Native Image, AOT, or JIT

When to use in your Java Architecture

Luram Archanjo

Who am I?

Software Engineer at **Zup Innovation**

MBA in Java projects

Java and Microservice enthusiastic

Agenda

Goal

Types of Architectures

Ahead of time (AOT) compilation

Native Image

Just in time (JIT) compilation

Questions

Goal

Types of Architectures

Centralized

- Monolithic
- Data Centric

Distributed

- Microservices
- Event Driven
- Serverless

But Architecture X is better than Y


Find the balance
of your
Architecture


All Architecture
has finite
resources!

How to use less resources using Java language?

Java Module?
Microframeworks?
Quarkus?
Micronaut?

None of them! First, we need to find the Root Cause!

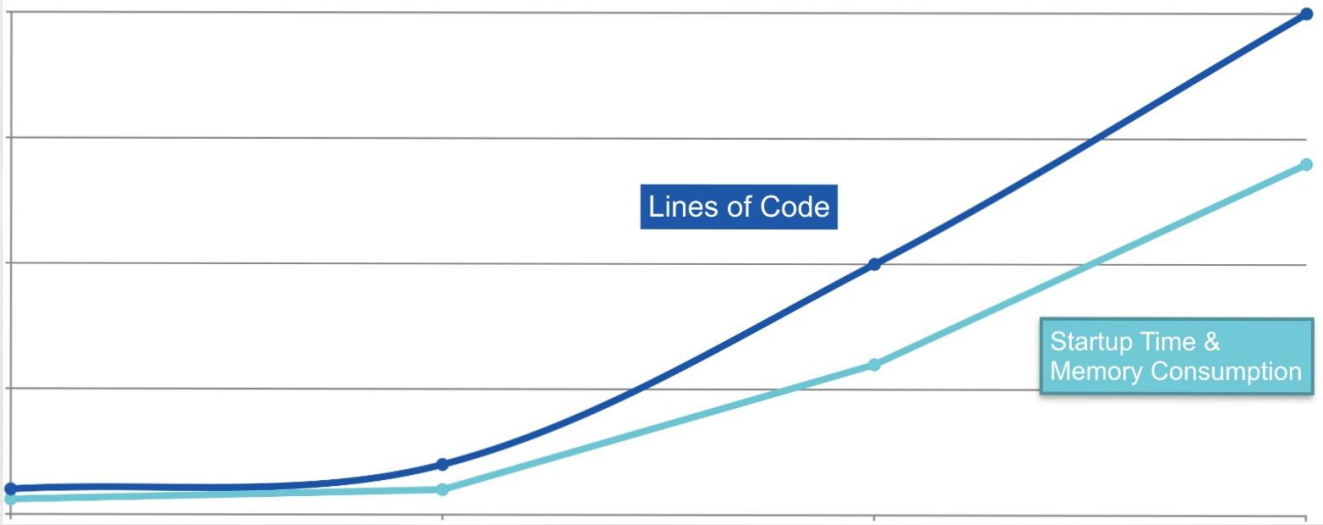

The villain of
Java's resources
is the Reflection

How Spring and Jakarta EE work?

Spring is an amazing technical achievement and does so many things, but does them at **Runtime**.

- Reads the byte code of every bean it finds.
- Synthesizes new annotations for each annotation on each bean method, constructor, field etc. to support Annotation metadata.
- Builds Reflective Metadata for each bean for every method, constructor, field etc.

The use of Reflection!



Is it possible to have the same productivity but without Reflection?


Yes, with Ahead
Of Time (AOT)
Compilation

Ahead of Time (AOT) Compilation

Ahead-of-time compilation (AOT compilation) is the act of compiling a higher-level programming language, or an intermediate representation such as Java bytecode, into a **native machine code** so that the resulting binary file can execute natively.

What are the results of using Ahead Of Time (AOT) Compilation?

OpenJDK 14 on 2019 iMac Pro Xeon 8 Core. Winner in Red.

