

Serverless ⚡ Spring

with GraalVM Native Support

김보배

Serverless ?

Serverless ?

= 서버 + 없는 ?

우리의 밥벌이…?

Serverless ?



서버가 없다면…?

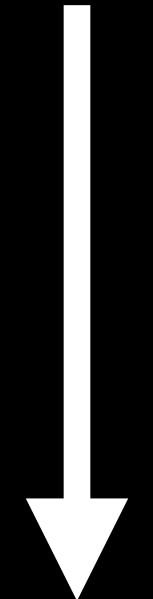
- 누가 요청을 받을건데?
- 요청을 안받으면 어떻게 유저를 엮어주지?
- 데이터베이스는?

Serverless ?

이런건 아니고...
서버가 없는 것처럼 관리할 수 있다 !

이런건 아니고…
서버가 없는 것처럼 운영할 수 있다!
== Server (운영) + Less (없다)

핵심은 “항상”



서버가 항상 구동되고 있는가?

Serverless X

서버가 요청이 들어올 때만 구동되는가? **Serverless O**

Serverless ?



내 앞에 계속 있는 종업원

벨을 누르면 찾아오는 종업원

Serverless ?

Server가 요청이 있을 때만 구동된다는 것 → 이게 뭐가 좋은 걸까 ?

서버 운영 ↓

- 서버 운영에 신경을 써야하는 시간
 - 서버 구축, Spec, 확장 고려, 유지보수, ...
- 실제로 서버가 운영되는 시간동안 지불해야하는 비용



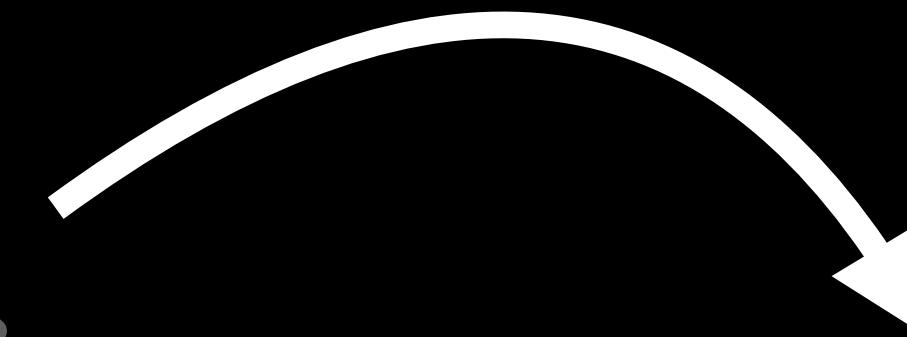
개발에만 집중 !

Serverless ?

Server가 요청이 있을 때만 구동된다는 것 → 이게 뭐가 좋은 걸까 ?

서버 운영 비용 ↓

- 서버 운영에 신경을 써야하는 시간
- 서버 구축, Spec, 확장 고려, 유지보수, ...
- 실제로 서버가 운영되는 시간동안 지불해야하는 비용



개발에만 집중 !

- 서버가 죽는 것을 고려하지 않아도 된다 😊
- Cloud 환경 ☁
- 요청 당 비용을 지불 💰

엥? 너무 좋은데?

No Silver Bullet

Serverless ?

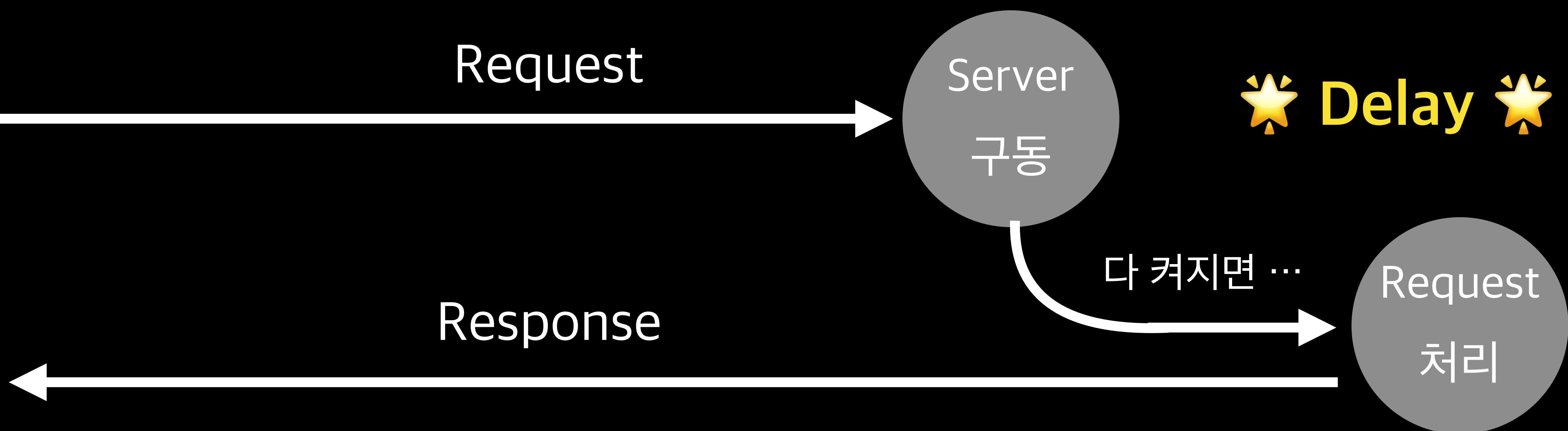


내 앞에 계속 있는 종업원

벨을 누르면 찾아오는 종업원

Serverless ?

