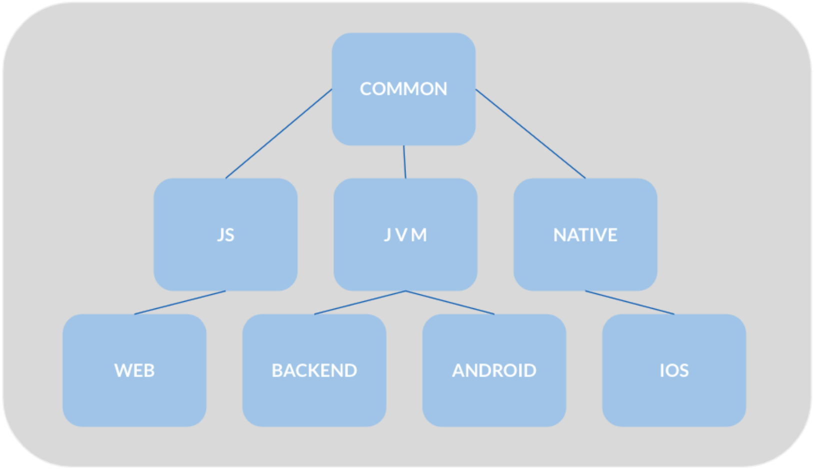
Compilação nativa com Kotlin e GraalVM

Víctor Orozco

10 de Junho de 2021

Nabenik

Ecossistema Kotlin



Kotlin Native

Kotlin Native

1. Código Kotlin + Kotlin stdlib + Bibliotecas ”Kotlin puro”
2. Bytecode LLVM
3. Bibliotecas do sistema -e.g. Cocoa -

Kotlin GraalVM Native

1. Código Kotlin + Kotlin stdlib + **Bibliotecas JVM**
2. Aplicativos ELF / Mach-O com GCC
3. LLVM backend
4. SubstrateVM

GraalVM

¿GraalVM?

- Maquina virtual poliglota da Oracle Labs
- JVM, Truffle, LLVM
- Escrita em Java
- Open Source e Enterprise Edition

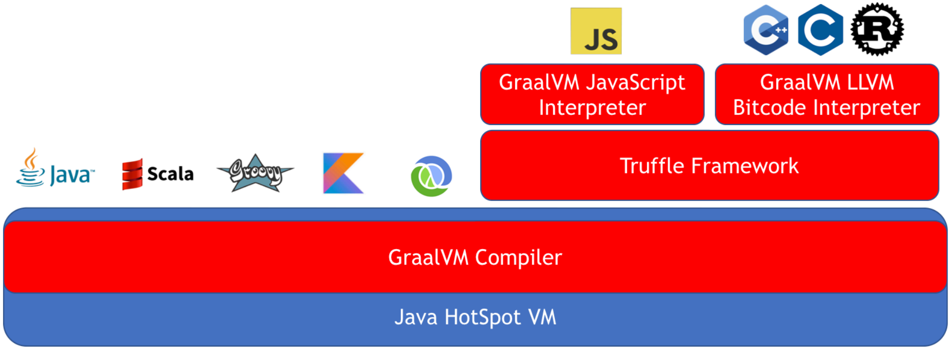


Figura 1: GraalVM Overview

GraalVM

Fatos importantes

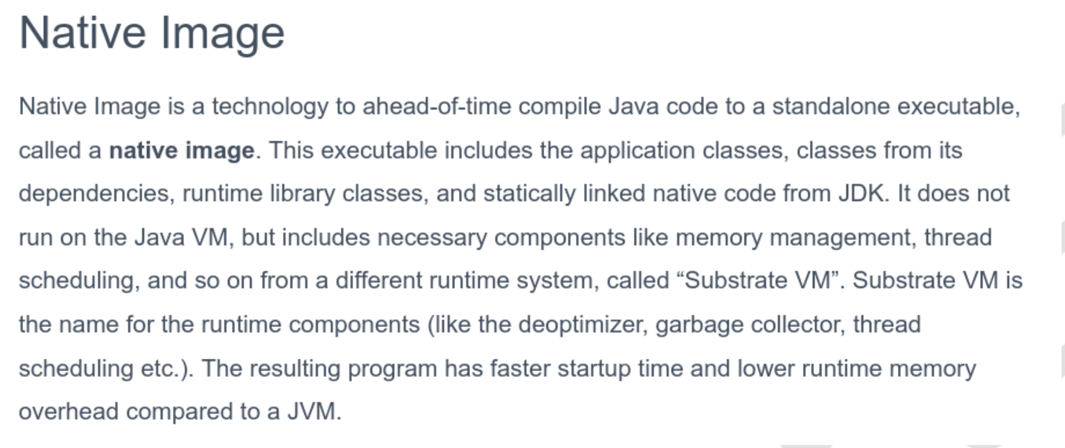
1. TCK'd JDK
2. Compilador JIT
3. **Java Native Image**
4. Polyglot VM

