# GraalVM and Spring Native

Aleksandr Barmin

April 2022

# Aleksandr Barmin

- Chief Software Engineer I
- Write Java since 2010
- AWS Solution Architect Associate
- AWS Developer Associate

Aleksandr_Barmin@epam.com

# Agenda

- What is GraalVM
- Building native images with GraalVM
- Spring Native - all you need to start with
- `@NativeHint` from Spring Native
- Code examples, not so many slides

# What is GraalVM?

- High performance JDK distribution
- Core distribution:
  - JVM
  - GraalVM Compiler
  - LLVM and JS runtimes
- Supports Java, JS, Ruby, Python, R, WebAssembly, multiple langs at once

# Getting Started

- Use SDKMan to install GraalVM
- Write a simple Java app, build, run, enjoy

# GraalVM Native Image

- AOT compile Java to a standalone executable - platform specific
- Executable includes all the code was available at **image build time**
- Substrate VM provides memory management, GC, thread scheduling, etc ordinary JVM features
- `$ gu install native-image`

# Closed World Optimizations

- Dynamic Class Loading
- Reflection
- Dynamic Proxy
- Java Native Interface - JNI
- Serialization

# Incompatible with Closed-World Optimizations

- `invokedynamic` Bytecode and Method Handles
- Security Manager

# Features that may operate differently in Native Images

- Signal Handlers
- Class Initializers
- Finalizers
- Thread methods like `Thread.stop()`
- Unsafe Memory Access
- Debugging and Monitoring

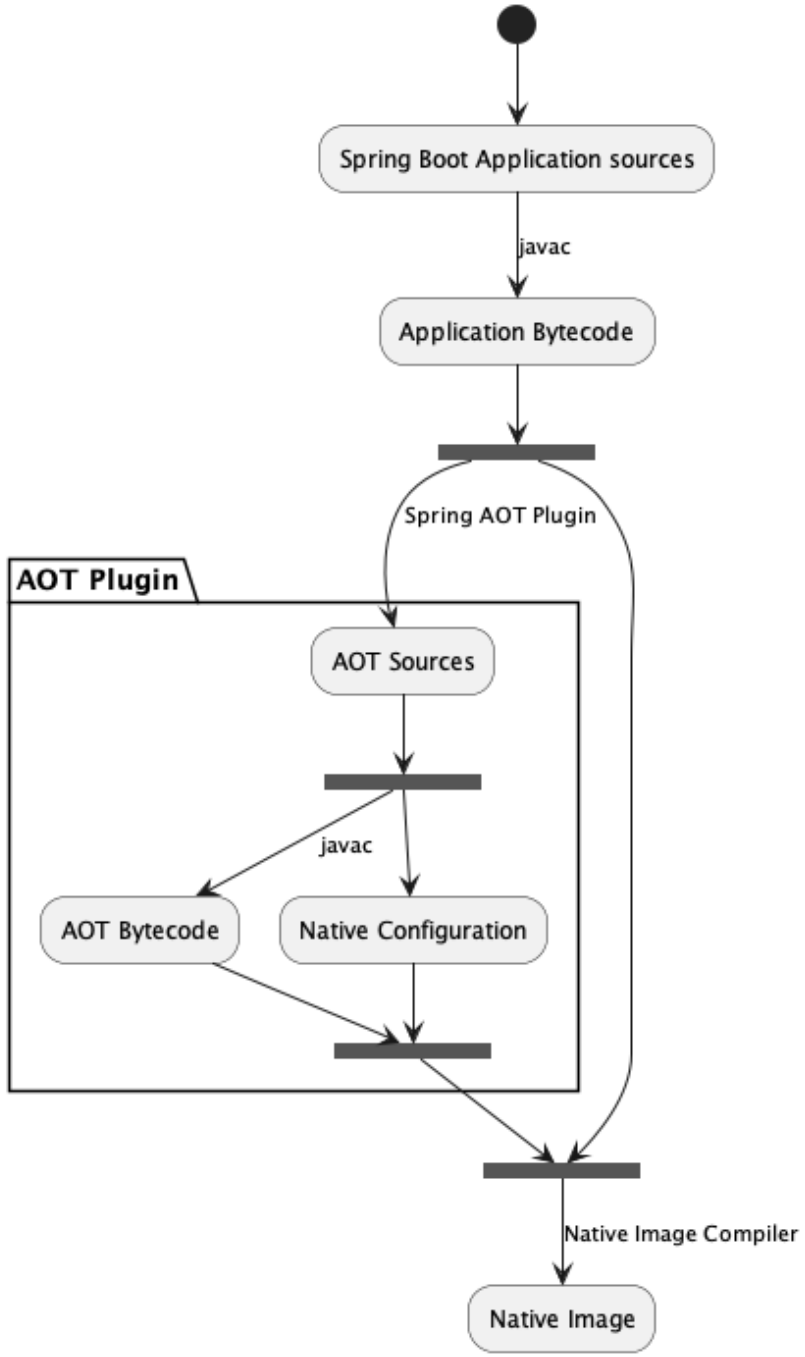
# Spring Native

Spring Native provides support for compiling Spring-based apps using GraalVM native image compiler

- spring-native
- spring-native-configuration
- spring-native-docs
- spring-native-tools
- spring-aot
- spring-aot-test
- spring-aot-maven/gradle-plugin

# Spring AOP

- Prepares `BeanFactory` in order not to configure it during runtime (read `@Configuration` classes, post-processing, etc).
- Generates code based on the configured `BeanFactory`.
- Additional code generation, ex. creation of `reflect-config.json`.



# Bean Factory Preparations

- Computation of `@ConditionalOn`-s.
- Profiles evaluation.
- Invoke any `BeanDefinitionRegistryPostProcessor` to create beans declared at runtime.

# Code Generation

Based on `RootBeanDefinition` Spring AOT finds a suitable `BeanRegistrationWriter` that will write a code that is responsible for initialization of a bean at runtime.

# Native Hints

- `@TypeHint` - for simple reflection
- `@NativeHint` - more complicated cases like field or method-level access
- `BeanFactoryNativeConfigurationProcessor` - good for bean introspection
- `BeanNativeConfigurationProcessor` - to inspect `BeanInstanceDescription` of every bean
- `NativeConfiguration` for hints not related to beans or `BeanFactory`

# Summary

- GraalVM provides significant performance boost by converting existing code
- Still requires additional meta information due to the closed world assumption
- Spring Native allows to convert existing Spring app into native image with less efforts

https://github.com/aabarmin/epam-spring-native-example-2022