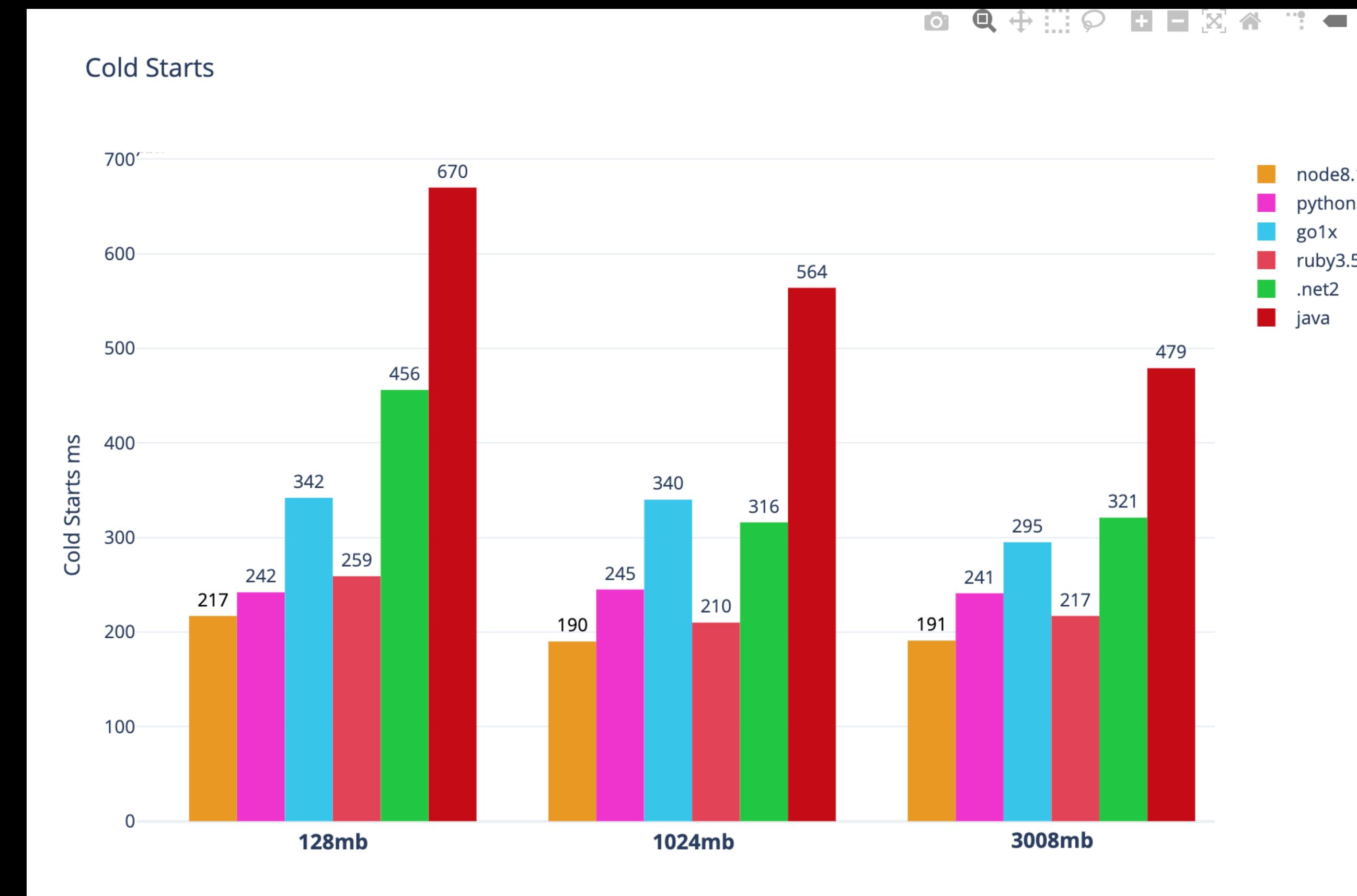
무조건 좋은 것은 아닙니다
여러 문제 중 하나인 **Cold Start**



Serverless ?

무조건 좋은 것은 아닙니다

여러 문제 중 하나인 **Cold Start**



Serverless ?



내 앞에 계속 있는 종업원

- 말하면 바로 주문 가능
- 앞에 있게 한 비용을 내야 함

벨을 누르면 찾아오는 종업원

- 벨을 눌르면 종업원 올 때까지 기다려야 함
- 부를 때마다 팁을 내야 함

AWS Lambda

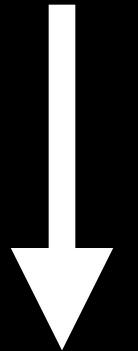
+

Spring

AWS Lambda + Spring

AWS에서 Lambda를 위한 여러 라이브러리를 지원

Spring 진영에서도 Serverless를 위한 프로젝트를 지원



이 중 spring cloud function을 사용하면 손쉽게 구현할 수 있다

** 알아둬야 할 것

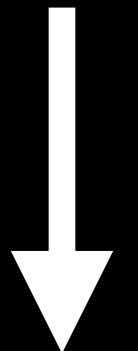
- 현재 Spring boot 3.0으로 되면서 JDK 17만 지원
- AWS Lambda 런타임에서는 JDK 11까지만 지원
- 즉, Spring boot 3.0을 사용하지 못함.

AWS Lambda + Spring

- Spring Cloud Function

Promote the **implementation of business logic via functions**

→ FaaS (Function as a Service) → Serverless



Support a uniform programming model across serverless providers, as well as the ability to run standalone