METRIC	MICRONAUT 2.0 M2	QUARKUS 1.3.1	SPRING 2.3 M3
Compile Time ./mvn clean compile	1.48s	1.45s	1.33s
Test Time ./mvn test	4.3s	5.8s	7.2s
Start Time Dev Mode	420ms	866ms (1)	920ms
Start Time Production java -jar myjar.jar	510ms	655ms	1.24s
Time to First Response	960ms	890ms	1.85s
Requests Per Second (2)	79k req/sec	75k req/sec	??? (3)
Request Per Second -Xmx18m	50k req/sec	46k req/sec	??? (3)
Memory Consumption After Load Test (-Xmx128m) (4)	290MB	390MB	480MB
Memory Consumption After Load Test (-Xmx18m) (4)	249MB	340MB	430MB

(1) Verifier Disabled
(2) Measured with: ab -k -c 20 -n 10000 http://localhost:8080/hello/John
(3) Spring WebFlux doesn't seem to support keep alive?
(4) Measured with: ps x -o rss,vsz,command | grep java

Source: https://micronaut.io/blog/2020-04-28-performance-comparison-spring-boot-micronaut.html

What about the native machine code?

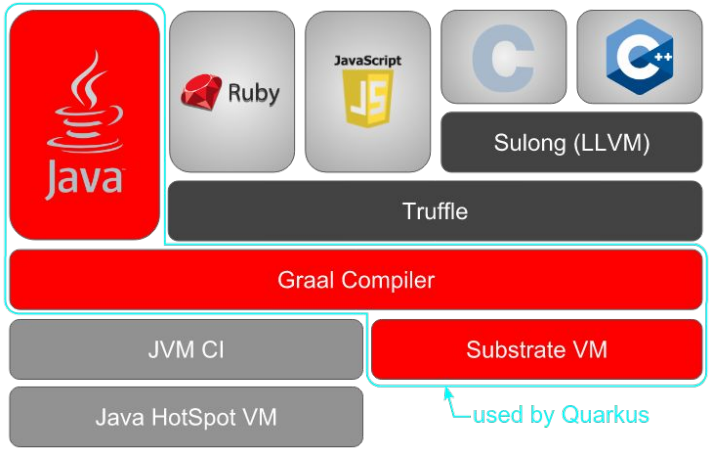

GraalVM™

GraalVM

- **Native Image**
- Ahead-of-Time Compilation
- For existing Java applications, GraalVM can provide benefits by running them faster, providing a faster **Just In Time (JIT) Compilation**

GraalVM

GraalVM Native Image **allows** you to **ahead-of-time compile** Java code to a **standalone executable**, called a **native image**.