Imagens nativas

Java Virtual Machine

- Thread scheduling, gestão de memoria
- JVM JIT (C2) tem 25 anos
- Peak performance
- Hotspots

GraalVM Native



Native Image

Native Image is a technology to ahead-of-time compile Java code to a standalone executable, called a **native image**. This executable includes the application classes, classes from its dependencies, runtime library classes, and statically linked native code from JDK. It does not run on the Java VM, but includes necessary components like memory management, thread scheduling, and so on from a different runtime system, called “Substrate VM”. Substrate VM is the name for the runtime components (like the deoptimizer, garbage collector, thread scheduling etc.). The resulting program has faster startup time and lower runtime memory overhead compared to a JVM.

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

GraalVM Native

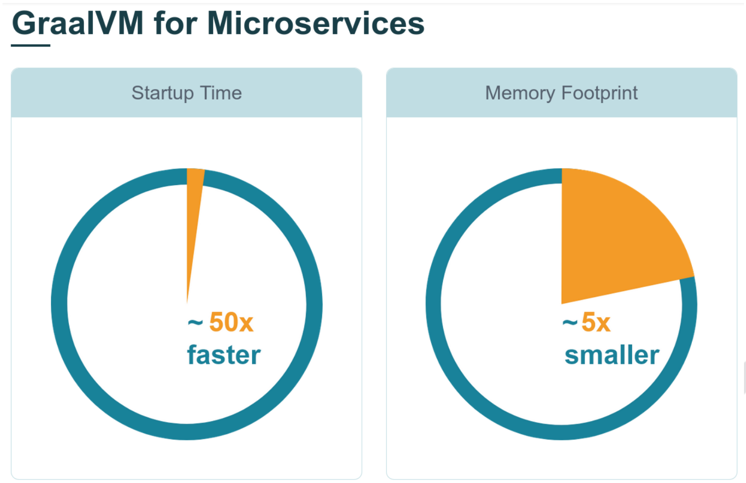
GraalVM Native

GraalVM Native é uma tecnologia para **compilación AOT do bytecode**. Permite criar um executável ”self-contained”com **static linking** de classes, bibliotecas e módulos da JVM.

AOT no mundo da JVM

- ExcelsiorJET
- GNU Compiler for Java
- ART (Android)
- **IBM OpenJ9**

GraalVM Native



GraalVM Native

- **CLI**
- Apps desktop
- Serverless
- **Microsserviçõs**
- Kubernetes operators

GraalVM Native

picocli (Anotações)

```
@CommandLine.Command(name = "kobsidian-backup",
mixinStandardHelpOptions = true,
version = ["kobsidian-backup␣1.0.10"],
description = ["Creates␣backups␣from␣Postgres␣and␣uploads␣these␣to␣Dropbox"])
class BackupOptions{


    @CommandLine.Option(names = ["-d", "--database"],
        paramLabel = "DATABASE␣NAME",
        description = ["Database␣target␣for␣backup␣actions"]
    )
    var databaseName: String?
```