- **Support serverless platform adapters (AWS, Azure, Google Cloud Function)**

AWS Lambda + Spring

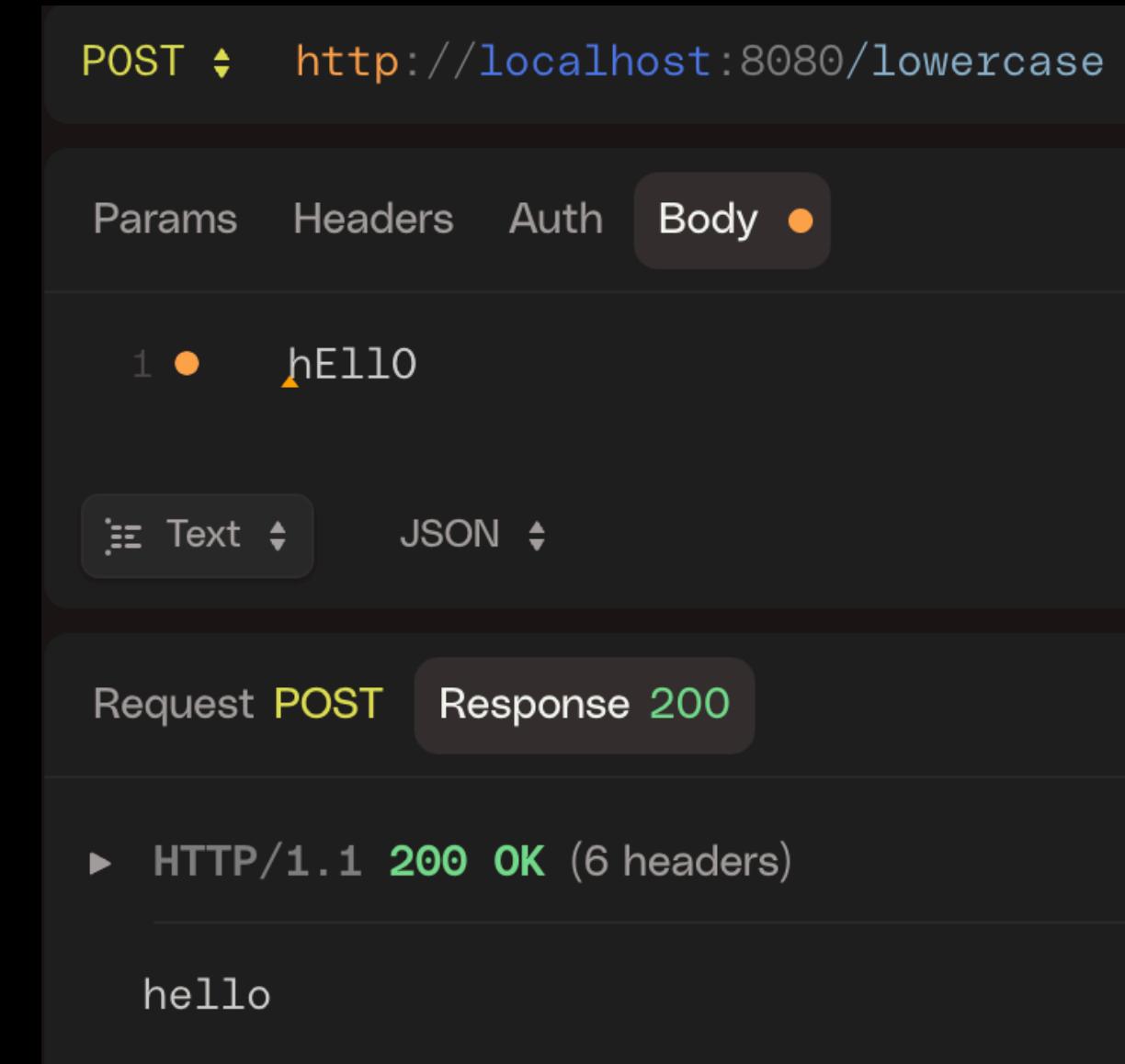
- Spring Cloud Function



```
@SpringBootApplication
public class ServerlessApplication {

    public static void main(String[] args) {
        SpringApplication.run(ServerlessApplication.class, args);
    }

    @Bean
    public Function<String, String> lowercase() {
        return request -> {
            System.out.println("** Request: " + request);
            return request.toLowerCase(Locale.ROOT);
        };
    }
}
```



FunctionInvoker가 함수에 맞게 알맞은 Bean을 찾아 실행

AWS Lambda + Spring

- Spring Cloud Function

Functional Interface (Function, Consumer, Supplier) → HTTP endpoint

Method	Path	Request	Response	Status
GET	/{supplier}	-	Items from the named supplier	200 OK
POST	/{consumer}	JSON object or text	Mirrors input and pushes request body into consumer	202 Accepted
POST	/{consumer}	JSON array or text with new lines	Mirrors input and pushes body into consumer one by one	202 Accepted
POST	/{function}	JSON object or text	The result of applying the named function	200 OK
POST	/{function}	JSON array or text with new lines	The result of applying the named function	200 OK
GET	/{function}/{item}	-	Convert the item into an object and return the result of applying the function	200 OK

AWS Lambda + Spring

- AWS Lambda 등록 시

https://docs.aws.amazon.com/ko_kr/lambda/latest/dg/java-package.html

.zip 또는 JAR 파일 아카이브를 사용하여 Java Lambda 함수 배포

[PDF](#)

AWS Lambda 함수의 코드는 스크립트 또는 컴파일된 프로그램과 해당 종속 항목으로 구성됩니다. [배포 패키지](#)를 사용하여 Lambda 함수 코드를 배포합니다. Lambda는 두 가지 유형의 배포 패키지(컨테이너 이미지 및 .zip 파일 아카이브)를 지원합니다.

이 페이지에서는 배포 패키지를 .zip 파일 또는 Jar 파일로 생성한 후 해당 배포 패키지를 사용하여 AWS Command Line Interface(AWS CLI)을(를) 사용하는 AWS Lambda에 함수 코드를 배포하는 방법을 설명합니다.

단원

- [사전 조건](#)
- [도구 및 라이브러리](#)

사전 조건

AWS CLI은(는) 명령줄 셸의 명령을 사용하여 AWS 서비스와 상호 작용할 수 있는 오픈 소스 도구입니다. 이 섹션의 단계를 완료하려면 다음이 필요합니다.

