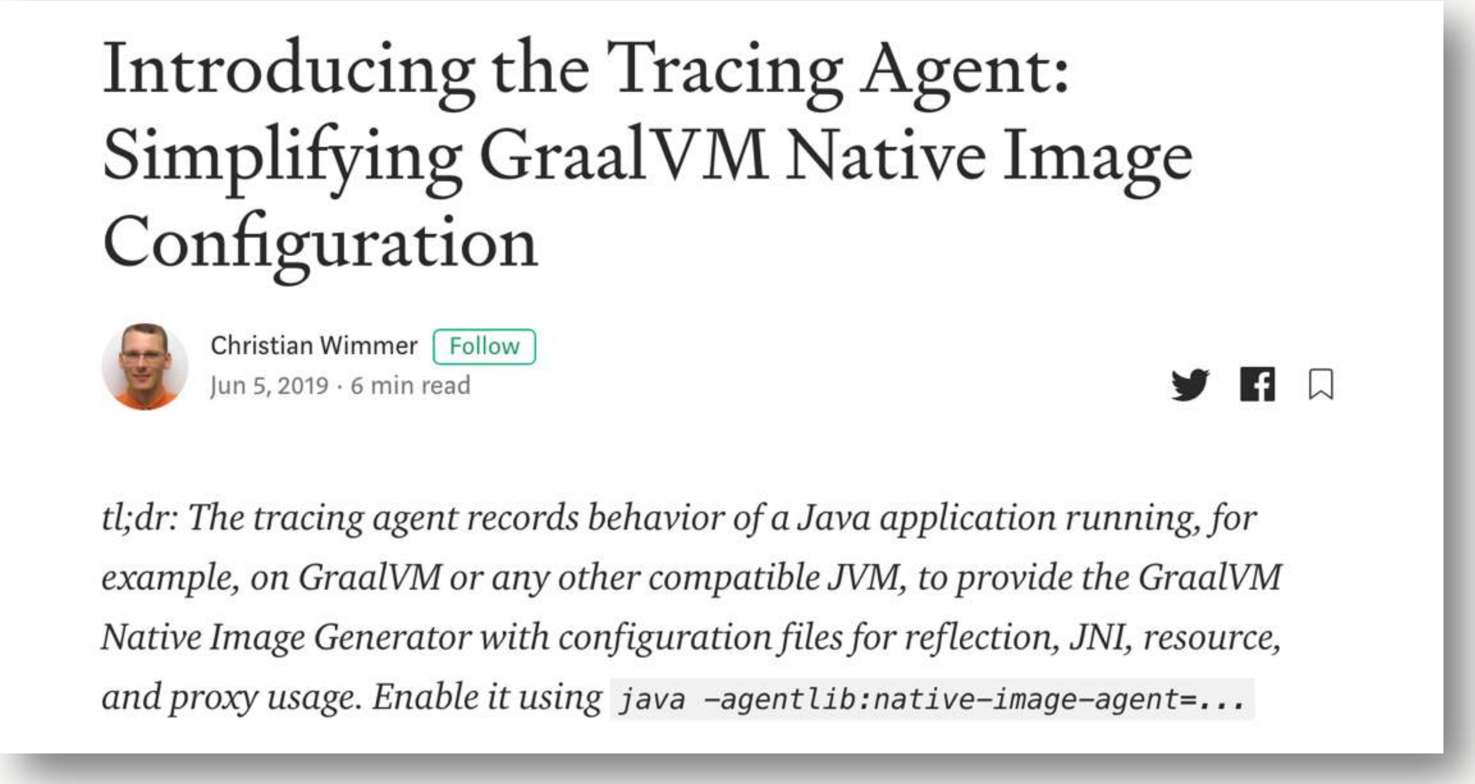


```
Alinas-MacBook-Pro:~/spring-graal-native/spring-graal-native-samples$ ls
commandlinerunner        spring-petclinic-jpa    vanilla-orm2
commandlinerunner-maven springmvc-tomcat        vanilla-rabbit
kotlin-webmvc           vanilla-grpc            vanilla-thymeleaf
logger                  vanilla-jpa             vanilla-tx
messages                vanilla-orm             webflux-netty
```

"Spring Graal Native" project: https://github.com/spring-projects-experimental/spring-graal-native

GraalVM native image for real-world projects

Simplify native image configuration



Introducing the Tracing Agent: Simplifying GraalVM Native Image Configuration

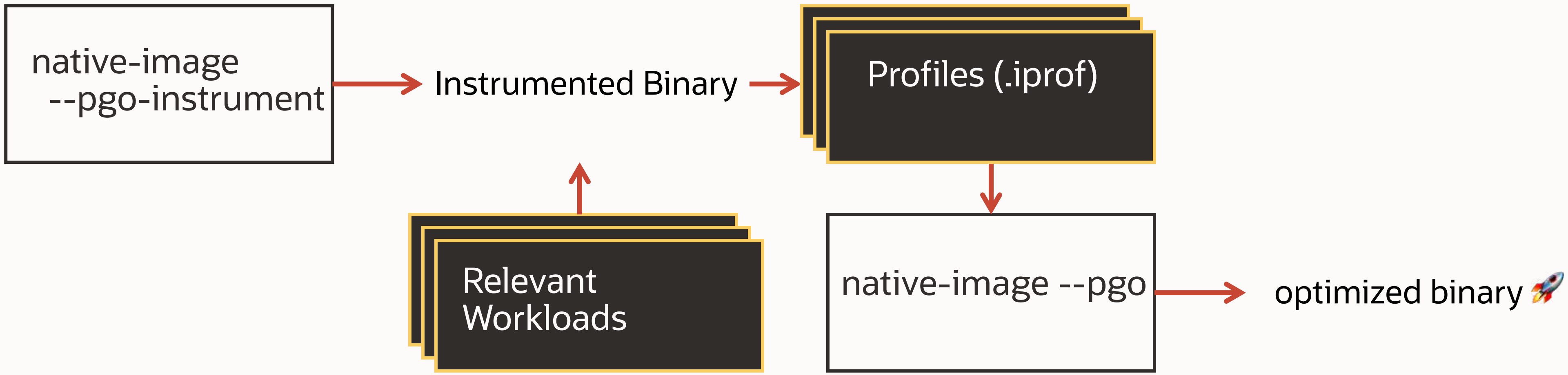
Christian Wimmer Follow

Jun 5, 2019 · 6 min read

tl;dr: The tracing agent records behavior of a Java application running, for example, on GraalVM or any other compatible JVM, to provide the GraalVM Native Image Generator with configuration files for reflection, JNI, resource, and proxy usage. Enable it using `java -agentlib:native-image-agent=...`

medium.com/graalvm/introducing-the-tracing-agent-simplifying-graalvm-native-image-configuration-c3b56c486271

Profile-Guided Optimizations (PGO)



native-image --pgo-instrument

Instrumented Binary

Profiles (.iprof)

