

Compilación nativa con Kotlin y GraalVM

Víctor Orozco

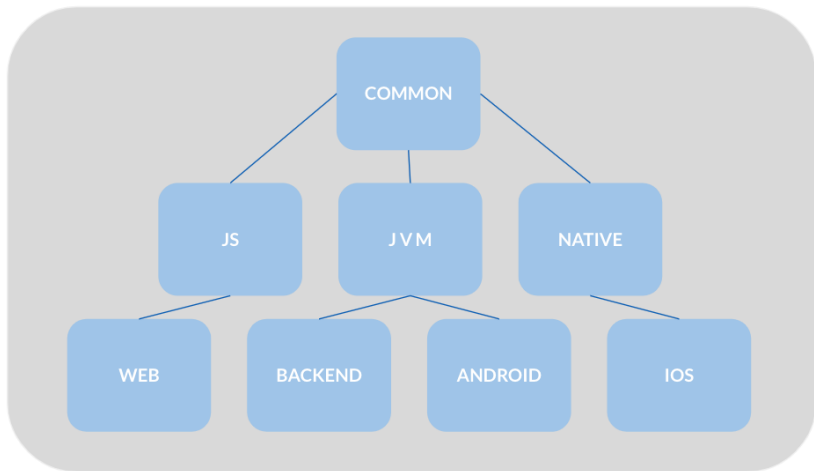
17 de diciembre de 2021

Nabenik



NABENIK

Ecosistema Kotlin



Kotlin Native

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

Kotlin GraalVM Native

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

¿GraalVM?

- Máquina virtual poliglota de Oracle Labs
- JVM, Truffle, LLVM
- Escrita en Java
- Open Source y Enterprise Edition

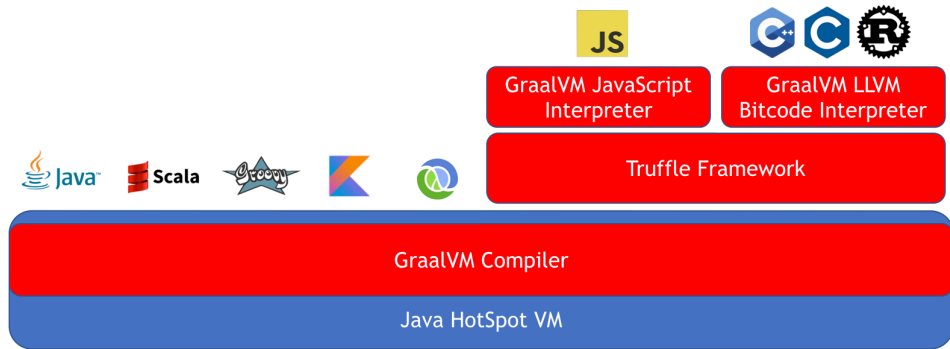


Figura 1: GraalVM Overview

Hechos importantes

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



The logo for GraalVM, with 'Graal' in blue and 'VM' in orange, followed by a small 'TM' trademark symbol. To the right of the logo, there are three light gray decorative lines that originate from circular nodes and extend diagonally across the slide.

GraalVMTM



Imágenes nativas



Java Virtual Machine

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

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.

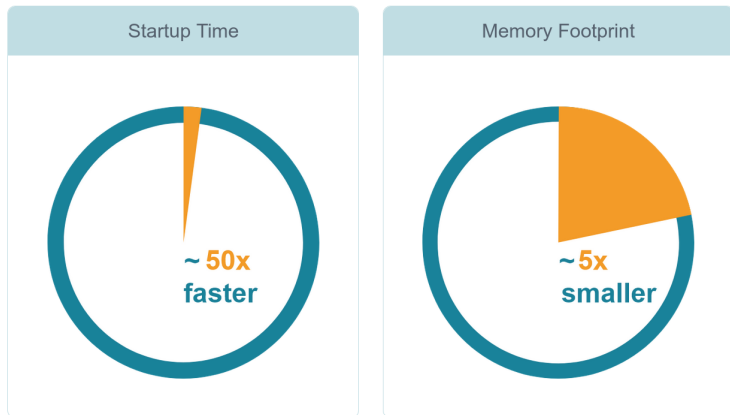
GraalVM Native

GraalVM Native es una tecnología para **compilación AOT de bytecode**. Permite crear un ejecutable **self-contained** con **static linking** de clases, bibliotecas y módulos de la JVM.

AOT en el mundo de la JVM

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

GraalVM for Microservices



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


picocli (Anotaciones)

```
@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?
```

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



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%

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



Consideraciones finales



Ventajas

- Compilación AOT
- Consumo de memoria
- Tiempo de inicio
- CLI, Desktop, Serverless, K8S

Desventajas

- Desempeño inferior en el largo plazo
- Reflection, dynamic proxies, invoke, bytecode generation
- Muchos frameworks jamas seran Cloud Native
- Un buen servidor CI/CD

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



Like (22)



Comment (4)



Save



Tweet



20.91K Views

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>



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



This work is licensed under
Creative Commons Attribution-
NonCommercial-ShareAlike 3.0
Guatemala (CC BY-NC-SA 3.0 GT).