- [AWS CLI – 버전 2 설치](#)
- [AWS CLI – aws configure를 통한 빠른 구성](#)

도구 및 라이브러리

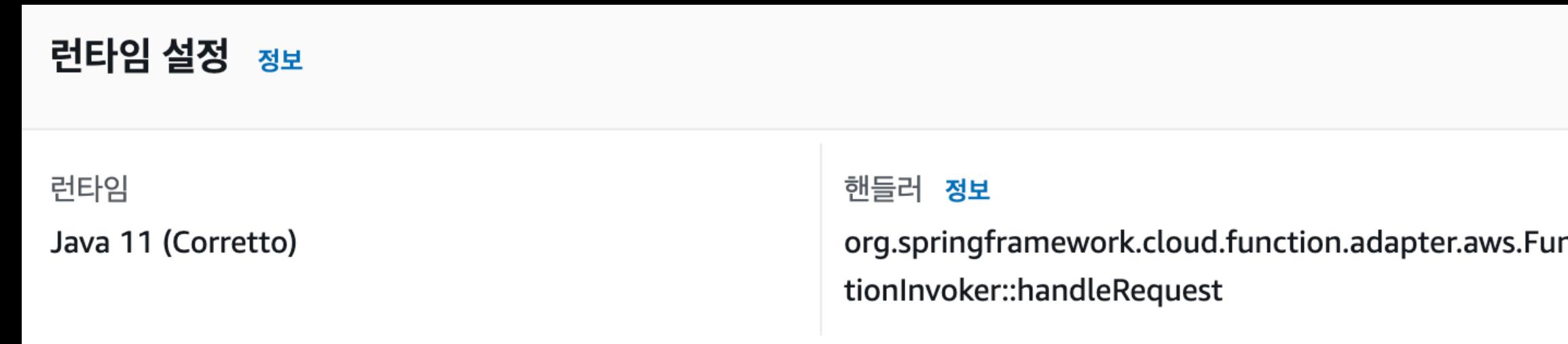
Lambda는 Java 함수를 위한 다음 라이브러리를 제공합니다.

- [com.amazonaws:aws-lambda-java-core](#) – 핸들러 메서드 인터페이스와 런타임이 핸들러에 전달하는 컨텍스트 객체를 정의합니다. 고유한 입력 유형을 정의하는 경우 이 라이브러리만 필요합니다.
- [com.amazonaws:aws-lambda-java-events](#) – Lambda 함수를 호출하는 서비스의 이벤트 입력 유형입니다.
- [com.amazonaws:aws-lambda-java-log4j2](#) – 현재 호출의 요청 ID를 [함수 로그](#)에 추가하는 데 사용할 수 있는 Apache Log4j 2용 appender 라이브러리입니다.

이 라이브러리는 [Maven Central Repository](#)를 통해 사용할 수 있습니다. 다음과 같이 빌드 정의에 이러한 라이브러리를 추가하세요.