Source: https://www.graalvm.org/docs/reference-manual/native-image/

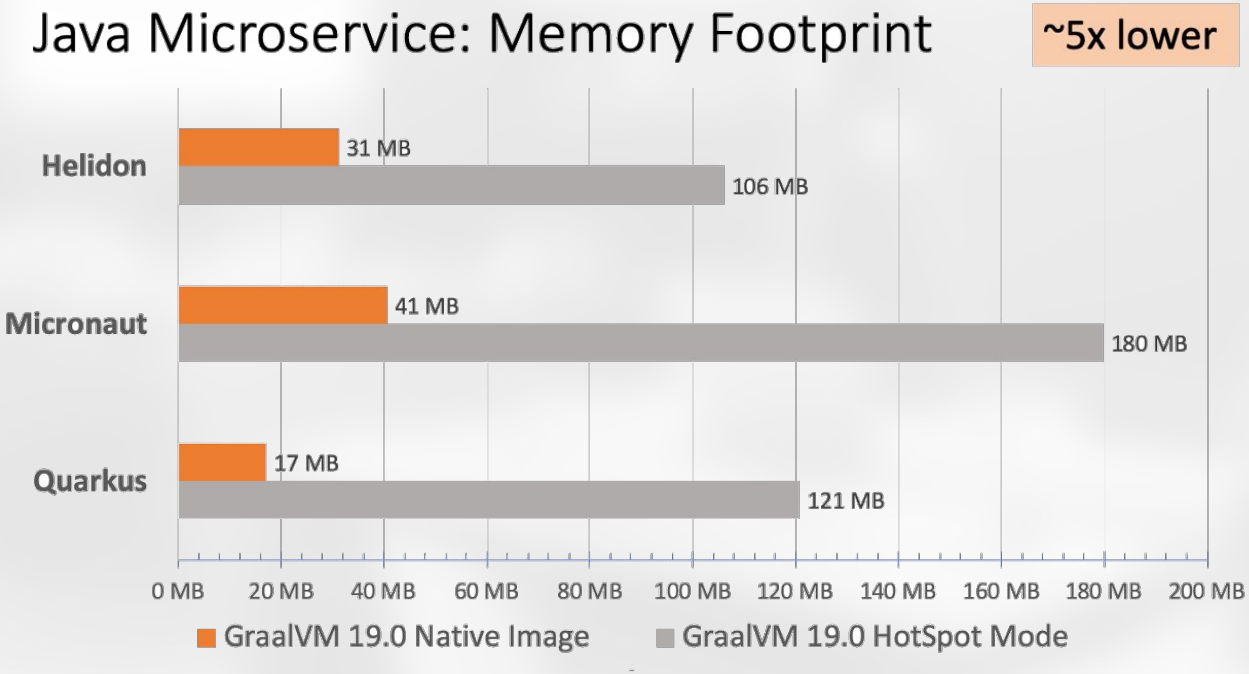
GraalVM – Limitations

Dynamic Class Loading: Deploying jars, wars, etc. at runtime impossible.

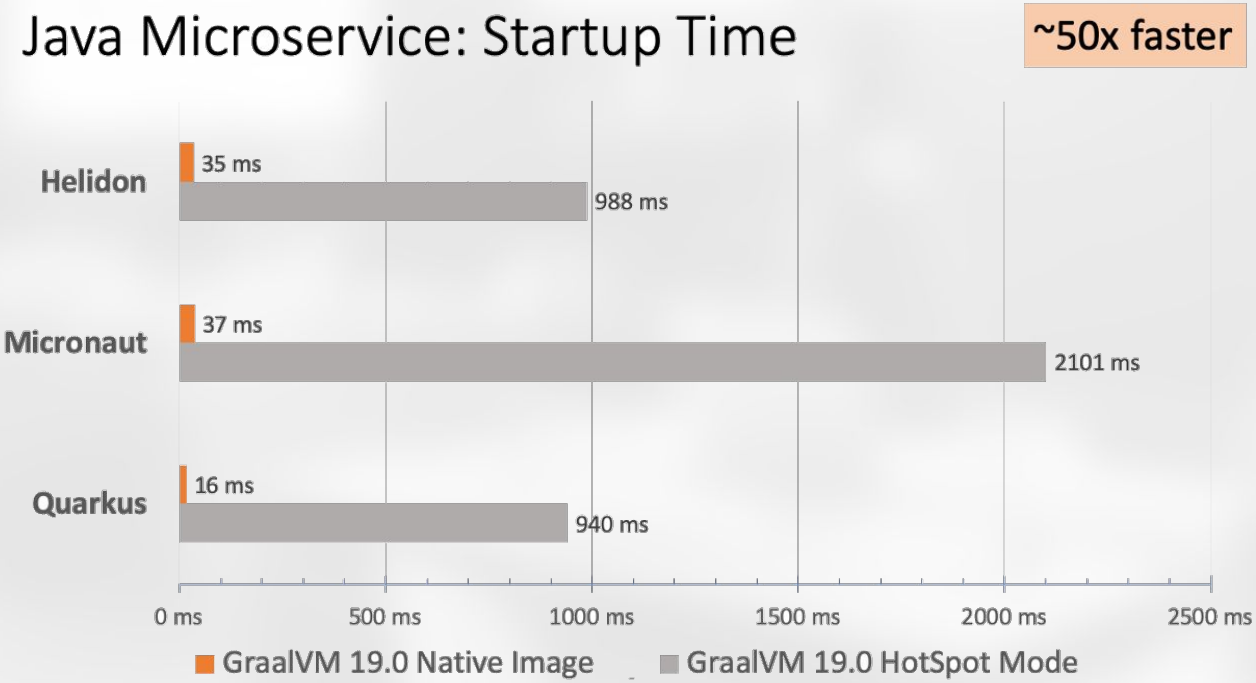
Reflection: Requires registration via native-image CLI/API.

Dynamic Proxy: No agents: JMX, JRebel, Byteman, profilers, tracers, etc.

What are the results of using Native Image?