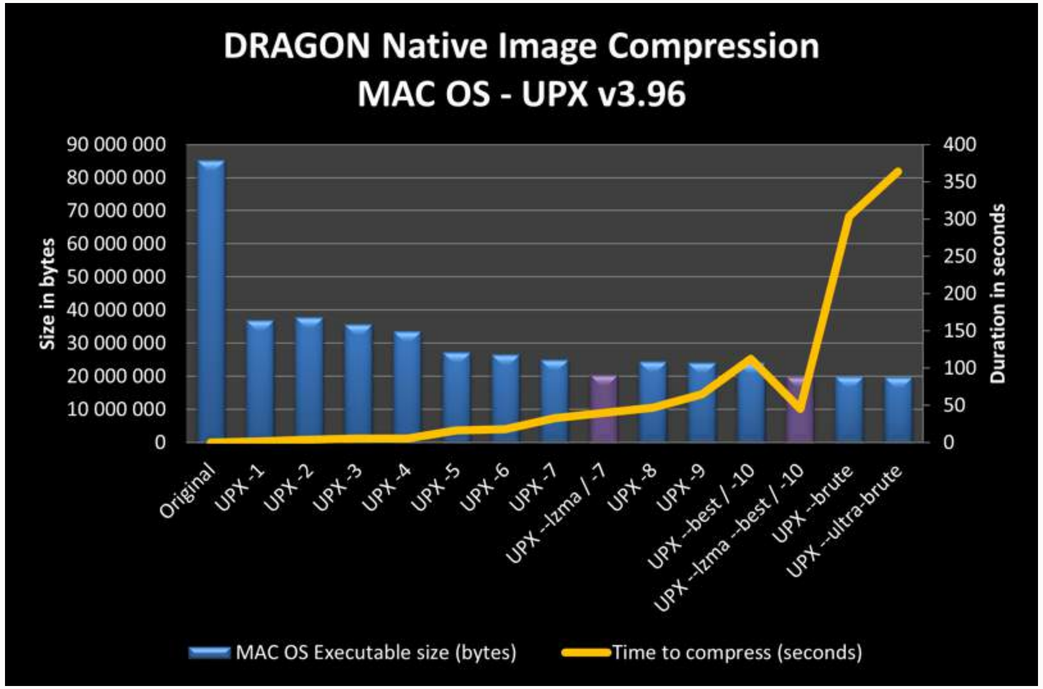
Relevant Workloads

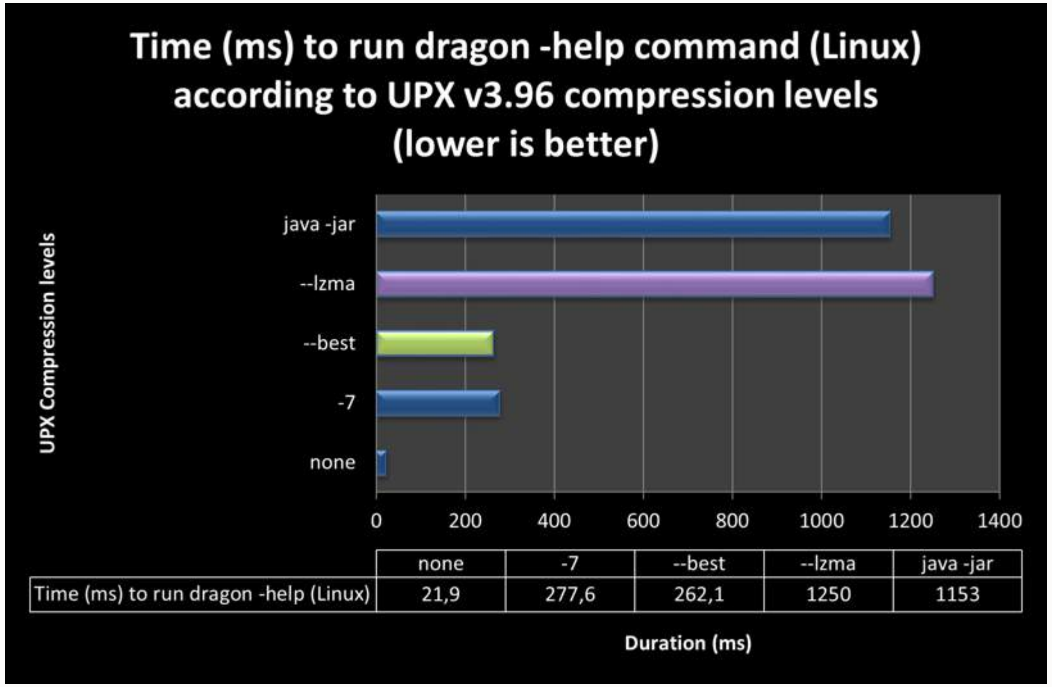
native-image --pgo

optimized binary 🚀

Optimizing the binary size even further

```
upx --best -k mynativeimage
```

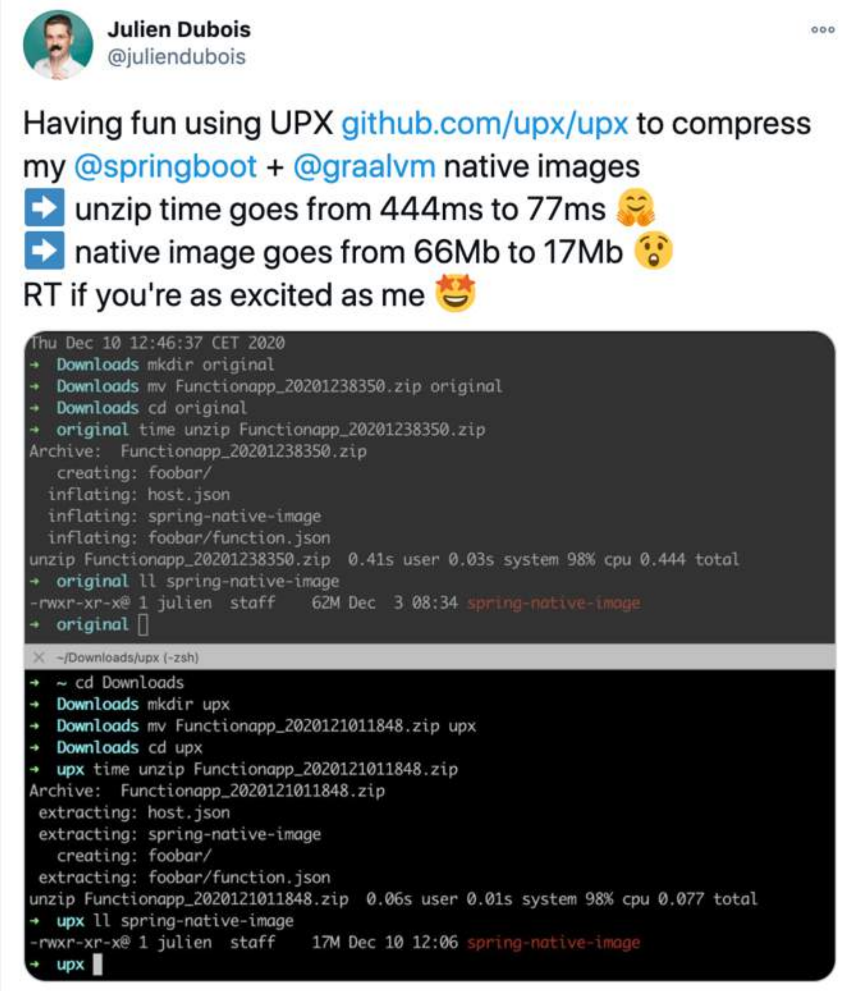




	none	-7	--best	--lzma	java -jar
Time (ms) to run dragon -help (Linux)	21,9	277,6	262,1	1250	1153

medium.com/graalvm/compressed-graalvm-native-images-4d233766a214

Optimizing the binary size even further



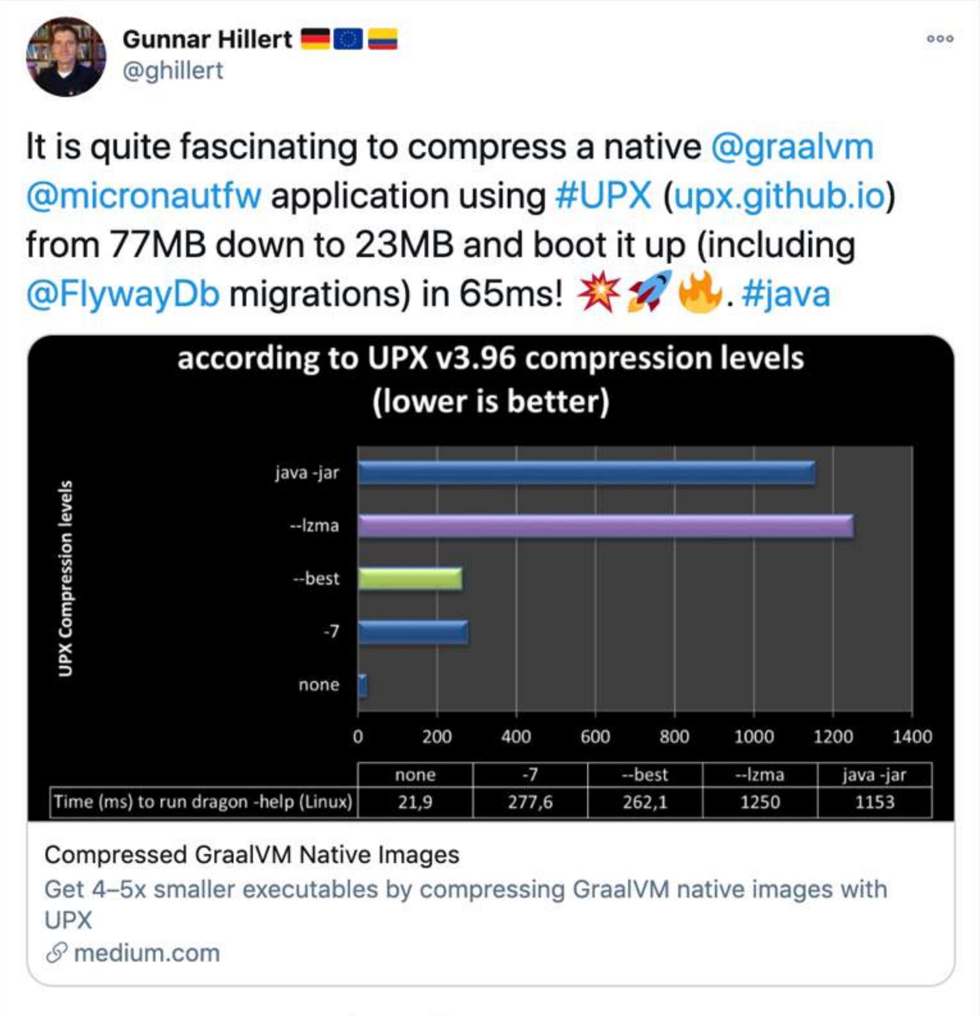
Julien Dubois
@juliendubois

Having fun using UPX 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 12:46:37 CET 2020
→  Downloads mkdir original
→  Downloads mv Functionapp_20201238350.zip original
→  Downloads cd original
→  original time unzip Functionapp_20201238350.zip
Archive:  Functionapp_20201238350.zip
   creating: foobar/
  inflating: host.json
  inflating: spring-native-image
  inflating: foobar/function.json
unzip Functionapp_20201238350.zip  0.41s user 0.03s system 98% cpu 0.444 total
→  original ll spring-native-image
-rwxr-xr-x@ 1 julien  staff    62M Dec  3 08:34 spring-native-image
→  original []
```