AWS Lambda + Spring

- AWS Lambda 등록 시



핸들러를 Spring cloud function의 FunctionInvoker로 지정

org.springframework.cloud.function.adapter.aws.FunctionInvoker::handleRequest

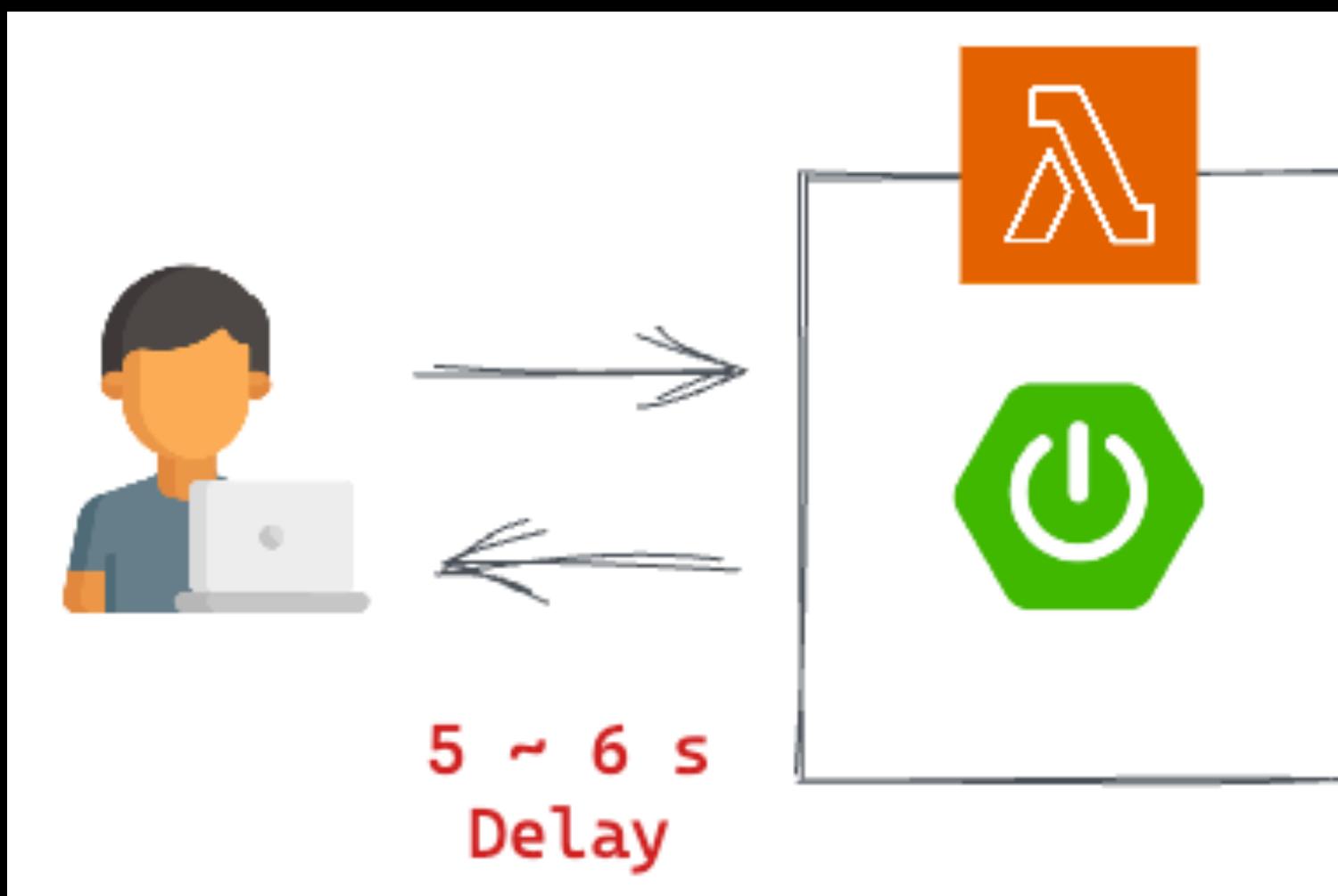
AWS Lambda + Spring

좋아! 돌려봅시다

예제는 요청 시 Slack API 호출하는 간단한 토이 프로젝트 입니다

AWS Lambda + Spring

[1] INFO 8 --- [main] lambdainternal.AWSLambda: Started AWSLambda in **4.145 seconds (JVM running for 5.659)**
[2] INFO 8 --- [main] lambdainternal.AWSLambda: Started AWSLambda in **3.977 seconds (JVM running for 5.486)**
[3] INFO 8 --- [main] lambdainternal.AWSLambda: Started AWSLambda in **4.097 seconds (JVM running for 5.67)**
[4] INFO 8 --- [main] lambdainternal.AWSLambda: Started AWSLambda in **3.906 seconds (JVM running for 5.385)**
[5] INFO 8 --- [main] lambdainternal.AWSLambda: Started AWSLambda in **3.941 seconds (JVM running for 5.495)**



5~6s Delay

요청을 해도 결과가 5~6초 뒤에 온다면?

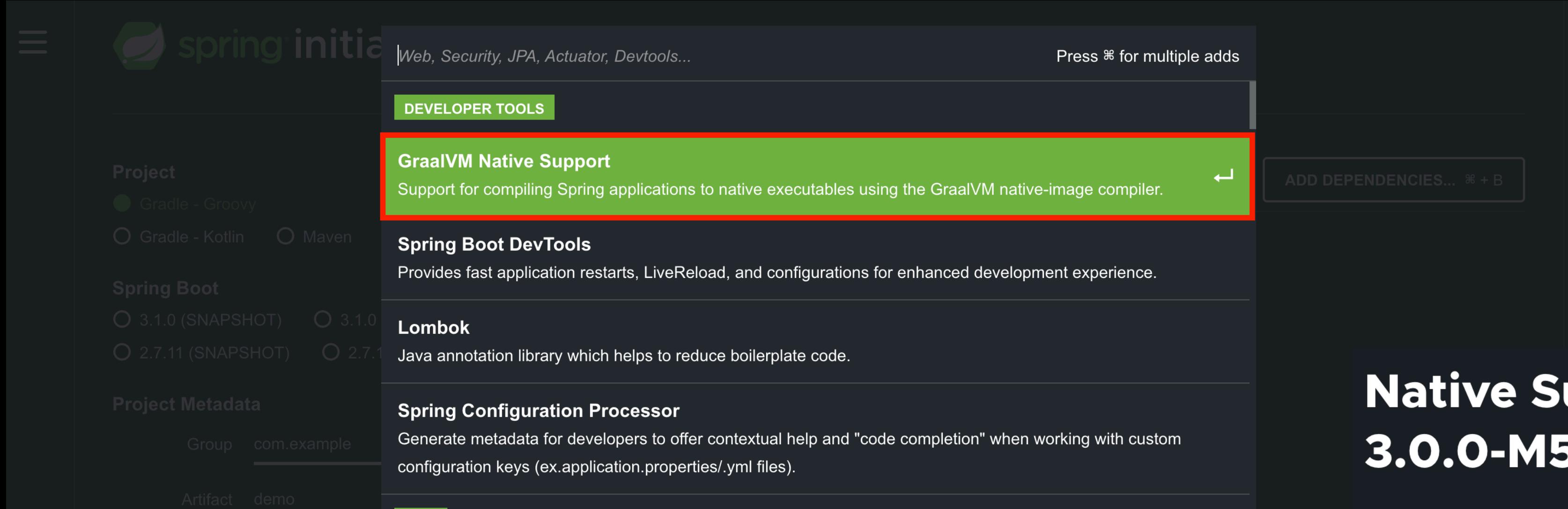
AWS Lambda + Spring

Cold Start

이것도 엄청 간단한 프로젝트의 예라는 것

프로젝트가 크면 더 Delay가 클텐데 버틸 수 있을까?

GraalVM Native Support



Native Support in Spring Boot 3.0.0-M5

ENGINEERING | STÉPHANE NICOLL | SEPTEMBER 26, 2022 | 40 COMMENTS

The Spring Team has been working on native image support for Spring Applications for quite some time. After 3+ years of incubation in the [Spring Native](#) experimental project with Spring Boot 2, native support is moving to General Availability with Spring Framework 6 and Spring Boot 3!

Native images provide almost instant startup time and reduced memory consumption for Java applications. The recent Spring Boot [3.0.0-M5](#) release marks the first time we're asking for broader community feedback on our native story. If you need to catch-up on the basics, please refer to the [Ahead Of Time basics blog post](#) from late March. You can also learn [how to prepare your applications for Spring Boot 3.0](#).

A lot has happened since March! We've improved compatibility with a larger number of use cases and libraries, fixing and improving our native support in the process. This blog post details what you need to know to get started.

