Java without JVM
A Change for the Better?

OJUG
February 26, 2026

By Aaron Grothe

Introduction

If you have questions/comments please feel free to ask them anytime. You don't have to hold them until the end of the talk.

If there are other resources similar to these that you think might be useful to people please let the group know.

I'm still figuring this stuff out and I might know the answer :-(

Hopefully this will be an interactive and productive session.

Slides will be at https://grothe.us

JVM - Survey

Quick question. How many of your are running one of the following:

- Java's JDK?
- OpenJDK?
- Azul?
- Cornetto - Amazon's packaging of the OpenJDK for Microsoft Windows/Linux/macOS?
- Oracle Graalvm?
- GraalVM CE (Community Edition)?
- Eclipse OpenJ9?
- Other?

Java and JVM

Which came first Java or the JVM?

Java as the programming language Oak, predates the JVM.

It is the JVM that made Java's acceptance happen

Similar to the way that C and Unix working together led to the eventual rise of Unix -> Linux -> Microsoft Windows :-(

Other Languages on the JVM

There are other languages that run on the JVM

- Kotlin - popular for Android development
- Groovy - scripting java, can be combined with Java
- Scala - the language the "Cool kids" keep talking about
- Some other languages run on top of the JVM as well
  - Jython, JRuby, Rhino (Javascript), JPHP (PHP)

Some of these such as Jython and Groovy interact quite well with Java objects, classes and procedures

GraalVM

We'll mostly be talking about using GraalVM as an alternative to the Java JVM

GraalVM has three modes

GraalVM JIT - drop in replacement for Java JVM
GraalVM Native Image - builds a single executable from a Java app, ideal for containers/microservices
GraalVM WASM - compiles to Web Assembly theoretically :-(

We'll mostly be talking about GraalVM Native Image

GraalVM CE vs Oracle GraalVM

Need to clarify the differences between these two projects

GraalVM Community Edition (CE): GPLv2 with ClassPath exception. Built on top of OpenJDK free for any use, forever

Oracle GraalVM: free to use under GraalVM Free Terms and conditions, including for commercial use as long as you stay on supported versions (JDK 17, 21, 23, 25)

GraalVM vs Oracle GraalVM

Some of the benefits of the Oracle GraalVM

- PGO (Profile-Guided Optimization)
- Better garbage collection with G1 Garbage Collector
- Better inlining
- Full SIMD (for Math/AI)
- Native Image size is compressed with Oracle GraalVm

Oracle GraalVM is basically GraalVM CE but “turbo” mode

Why Java w/o JVM?

What are a couple of the biggest problems with Java/JVM?

- "Cold Start" Problem
- Large Memory Footprint
- "Warm-up" Delay
- Packaging

Each of these is worth a slide or two

We'll compare these to the GraalVM (Native image)

JVM - Cold Start Problem

JVMs are known for slow startup - JVM does the following

- Start the VM
- Load/Verify thousands of classes
- Interpret code - until JIT compiles it
- Run JIT compiler to compile byte-code to machine code

Project Leyden is a project that fires up the JVM loads the app and then snapshots the application - reducing startup time

AWS has Lambda Quickstart similar to Project Leyden

GraalVM - Cold Start Problem

GraalVM: Native Image (AOT) - self contained executable

Startup time ideally goes from seconds to milliseconds

JVM - Large Memory Footprint

The JVM has never said "I have enough memory"

Everybody who has programmed Java or run a compiled Java application has come across the classic "fix" to any problem

E.g. I'm having an issue with -Xmx2g -Xms512m
Response: "have you tried -Xmx4g - Xms1g?"

And the amazing thing is that this works a lot of the time

Works well enough for an application, but for a service can be a killer

GraalVM - Large Memory Footprint

Native Image Mode

- Only includes code that is actually reachable
- Binary doesn't need JIT compiler or metadata

JIT - mode

- Graal compiler is written in Java, more aggressive garbage collection

JVM - "Warm-Up" Delay

Maximum performance of a Java program happens after running for some time

- JIT compiler needs time to monitor code and apply optimizations (such as inlining)

GraalVM - "Warm-Up" Delay

Native image is pre-compiled.