```
→  ~ cd Downloads
→  Downloads mkdir upx
→  Downloads mv Functionapp_2020121011848.zip upx
→  Downloads cd upx
→  upx time unzip Functionapp_2020121011848.zip
Archive:  Functionapp_2020121011848.zip
 extracting: host.json
 extracting: spring-native-image
   creating: foobar/
 extracting: foobar/function.json
unzip Functionapp_2020121011848.zip  0.06s user 0.01s system 98% cpu 0.077 total
→  upx ll spring-native-image
-rwxr-xr-x@ 1 julien  staff    17M Dec 10 12:06 spring-native-image
→  upx █
```

twitter.com/juliendubois/status/1337005381436977152



Gunnar Hillert 🇩🇪🇪🇺🇨🇴
@ghillert

It is quite fascinating to compress a native @graalvm @micronautfw application using #UPX (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)

	none	-7	--best	--lzma	java -jar
Time (ms) to run dragon -help (Linux)	21,9	277,6	262,1	1250	1153

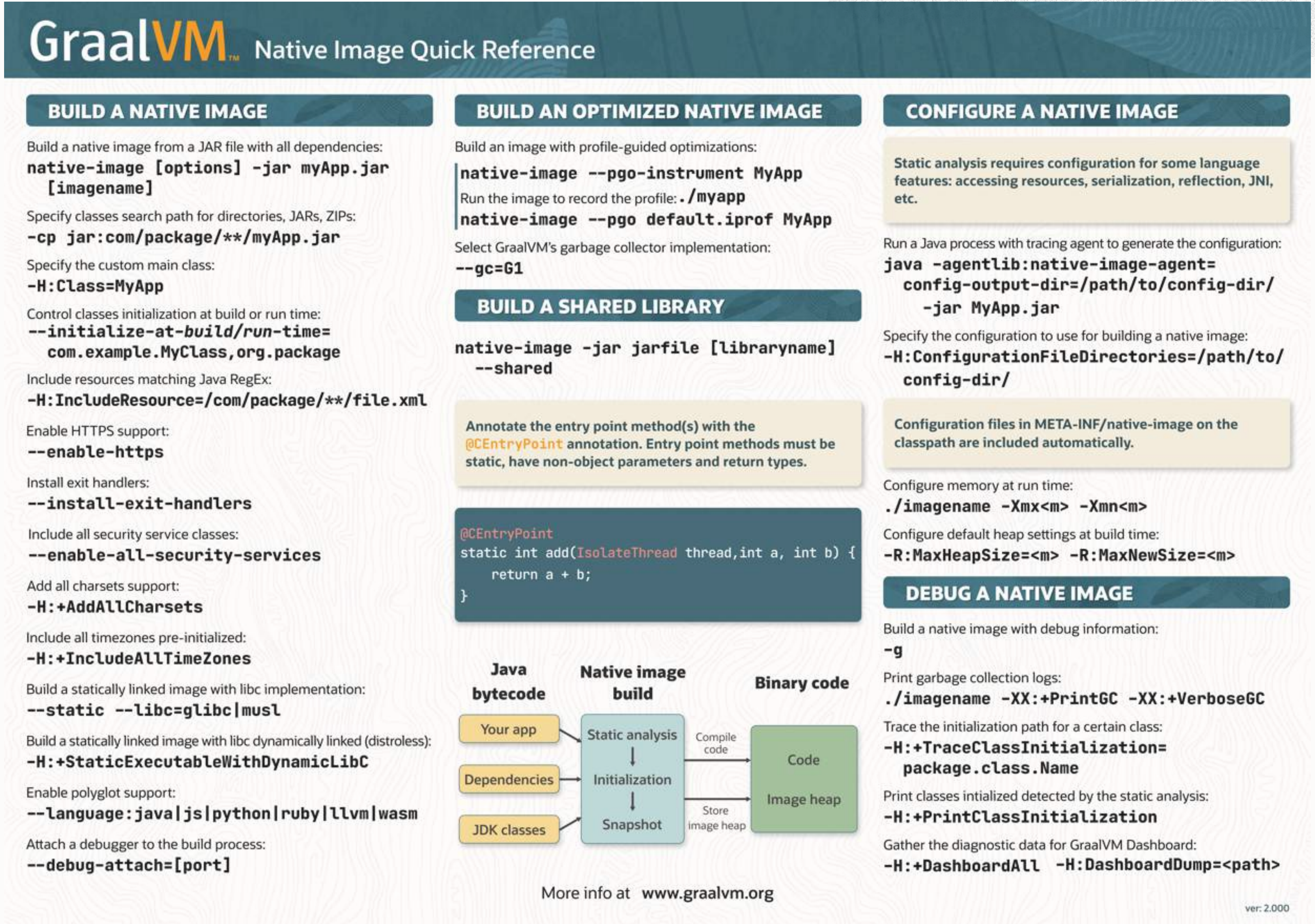
Compressed GraalVM Native Images
Get 4–5x smaller executables by compressing GraalVM native images with UPX
🔗 medium.com

twitter.com/ghillert/status/1341582440523939840

Resources for working with native image

- How to configure native image generation
- Native image Maven plugin
- Memory & GC configuration
- Understand class initialization in native Image
- Isolates in native image

Resources for working with native image



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
  [imagename]
```

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 polyglot 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 `@CEntryPoint` annotation. Entry point methods must be static, have non-object parameters and return types.

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

Java bytecode

Native image build

Binary code

Your app

Dependencies

JDK classes

Static analysis

Initialization

Snapshot

Compile code

Store image heap

Code

Image heap

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:

```
./imagename -Xmx<m> -Xmn<m>
```

Configure default heap settings at build time:

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

DEBUG A NATIVE IMAGE

Build a native image with debug information:

```
-g
```

Print garbage collection logs:

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

Trace the initialization path for a certain class:

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

Print classes intialized detected by the static analysis:

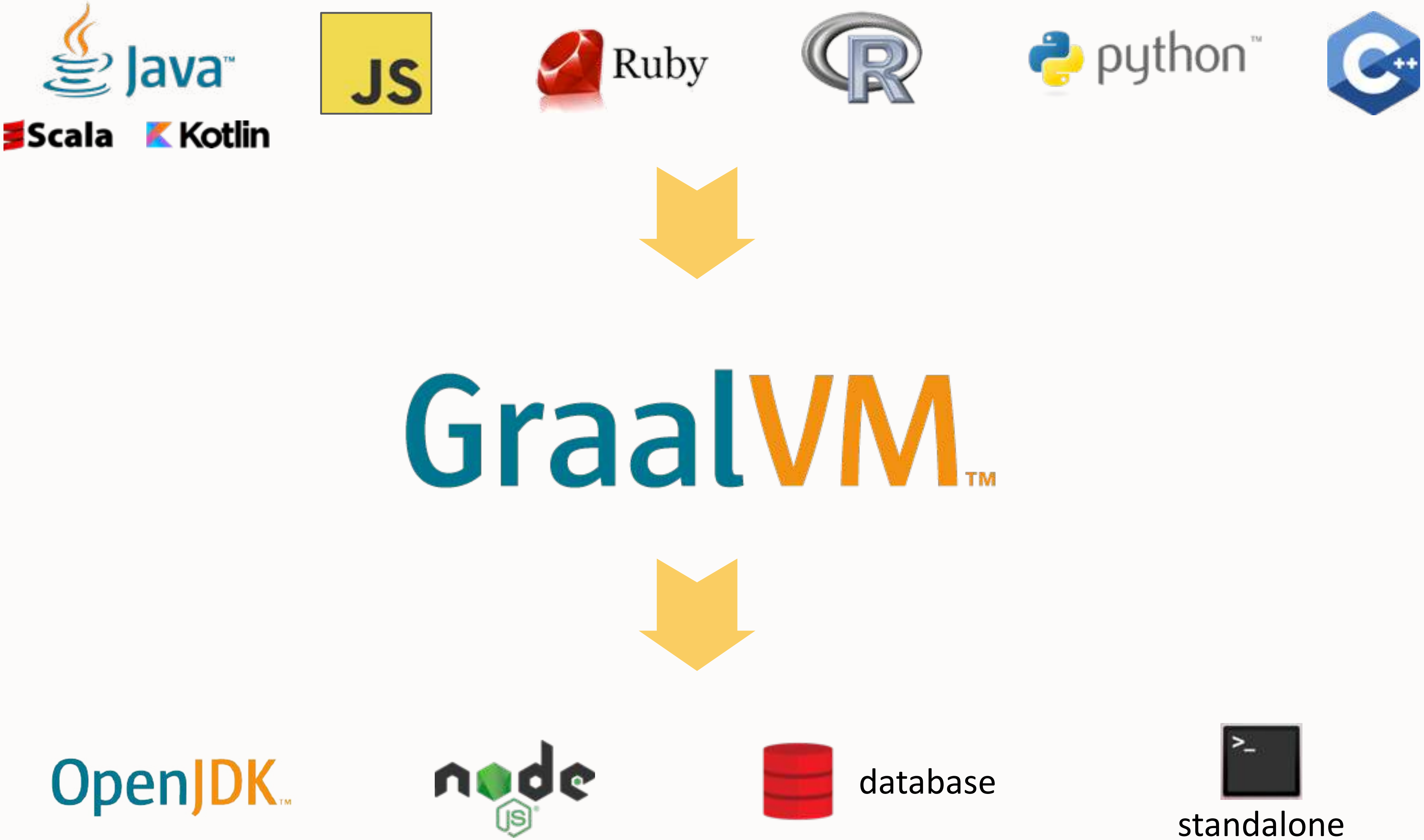
```
-H:+PrintClassInitialization
```

Gather the diagnostic data for GraalVM Dashboard:

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

ver: 2.000

GraalVM
Languages Ecosystem



database

standalone

JavaScript & Node.js

- ECMAScript 2020 complaint JavaScript engine;
- Access to GraalVM language interoperability and common tooling;
- Constantly tested against 90,000+ npm modules, `including express, react, async, request`

Many ways to use GraalVM JavaScript

GraalVM as VM

- ✓ Best integrated
- ✓ Best tested
- ✓ Supports Node.js
- – Extra download

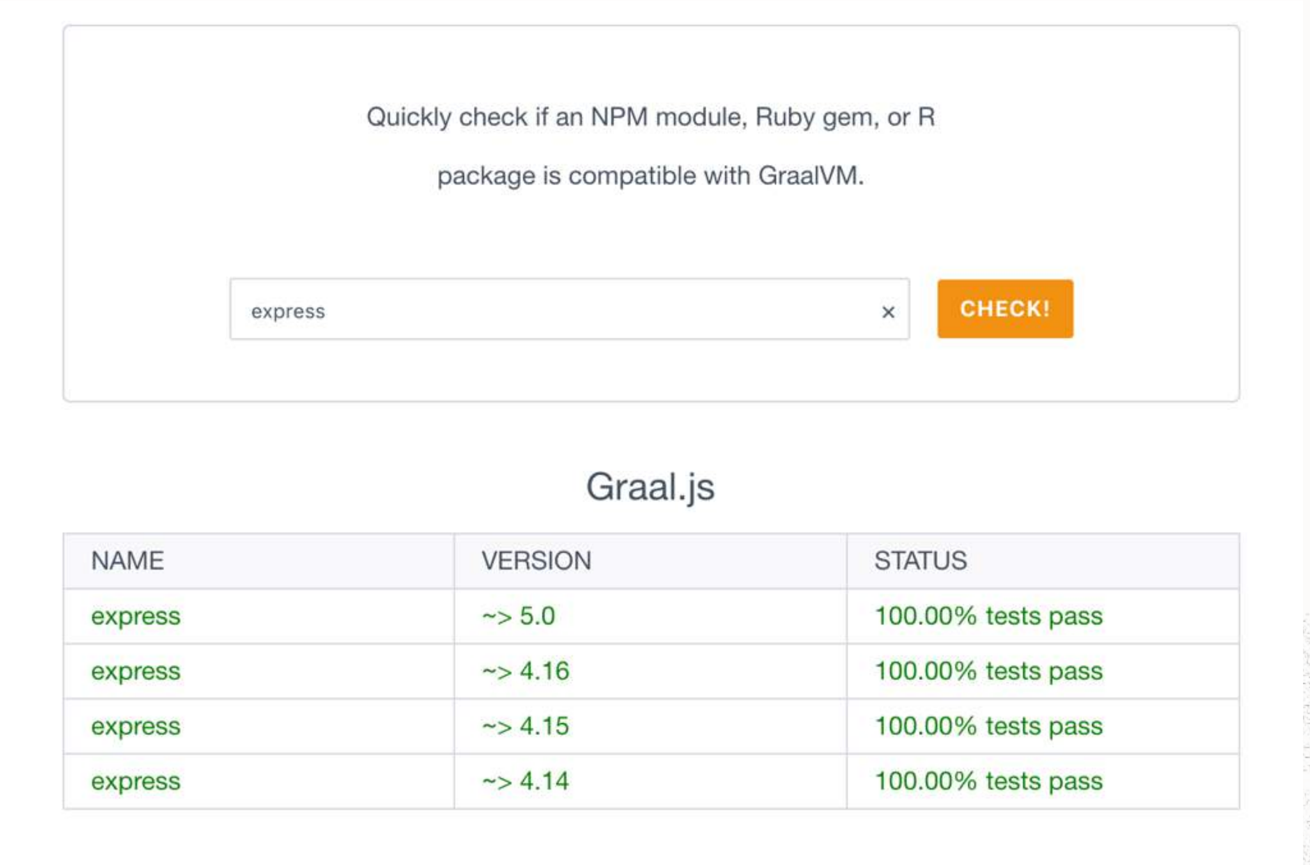
Stock JVM

- ✓ You already have the JVM
- ✓ Our JARs are on Maven
- – More variability

native-image of App+JS

- ✓ No JVM required
- ✓ No dependencies
- – Additional configuration
- – Less flexibility

Find out about your packages compatibility

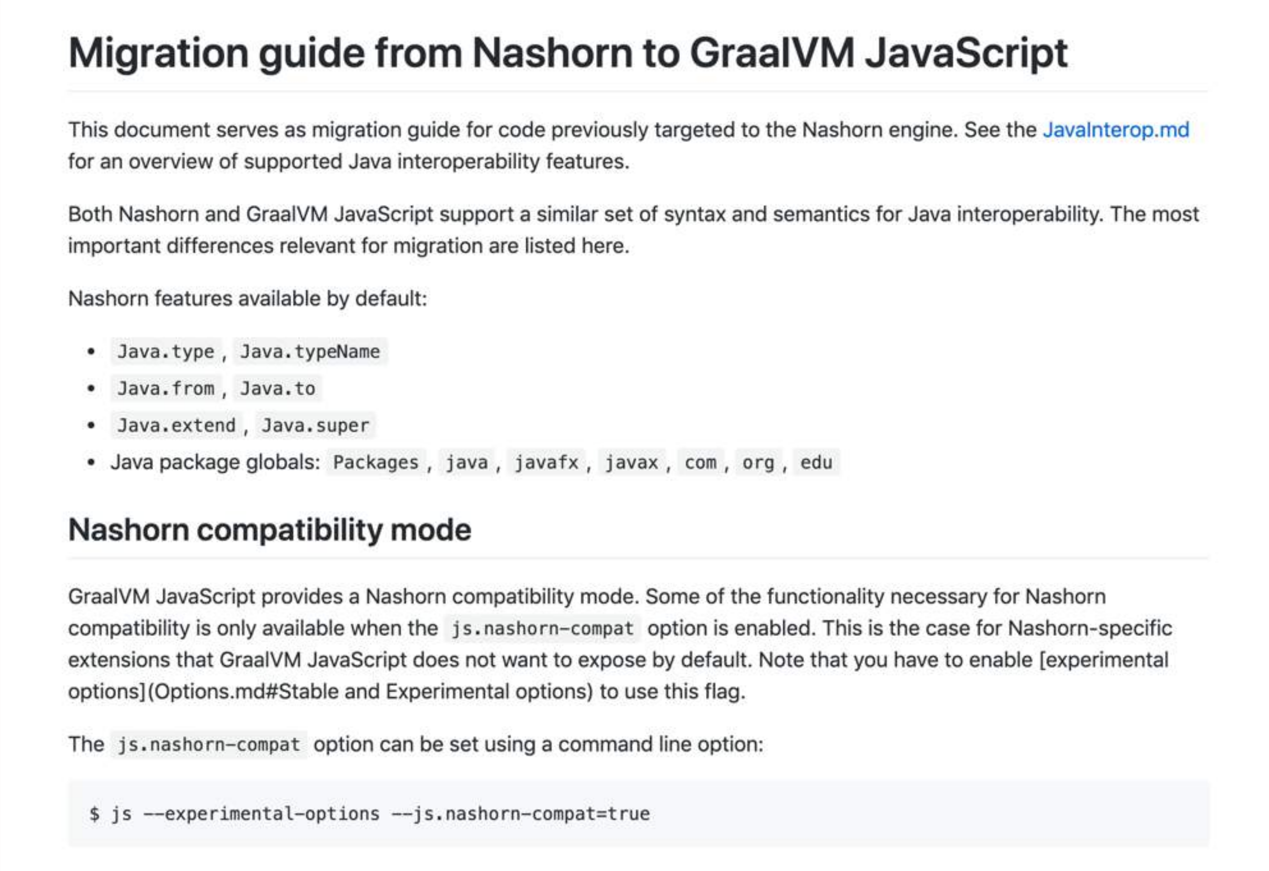


Quickly check if an NPM module, Ruby gem, or R package is compatible with GraalVM.

Graal.js

NAME	VERSION	STATUS
express	~> 5.0	100.00% tests pass
express	~> 4.16	100.00% tests pass
express	~> 4.15	100.00% tests pass
express	~> 4.14	100.00% tests pass

Nashorn Migration Guide



Migration guide from Nashorn to GraalVM JavaScript

This document serves as migration guide for code previously targeted to the Nashorn engine. See the JavaInterop.md for an overview of supported Java interoperability features.

Both Nashorn and GraalVM JavaScript support a similar set of syntax and semantics for Java interoperability. The most important differences relevant for migration are listed here.