이전까지는 Spring native 라는 experimental project 였음

-> Spring boot 3 부터 정식 지원

GraalVM Native Support - 공식문서

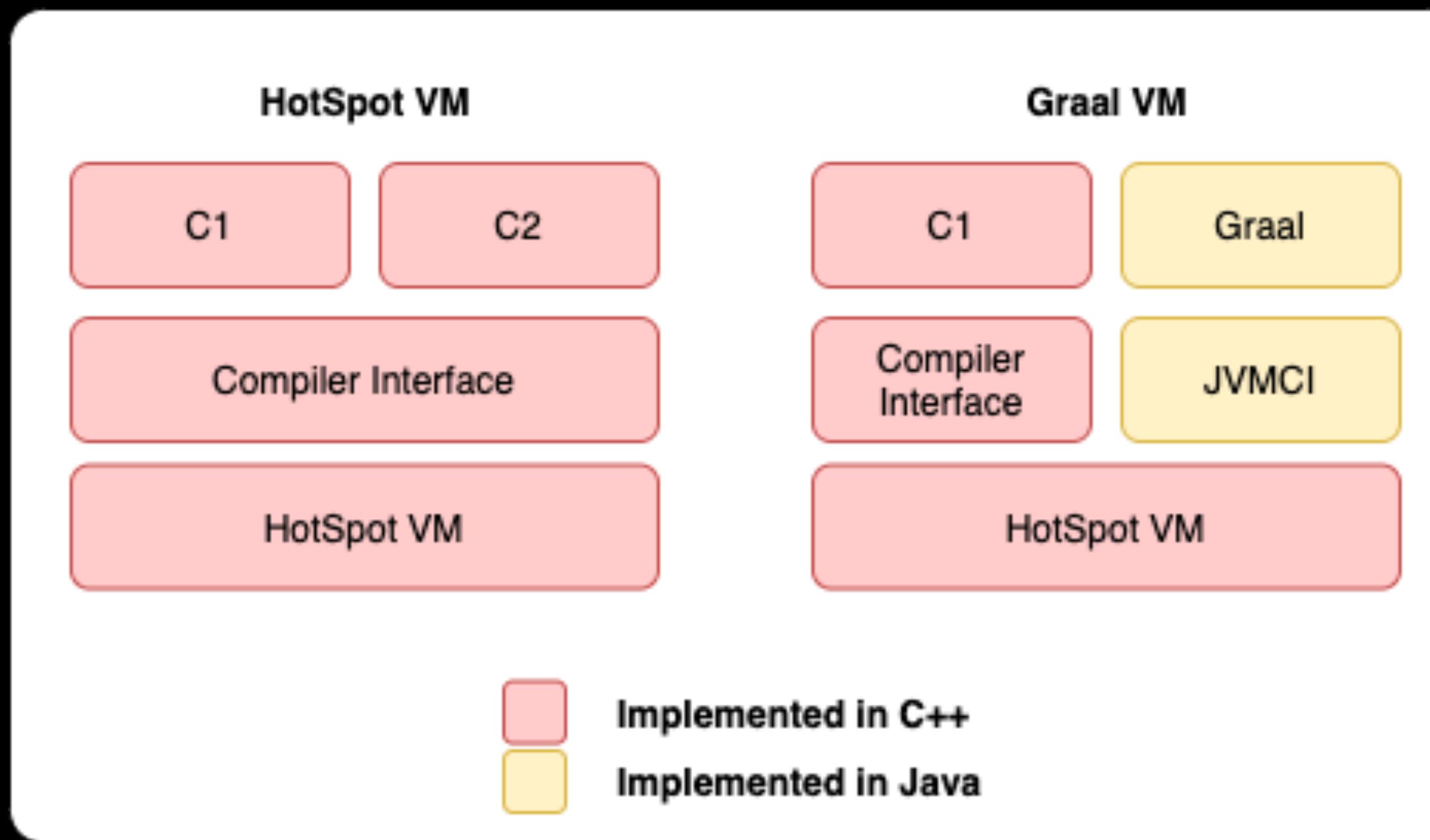
GraalVM Native Images provide a new way to deploy and run Java applications. Compared to the Java Virtual Machine, **native images can run with a smaller memory footprint and with much faster startup times.**

They are well suited to applications that are deployed using container images and are especially interesting when combined with "Function as a service" (FaaS) platforms.

Unlike traditional applications written for the JVM, GraalVM Native Image applications require **ahead-of-time processing in order to create an executable.** This ahead-of-time processing involves statically analyzing your application code from its main entry point.

A GraalVM Native Image is a complete, platform-specific executable. You do not need to ship a Java Virtual Machine in order to run a native image.

GraalVM



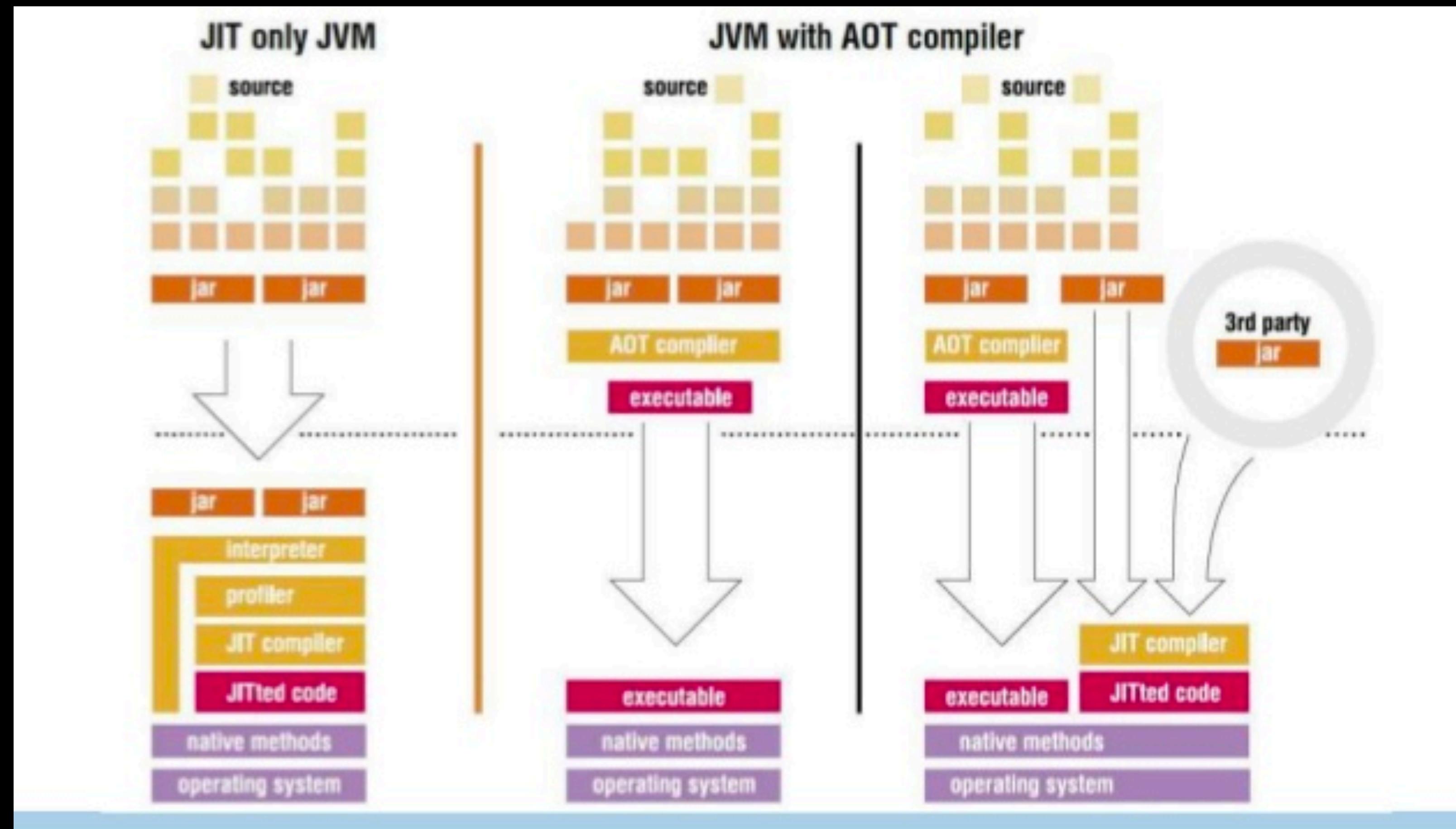
High performance optimized JIT (Just-In-Time) compiler

