Ahead of time: going native

Native builds with SpringBoot 3

Johan Hutting

JCON - June 20th, 2023

do your thing

Introduction

With Spring 6 and SpringBoot 3 available – should you go native with your application?

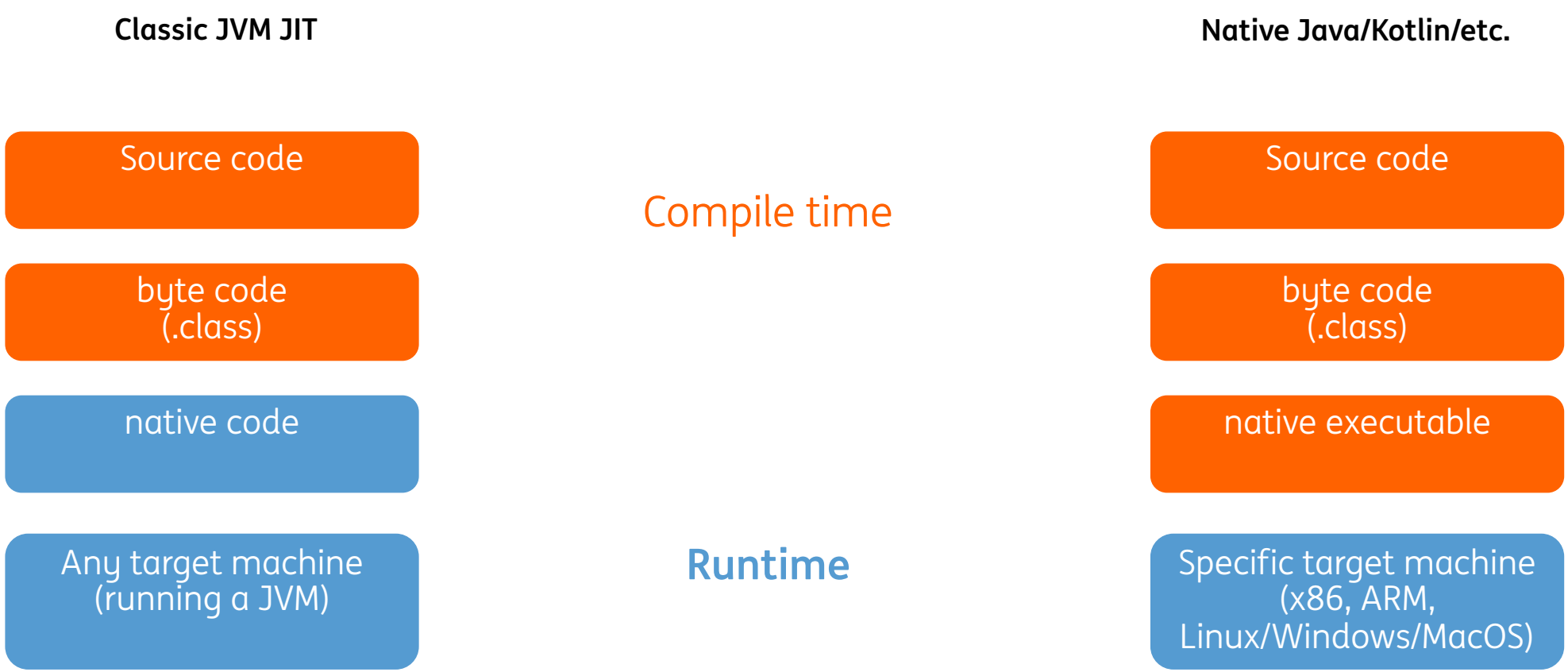
- Introduction to native frameworks and GraalVM
- Spring's native offering, benefits & considerations
- configuration options and how to apply those
- Live demo

% whoami

Senior software engineer and Subject Matter Expert at ING.

Strong focus on efficient solutions for real world challenges as well as sharing knowledge to empower those around me.

JIT versus AOT



Native frameworks and GraalVM

Current native frameworks (Q2 2023):

- Quarkus.io Redhat's Microprofile/EE-based offering with developer mode
- Micronaut Microprofile/EE-based offering by the Grails team
- Helidon.io Microprofile/EE-based, main support by Oracle
- Spring Boot 3/Spring 6

GraalVM is the glue between the framework and the native executable.

Spring Native

A component to assist you in developing native Spring applications.

- Takes care of most GraalVM config to recognize and load Spring Beans
- Annotations for those cases that were left out

Starting with SpringBoot 3 / Spring 6 “Spring Native” is now part of Spring itself.

Adding resources

Resources need to be defined before you can use them in a native application – after all, they need to be packaged inside the binary.

These can be declared using:

- Individually using a graalVM parameter (not prefered)
- Add them to resource-config.json
- Spring 6 RuntimeHints in code

Adding types

Like resources GraalVM needs to know about types at compile-time, even though it manages to find most information by itself.

These can be declared using:

- Individually using a graalVM parameter (not prefered)
- Add them to reflect-config.json and serialization-config.json
- Spring 6: @TypeHint annotation or @RegisterReflectionForBinding(Pojo.class)

Other helpers

Next to resources and serialization/reflection we also have helpers for:

- Java Proxies – proxy-config.json
- JNI – jni-config.json or @CEntryPoint

For details: https://www.graalvm.org/latest/reference-manual/native-image/ and https://docs.spring.io/spring-boot/docs/current/reference/html/native-image.html

What about libraries?

Many modern libraries already offer the resource and type definitions.
Overview: https://www.graalvm.org/native-image/libraries-and-frameworks/

For internal libraries you should take care of this yourself by generating them during the build.
To generate traces:

```
$ java -Dspring.aot.enabled=true \
    -agentlib:native-image-agent=config-output-dir=/path/to/config-dir/ \
    -jar target/myproject-0.0.1-SNAPSHOT.jar
```

(Or add these as jvmArguments to your spring-boot plugin)

External unsupported library? Join this OS initiative: https://github.com/oracle/graalvm-reachability-metadata

Live Demo

Performance tips

Native applications offer very fast performance at start-up, but over time the JIT version adapts and overcomes.

Consider applying “Profile Guided Optimisations” for longer running applications by generating a profile.

Instructions: https://www.graalvm.org/22.0/reference-manual/native-image/PGO/

Takeaways

Recap

Native AoT version:

- Very suitable for lambda applications
- Better efficiency in resources
- No runtime optimisation
- Fewer options to debug
- (Much) longer build times

”Classic” JVM JIT version:

- Better long-running performance
- Offers all the good and bad things you’ve been using the past years

Experiment and validate which one suits your solutions best.


Questions?

https://www.ing.de

do your thing

Sheets available at https://github.com/jhutting/Talks/