- You get maximum performance from the getgo
- No "Warm-Up" period

JVM - Packaging

For typical Java application you need to distribute the following

- JAR+JVM

This can be a pretty good size package as you're potentially including some classes you don't need

GraalVM - Packaging

GraalVM Native Image

- Single executable

GraalVM Limitations

GraalVM is a cool piece of software but has a few issues in Native Image mode

- Compile Times
- Closed World Restriction
- Peak Performance
- Debugging Issues

Each of these is worth a slide or two

GraalVM - Compile Times

GraalVM - Native mode takes a lot longer to create an executable

Extremely resource intensive to create an image

- Applications that take seconds to compile for a regular JVM can take minutes to compile
- It has to go through examine all paths and libraries to figure out what it needs to include

This can have a real impact on CI/CD pipelines and processes

GraalVM - Closed World Restriction

All code needs to be know at build time.

This makes some of these features a bit more difficult

- Reflection
- Dynamic Proxies
- Serialization

You may have to provide configuration files to have GraalVM use dynamic features

GraalVM - Peak Performance

Over time a JVM with JIT compilation will improve the performance of an application

GraalVM's native compilation provides maximum performance from the beginning of the run

GraalVM - Debugging Issues

Since you've got an executable you can't use regular Java debugging tools in your IDE

You instead have to use native debuggers such as GDB or LLDB

Other Options

GraalVM is just one solution. There are a lot of others

- Azul Prime - Azul has a commercial drop in replacement for the JVM (Zing). Provides more control over Garbage Collection and other options
- Eclipse OpenJ9 - designed to be faster JVM uses shared libraries to reduce memory overhead
- Mandrel - stripped down version of GraalVM maintained by Red Hat for Quarkus framework - Linux executables only
- Temurin - Community backed HotSpot derived JVM

Demos

Couple of Simple Demos

Spring Boot - Hello World

- Run with Java JDK
- Run with GraalVM
- Compile to Native Image and Run

Regular Hello World

- Compile to WASM and Run (GraalVM Oracle edition)

Spring Boot - Hello World

Consists of Three files

- pom.xml - Maven Project Object Model
- DemoApplication.java - entry point for program
- HelloController.java - endpoint

pom.xml (part 1 of 3)

```
<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.example</groupId>
  <artifactId>demo-graalvm-wasm</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>3.2.2</version> </parent>
```

pom.xml (part 2 of 3)

```
<properties>
    <java.version>21</java.version>
  </properties>

  <dependencies>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
  </dependencies>
```

pom.xml (part 3 of 3)

```
<build>
    <plugins>
        <plugin>
            <groupId>org.graalvm.buildtools</groupId>
            <artifactId>native-maven-plugin</artifactId>
        </plugin>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
        </plugin>
    </plugins>
  </build>
</project>
```

src/main/java/com/example/demo/DemoApplication.java

```
package com.example.demo;

import org.springframework.boot.SpringApplication;
import
org.springframework.boot.autoconfigure.SpringBootApplicatio
n;

@SpringBootApplication
public class DemoApplication {
    public static void main(String[] args) {
        SpringApplication.run(DemoApplication.class, args);
    }
}
```

src/main/java/com/example/demo/HelloController.java

```
package com.example.demo;

import org.springframework.web.bind.annotation.GetMapping;
import
org.springframework.web.bind.annotation.RestController;

@RestController
public class HelloController {

    @GetMapping("/")
    public String hello() {
        return "Hello World from Spring Boot!";
    }
}
```

Layout of Directories

Quick bit of Information

I have the following directories on my system

- graalvmce - Community edition of graalvm
- grallvm - Oracle edition of graalvm
- openjdk - Placeholder for OpenJDK on my system

Using direnv to set JAVA_HOME and PATH, highly recommend direnv if you're going to use multiple JVMs

Run App with Regular JDK

Change to the openjdk directory

% cd ~/openjdk

Use maven to start spring boot application

% mvn spring-boot:run

Wait a couple of seconds, hit the web browser at

http://localhost:8080

Success?

Run App with GraalVM

Change to the graalvmce directory

% cd ~/graalvmce

Use maven to start spring boot application

% mvn spring-boot:run

Wait a couple of seconds, hit the web browser at

http://localhost:8080

Success?