Polyglot programming (support Scala, Python, JS, ...)

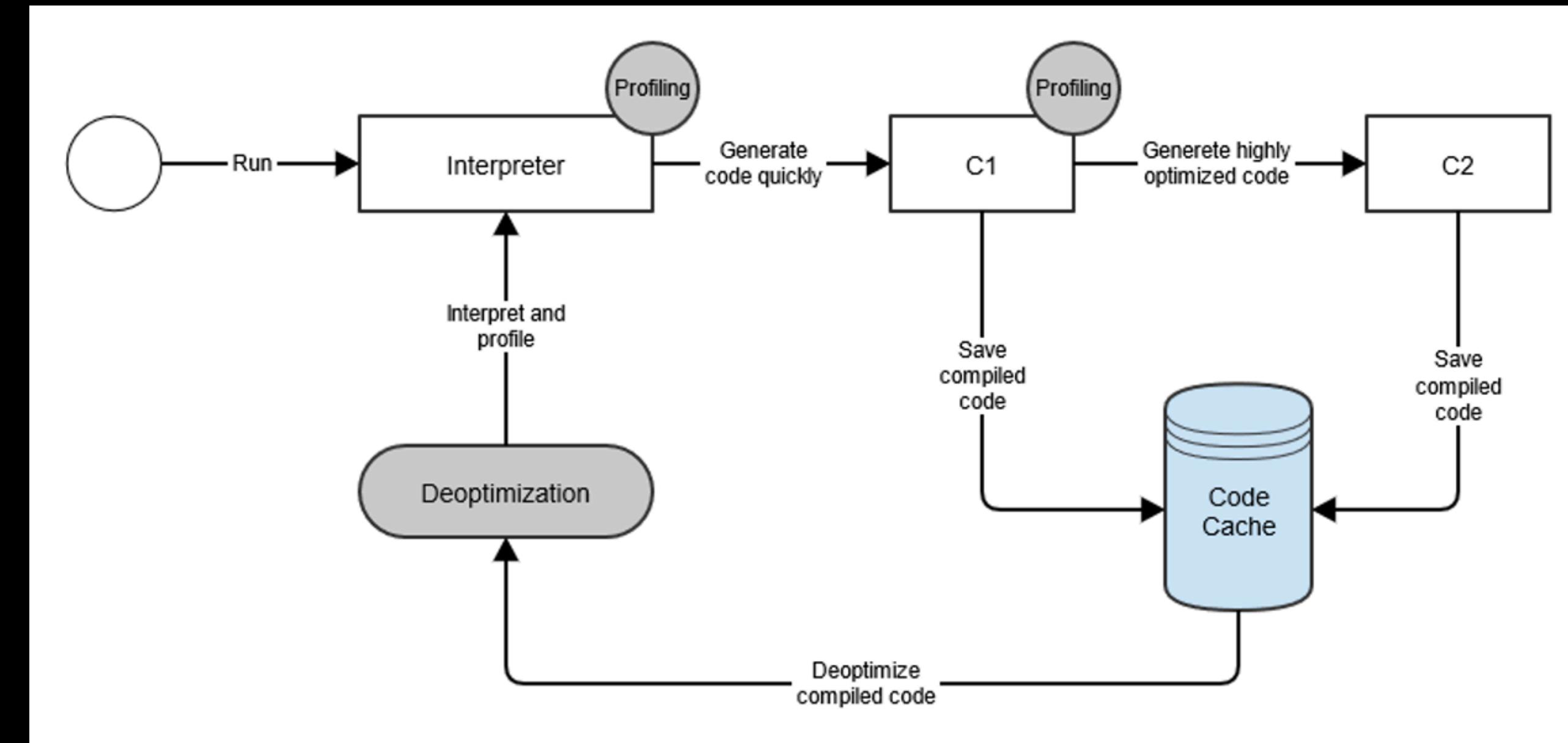
For microservices frameworks (Memory, startup time)

AOT (Ahead-Of-Time) Compiler, Native image

GraalVM



JIT (Just-In-Time)