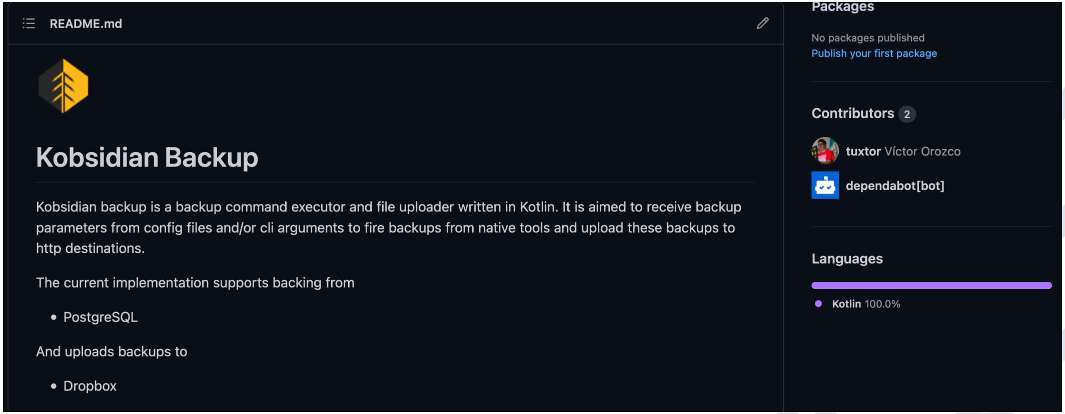
GraalVM Native

clikt (Kotlin DSL)

```
class Hello : CliktCommand() {
    val count: Int by option(help="Number␣of␣greetings").int().default(1)
    val name: String by option(help="The␣person␣to␣greet").prompt("Your␣name")

    override fun run() {
        repeat(count) {
            echo("Hello␣$name!")
        }
    }
}
```

Kobsidian Backups



README.md

Kobsidian Backup

Kobsidian backup is a backup command executor and file uploader written in Kotlin. It is aimed to receive backup parameters from config files and/or cli arguments to fire backups from native tools and upload these backups to http destinations.

The current implementation supports backing from

- PostgreSQL

And uploads backups to

- Dropbox

Packages

No packages published
Publish your first package

Contributors 2

tuxtor Víctor Orozco
dependabot[bot]

Languages

● Kotlin 100.0%

Demo

1. Maven Quickstart (Java)
2. Kotlin stdlib
3. PicoCLI
4. GraalVM Native

Considerações finais

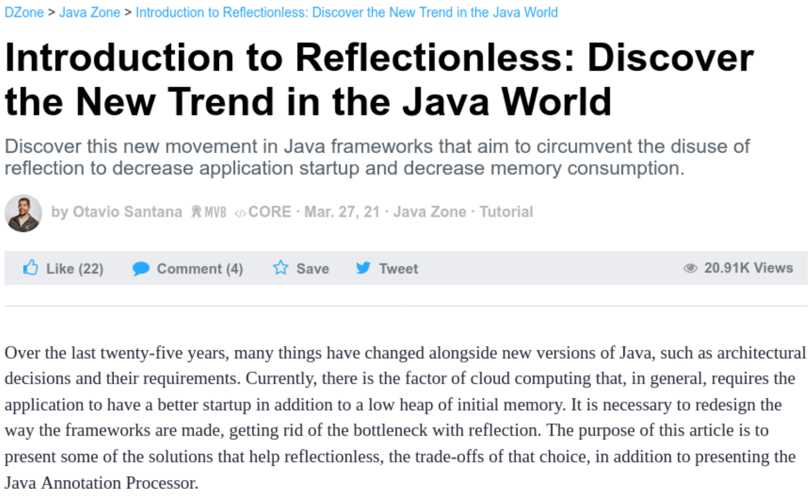
Considerações finais

Vantagens

- Compilação AOT
- Consumo de memoria
- Tempo de startup
- CLI, Desktop, Serverless, K8S

Desvantagens

- Desempenho inferior no longo prazo
- Reflection, dynamic proxies, invoke, bytecode generation
- Muitos frameworks e biblitecas jamais serão native
- É preciso um bom CI/CD



DZone > Java Zone > Introduction to Reflectionless: Discover the New Trend in the Java World

Introduction to Reflectionless: Discover the New Trend in the Java World

Discover this new movement in Java frameworks that aim to circumvent the disuse of reflection to decrease application startup and decrease memory consumption.

by Otavio Santana MVB CORE · Mar. 27, 21 · Java Zone · Tutorial

Over the last twenty-five years, many things have changed alongside new versions of Java, such as architectural decisions and their requirements. Currently, there is the factor of cloud computing that, in general, requires the application to have a better startup in addition to a low heap of initial memory. It is necessary to redesign the way the frameworks are made, getting rid of the bottleneck with reflection. The purpose of this article is to present some of the solutions that help reflectionless, the trade-offs of that choice, in addition to presenting the Java Annotation Processor.

https://dzone.com/articles/introduction-to-reflectionless-know-what-the-new-t

Víctor Orozco

- vorozco@nabenik.com
- @tuxtor
- http://vorozco.com
- http://tuxtor.shekalug.org

Duke's Choice Award

2016 Winner

Oracle Groundbreakers

Java Champions

ORACLE Certified Professional Java SE 8 Programmer

ORACLE Certified Associate Java SE 8 Programmer