Run App with GraalVM (Native) - part 1

Change to the graalvm directory

% cd ~/graalvm

Use maven to compile the program to a native application

% mvn -Pnative native:compile

Wait a couple of minutes - 6 minutes in my vm

Run App with GraalVM (Native) - part 2

Start the executable

./target/demo-graalvm-wasm

Wait a couple of micro seconds, hit the web browser at

http://localhost:8080

We'll stop it and take a look at the executable now (ls, file, etc)

Run HelloWorld with GraalVM (WASM)

Caveats

Why not the Spring boot app?

- Spring boot has a built-in tomcat server and it is currently beyond the capability of the WASM environment

Why are we doing this in the Oracle GraalVM Environement?

- This is new capability and is working its way into Community Edition

Run HelloWorld with GraalVM (WASM)

HelloWasm.java

```
public class HelloWasm {
    public static void main(String[] args) {
        System.out.println("Hello from GraalVM WebAssembly!");
    }
}
```

Run HelloWorld with GraalVM (WASM)

Compile to class file

% javac HelloWasm.java

Compile to WASM

% native-image --tool:svm-wasm HelloWasm

Run it

% node --experimental-wasm-exnref hellowasm.js

Success??

Run HelloWorld with GraalVM (WASM)

Now let's run it in a web browser

For this we'll need to do the following

- Create an html file to host the wasm file
- Run a local webserver
- Access in a local browser at localhost:8080

Run HelloWorld with GraalVM (WASM)

Creating an html file (index.html)

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>GraalVM Wasm Test</title>
</head>
<body>
    <h1>Java WebAssembly Test</h1>
    <p>Press <strong>F12</strong> to open the Developer Tools, then click the
<strong>Console</strong> tab to see your Java output!</p>

    <script src="hellowasm.js"></script>
</body>
</html>
```

Run HelloWorld with GraalVM (WASM)

Starting up a web server

We'll just run a simple server with python

% python3 -m http.server 8000

This will start serving the web page

Run HelloWorld with GraalVM (WASM)

Hitting the web page

Open web browser and go to localhost:8000

- We'll have to hit f12 to open Browser's Developer's tool
- Then we'll go to the console tab

Do we see "Hello from GraalVM WebAssembly!"

Success?

Demo Summary

So what did these demos show?

- OpenJDK demo - basic spring boot app
- GraalVM demo - GraalVM as drop in for OpenJDK
- GraalVM native demo - native application generation with GraalVM
- GraalVM WASM HelloWorld - potential of WASM in the future

Gives you a base idea of some of the possibilities

Tips - 6 Tips for a Better Experience

1. Start with GraalVM as a drop in replacement for the JVM. You'll probably be happy with the performance
2. Create a couple of very simple applications and compile them to Native images
3. Watch a couple of Youtube videos, there is a lot of great information about using Graalvm and generating native images - "graalvm legacy native"
4. GraalVM has an active Github page and other resources in the community
5. Try out the options: Try out GraalVM, OpenJDK, Cornetto, Eclipse OpenJ9, Mandrel, etc.
6. WASM is interesting, but very frustrating at the moment not production ready

Summary

GraalVM is a pretty good drop-in replacement for the Java JVM

It opens some new possibilities for deployment

- Native Images
- WASM deployment - in the future
- Truffle mixing languages - GraalPy, GraalJS, and TruffleRuby

It and other alternatives to the classic JVM are worth evaluating

Thanks

Thanks for Listening

Any Questions?

Links

GraalVM Homepage

- https://www.graalvm.org/

Project Leyden

- https://openjdk.org/projects/leyden/

Links

The Register article about Nearly 3 out of 4 Oracle Java users say they've been audited in the last 3 years

- https://www.theregister.com/2025/07/15/oracle_java_users_audited

The Register article about high-performance JVM alternatives

- https://www.theregister.com/2026/02/04/why_highperformance_java_becoming

Links

Good Youtube Video on Spring Boot Apps and Native Image

- https://www.youtube.com/watch?v=FjRBHKUP-NA
- https://www.youtube.com/watch?v=ruFTMi-SPHY