Nashorn features available by default:

- `Java.type`, `Java.typeName`
- `Java.from`, `Java.to`
- `Java.extend`, `Java.super`
- Java package globals: `Packages`, `java`, `javafx`, `javax`, `com`, `org`, `edu`

Nashorn compatibility mode

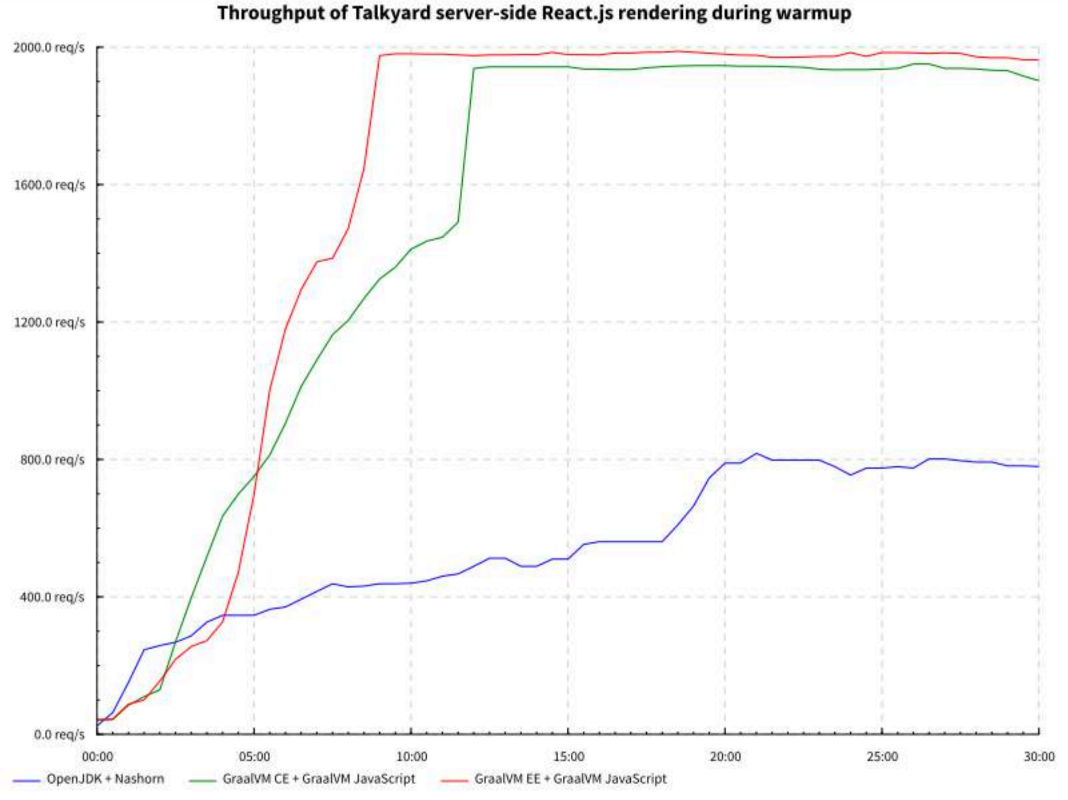
GraalVM JavaScript provides a Nashorn compatibility mode. Some of the functionality necessary for Nashorn compatibility is only available when the `js.nashorn-compat` option is enabled. This is the case for Nashorn-specific extensions that GraalVM JavaScript does not want to expose by default. Note that you have to enable [experimental options](Options.md#Stable and Experimental options) to use this flag.

The `js.nashorn-compat` option can be set using a command line option:

```
$ js --experimental-options --js.nashorn-compat=true
```

https://github.com/graalvm/graaljs/blob/master/docs/user/NashornMigrationGuide.md

React.js Server Side Rendering



- Example app: Talkyard.io
- Server-side part written in Scala, client side: React.js;
- Nashorn: **~800** renders per second;
- GraalVM: **~2000** renders per second.

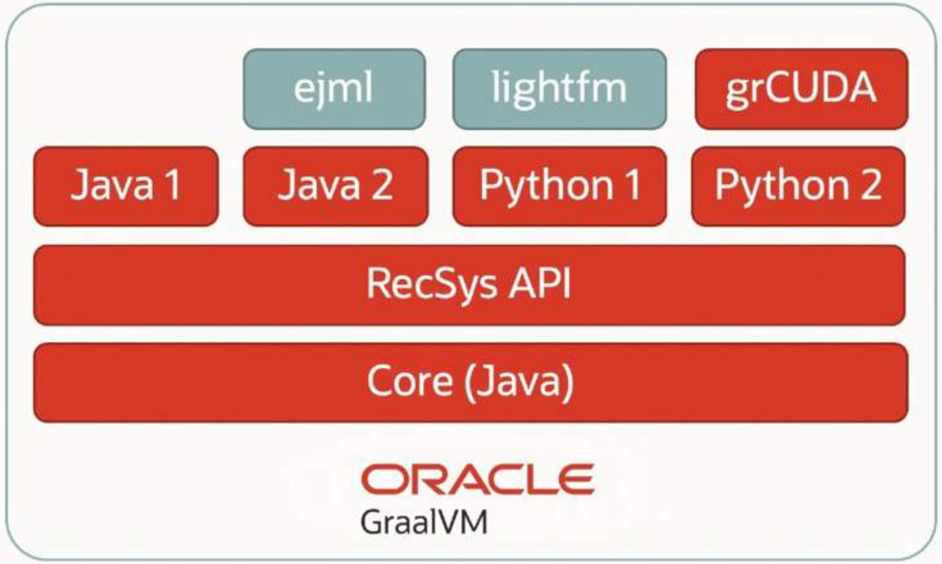
medium.com/graalvm/improve-react-js-server-side-rendering-by-150-with-graalvm-58a06ccb45df

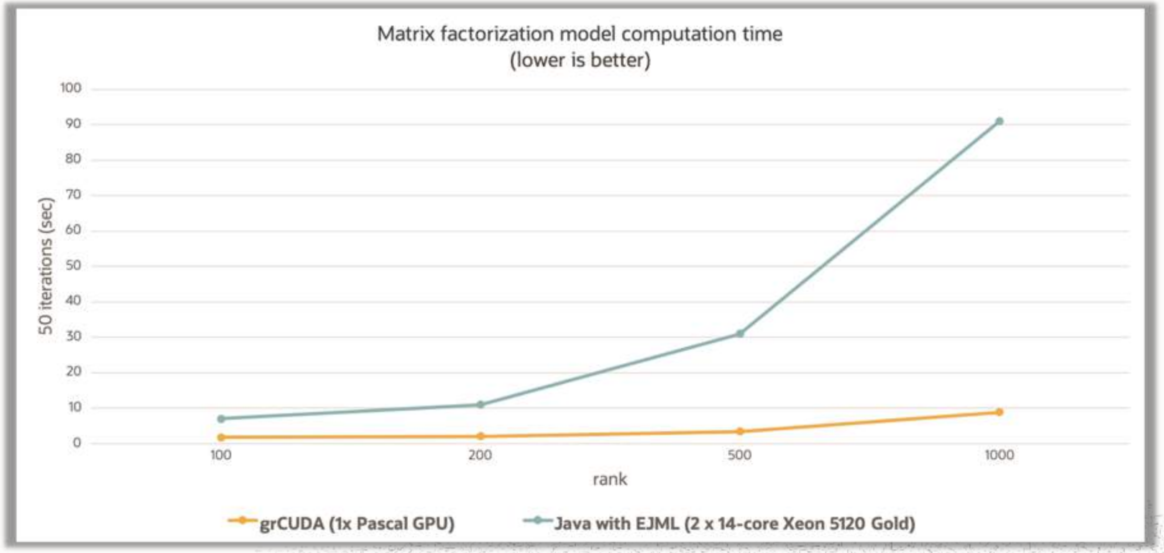
GraalVM Python

- Python 3 implementation;
- High performance;
- Focus on supporting SciPy and its constituent libraries;
- Easy interop with Java and the rest of GraalVM languages.

```
$ graalpython [options] [-c cmd | filename]
```