Source: https://www.graalvm.org/docs/why-graal/



Source: https://www.graalvm.org/docs/why-graal/

Early Adopter Technology!

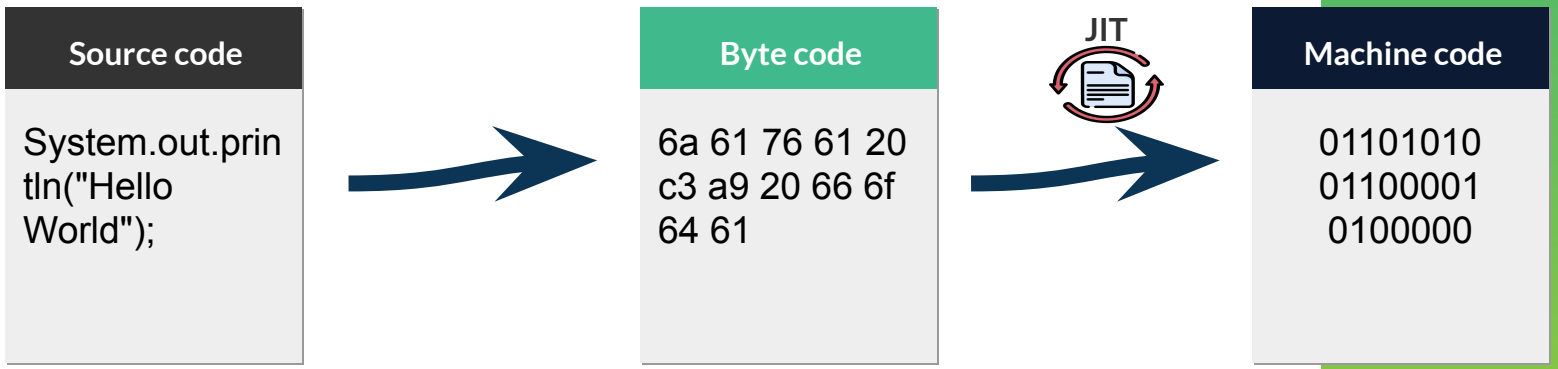
I don't use Quarkus, Micronaut and Helidon!

How can I improve my current system?

What about the Just In Time (JIT) Compilation?

Just In Time (JIT) Compilation

Just In Time (JIT) compilation is a way of executing computer code that involves **compilation during execution of a program**. It runs complex optimizations to generate high-quality machine code



What are the results of using Just In Time (JIT) Compilation?

Twitter

Every company is **constantly** looking into ways to increase **availability** of the platform while keeping an eye on costs. **Twitter** saw Oracle **GraalVM**, a language-independent compiler engine and virtual machine, and decided to try it. Average **CPU savings** for compiler innovation are in the **1–2** percent range, but using Oracle GraalVM, Twitter realized between **8 and 11** percent CPU savings, **depending** on the microservice ported.

Source: https://www.constellationr.com/research/how-oracle-graalvm-supercharged-twitter-s-microservices-platform

What are the results of using Just In Time (JIT) Compilation?

OpenJDK

Count	4589
Total	60.00 s
Slowest	3.79 s
Fastest	5.85 ms
Average	130.43 ms
Requests / sec	76.48

GraalVM™

Count	5815
Total	60.00 s
Slowest	2.36 s
Fastest	2.15 ms
Average	102.87 ms
Requests / sec	96.91

A lot of cool initiatives!

Let us recap?

Summary

2º Place

Native Image

- Low memory footprint 5x lower
- Fast Startup 50x lower
- Early Adopter Technology

1º Place

Ahead of Time (AOT) Compilation

- Low memory footprint
- Fast Startup
- IoC & SQL

3º Place

Just in Time (JIT) Compilation

- Latency
- Throughput

Native Image, AOT, or JIT?

When to use in your Java Architecture?

Native Image, AOT, or JIT? When to use in your Java Architecture?

Existent Application

Just In Time Compilation

- GraalVM
  - Java 8
  - Java 11
- Community
- Enterprise

New Application

Ahead Of Time Compilation

- Spring*
- Quarkus
- Micronaut
- Helidon

Serverless & CLI

Native Image

- GraalVM

Java is dying?

Thanks a million!

Questions?

/larchanjo

/luram-archanjo