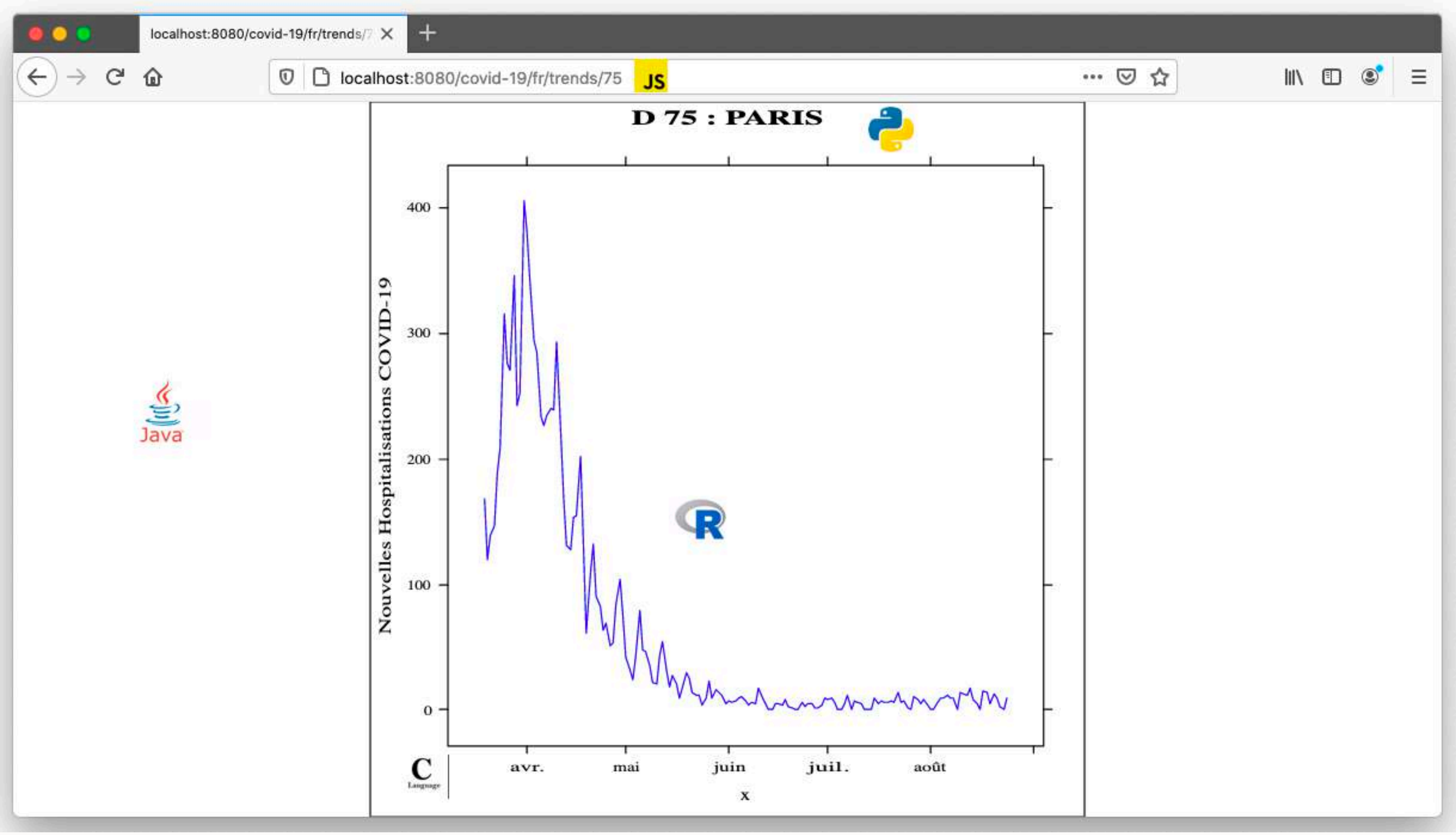
Recommender system POC at Oracle NetSuite





blogs.oracle.com/graalvm/optimizing-machine-learning-performance-at-netsuite-with-graalvm-and-nvidia-gpus-v2

Polyglot data science application with GraalVM


localhost:8080/covid-19/fr/trends/7
localhost:8080/covid-19/fr/trends/75
JS
D 75 : PARIS
Java
Nouvelles Hospitalisations COVID-19
400
300
200
100
0
avr.
mai
juin
juil.
août
x
C
Language
R

medium.com/@nelvadas/polyglot-micro-service-for-visualizing-covid-19-trends-with-graalvm-helidon-java-r-python-c-a3dce4262eb3

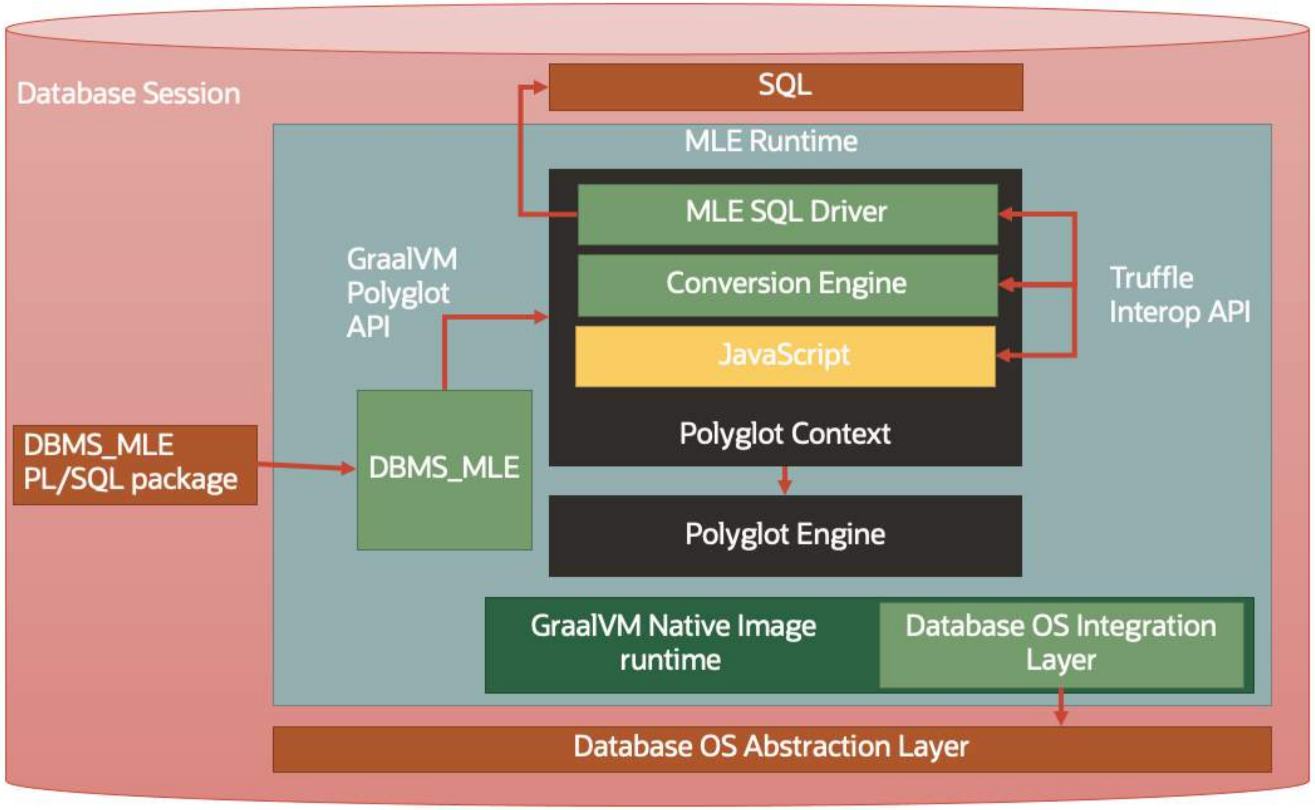
GraalVM Use Cases

Multilingual Engine: Executing JavaScript in Oracle Database

Oracle Database

Starting with 21c, Oracle Database can now execute JavaScript, powered by GraalVM:

Executing JavaScript in Oracle Database



ORACLE
Database

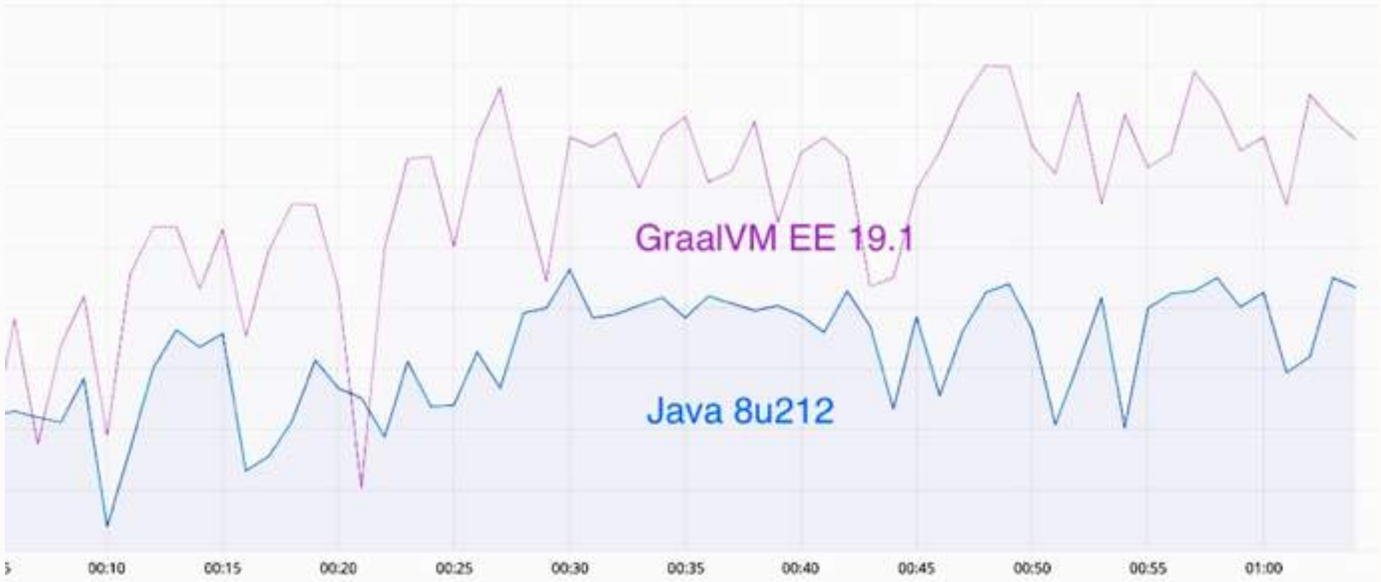
Migrated Monitoring Services to GraalVM

Oracle Cloud Infrastructure

Peak performance: +10%

Garbage collection time: -25%

Easy migration



https://blogs.oracle.com/cloud-infrastructure/graalvm-powers-oracle-cloud-infrastructure

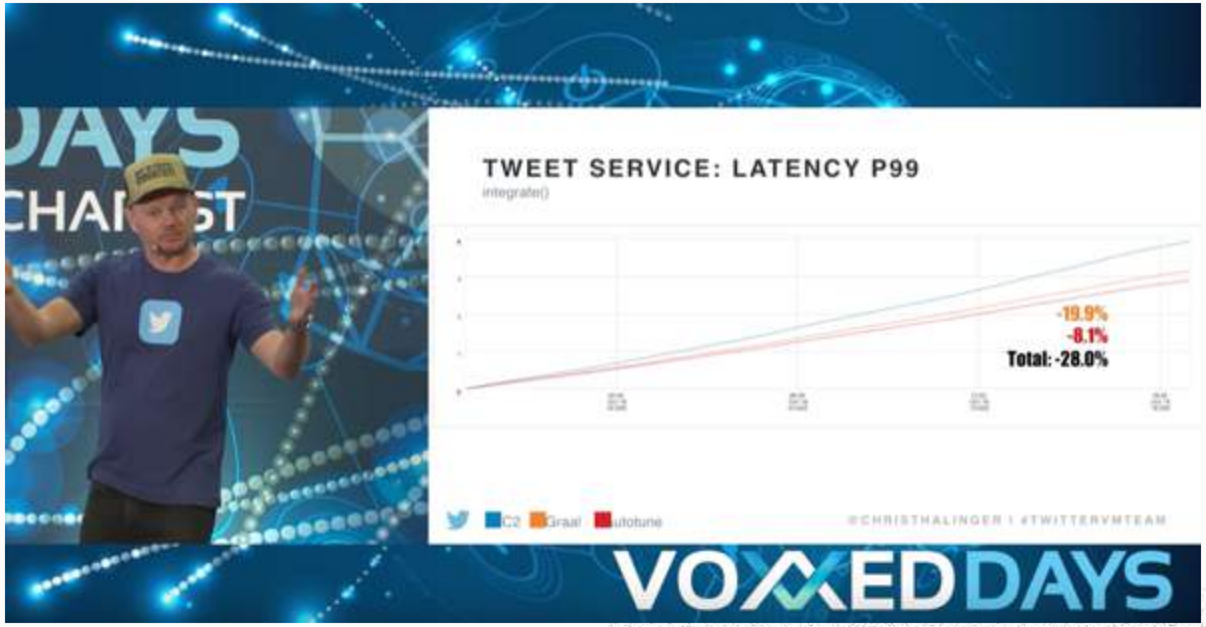
ORACLE
Cloud Infrastructure

Moved Scala Microservices to GraalVM

Twitter

Improved peak performance by 12% by just switching to GraalVM compiler, and up to 18.2% with additional configuration

Reduced P99 latency by 19.9%

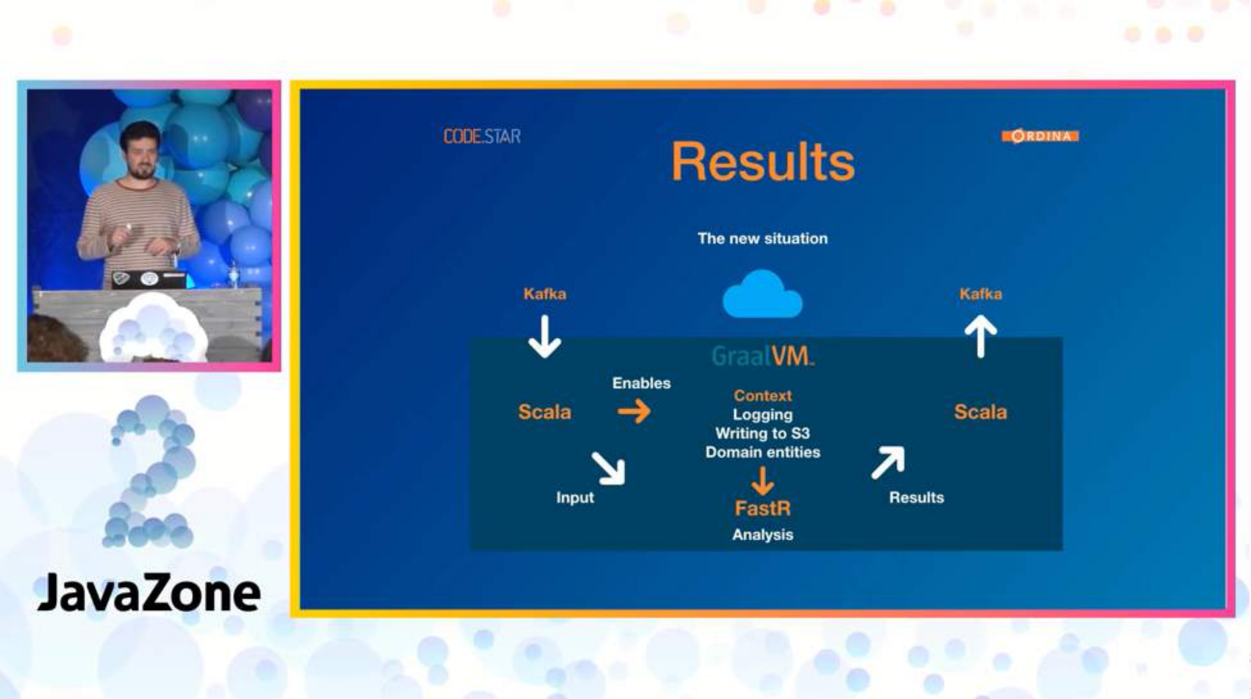


https://www.youtube.com/watch?v=W-5kUG8_mbk

Scala Microservices with R for Data Science

Dutch National Police

GraalVM enables easy collaboration between data scientists, who build statistical analysis functions in R, and developers, working with Scala and Java



CODE.STAR

ORDINA

Results

The new situation

Kafka

Scala

Enables

GraalVM.

Context
Logging
Writing to S3
Domain entities

Input

FastR
Analysis

Results

Kafka

Scala

JavaZone

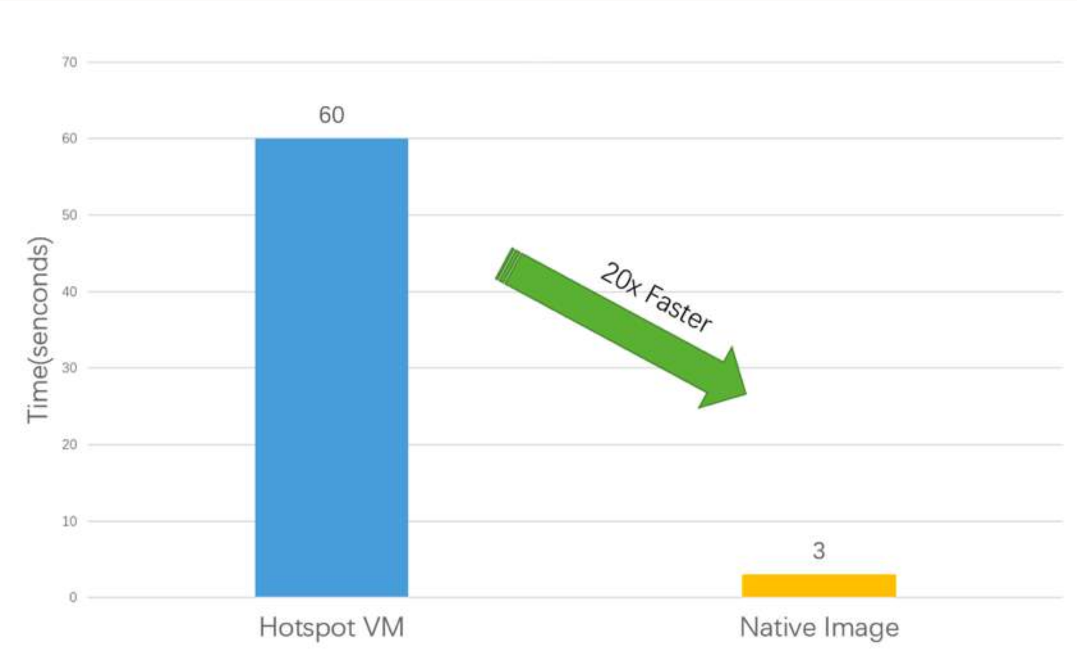
https://vimeo.com/360837119

Using GraalVM native image to improve startup performance

Alibaba

SOFABoot app: startup time decreased from 60 sec to 3 sec;

Micronaut app: startup time decreased from 454.33 ms to 4.27 ms, with memory cost of 1/6 from original.

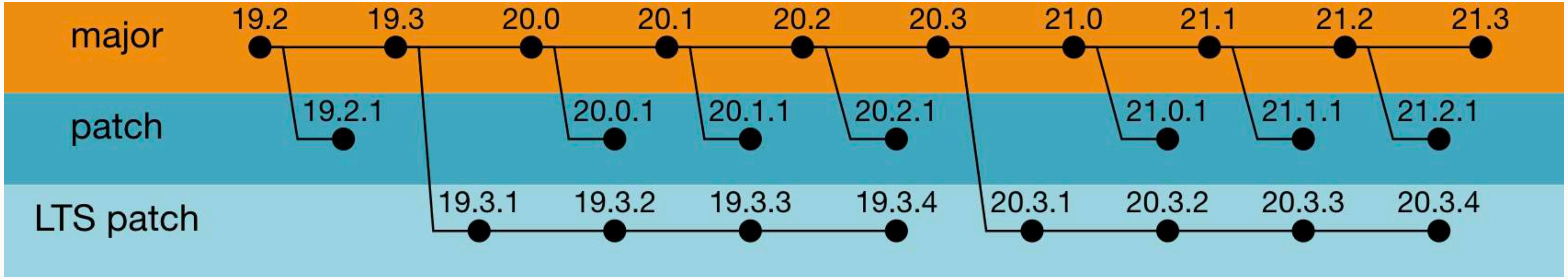


https://medium.com/graalvm/static-compilation-of-java-applications-at-alibaba-at-scale-2944163c92e

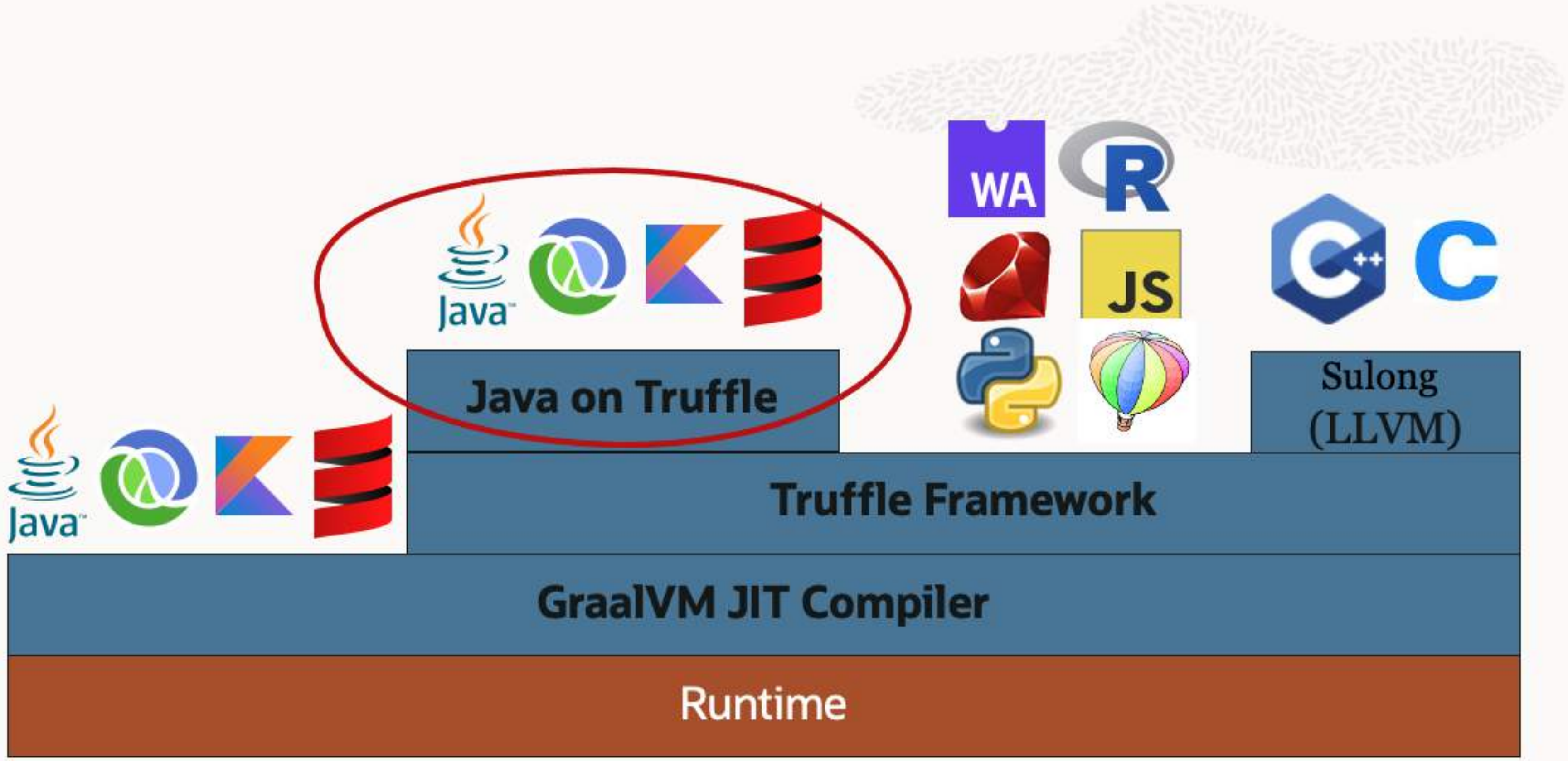
Project Roadmap and Community

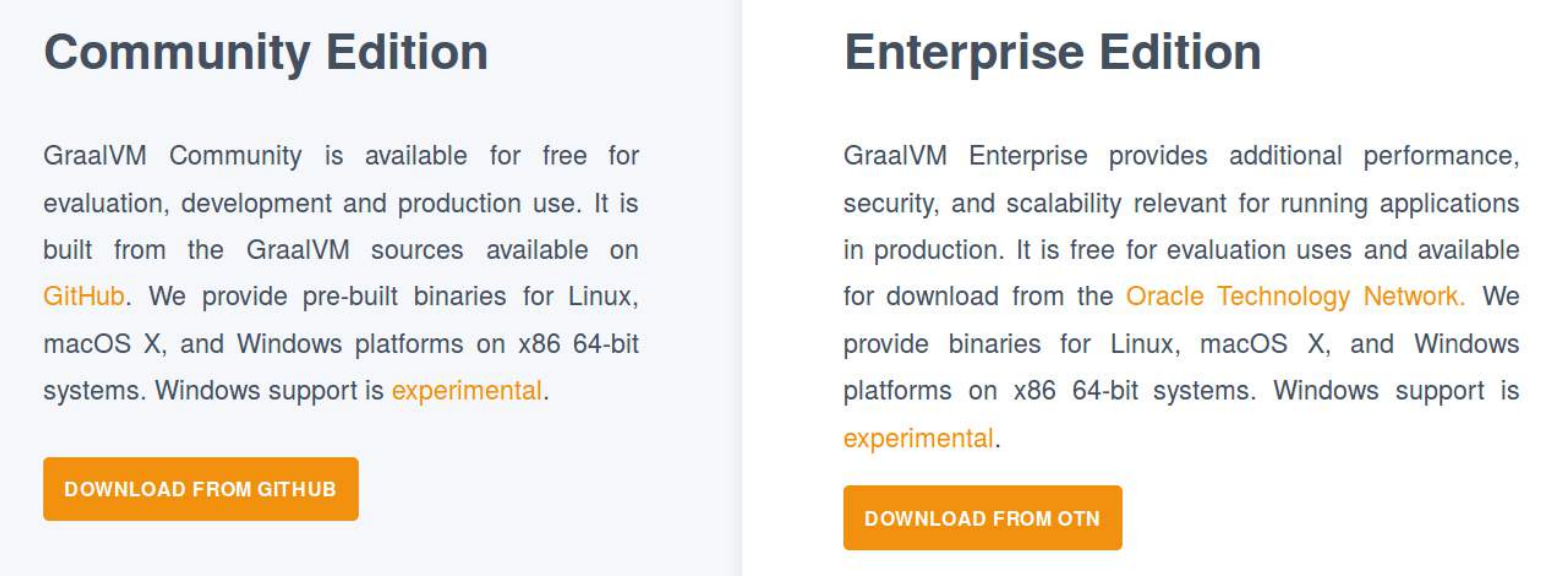
Version Roadmap

- Latest release: 21.0;
- Available for Java 8 and Java 11;
- Predictable release schedule;
- LTS releases: last major release of the year.



What's new in GraalVM 21.0: Java on Truffle





Community Edition

GraalVM Community is available for free for evaluation, development and production use. It is built from the GraalVM sources available on GitHub. We provide pre-built binaries for Linux, macOS X, and Windows platforms on x86 64-bit systems. Windows support is experimental.

Enterprise Edition

GraalVM Enterprise provides additional performance, security, and scalability relevant for running applications in production. It is free for evaluation uses and available for download from the Oracle Technology Network. We provide binaries for Linux, macOS X, and Windows platforms on x86 64-bit systems. Windows support is experimental.

get both: graalvm.org

Java Innovations with GraalVM

High Performance 🚀

Optimize application performance with GraalVM compiler

Fast Startup ☁️

Compile your application AOT and start instantly

Polyglot 🏗️

Mix & match languages with seamless interop

Open Source 👥

See what’s inside, track features progress, contribute

Get started with GraalVM!

Download:

graalvm.org/downloads

Follow updates:

@GraalVM / #GraalVM

Get help:

graalvm.org/slack-invitation/

graalvm-users
@oss.oracle.com

Thank you!

Alina Yurenko

@alina_yurenko

Q&A

ORACLE