Bytecode는 일반적으로 모든 플랫폼에서 실행할 수 있는 JAR 파일로 패키징

JAR 파일을 실행하면 실행 중 자주 사용되는 메서드(Hot Spot)이 식별되어 Native machine code로 실행 중 컴파일됨

이 과정에서 2가지 컴파일러를 사용

- C1 (빠른 컴파일, 낮은 최적화) - 초반
- C2 (느린 컴파일, 높은 최적화) - 후반

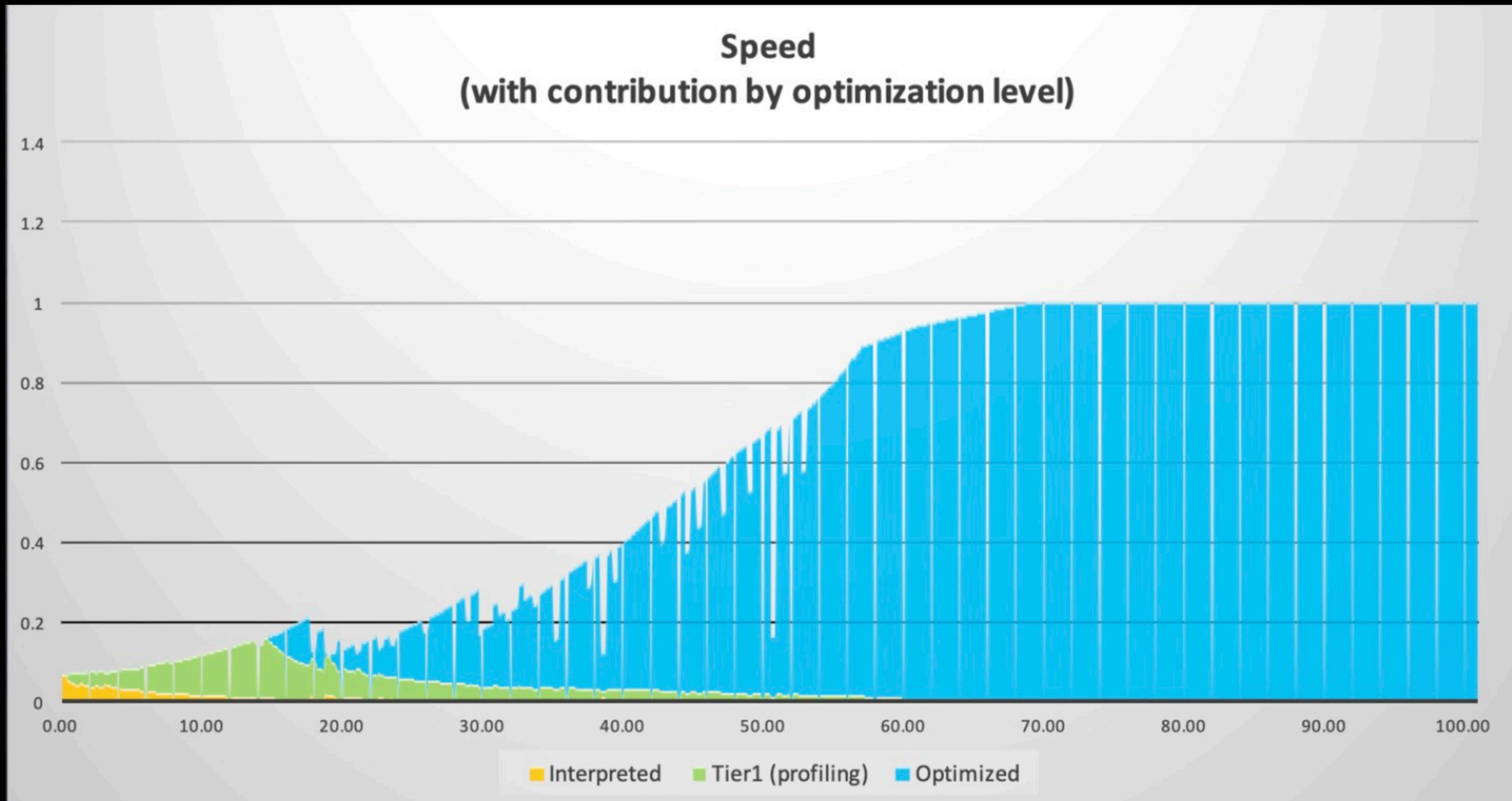
JIT (Just-In-Time)

```
PS C:\Users\NEEL KAUSHIK SHAH\Desktop> java -XX:+PrintCompilation CountUppercase Golang is the best
1277  1      3    java.lang.String::hashCode (55 bytes)
1362  2      3    java.lang.String::charAt (29 bytes)
1439  7      n 0   java.lang.System::arraycopy (native)  (static)
1441  3      3    java.lang.Character::toLowerCase (9 bytes)
1443  4      3    java.lang.CharacterData::of (120 bytes)
1446  5      3    java.lang.CharacterDataLatin1::toLowerCase (39 bytes)
1448  6      3    java.lang.CharacterDataLatin1::getProperties (11 bytes)
1455  8      3    java.lang.String::<init> (82 bytes)
1459  10     3    java.lang.String::length (6 bytes)
1461  11     3    java.lang.String::equals (81 bytes)
1465  9      3    java.util.Arrays::copyOfRange (63 bytes)
1483  12     3    java.lang.Object::<init> (1 bytes)
1514  13     3    java.lang.AbstractStringBuilder::ensureCapacityInternal (27 bytes)
1517  14     3    java.lang.String::indexOf (70 bytes)
1520  15     3    java.lang.AbstractStringBuilder::append (29 bytes)
1524  18     3    java.io.WinNTFileSystem::isSlash (18 bytes)
1530  17     3    java.lang.StringBuilder::append (8 bytes)
1537  19     s   java.lang.StringBuffer::append (13 bytes)
1541  20     3    java.lang.AbstractStringBuilder::append (50 bytes)
1548  16     3    java.lang.String::getChars (62 bytes)
1648  21     3    java.lang.Number::<init> (5 bytes)
1651  22     3    java.lang.Byte::<init> (10 bytes)
1654  24     1    java.lang.Object::<init> (1 bytes)
1657  12     3    java.lang.Object::<init> (1 bytes)  made not entrant
1660  23     3    java.lang.Short::<init> (10 bytes)
1665  25     3    java.lang.Long::<init> (10 bytes)
```

JIT Compilation이라고 함

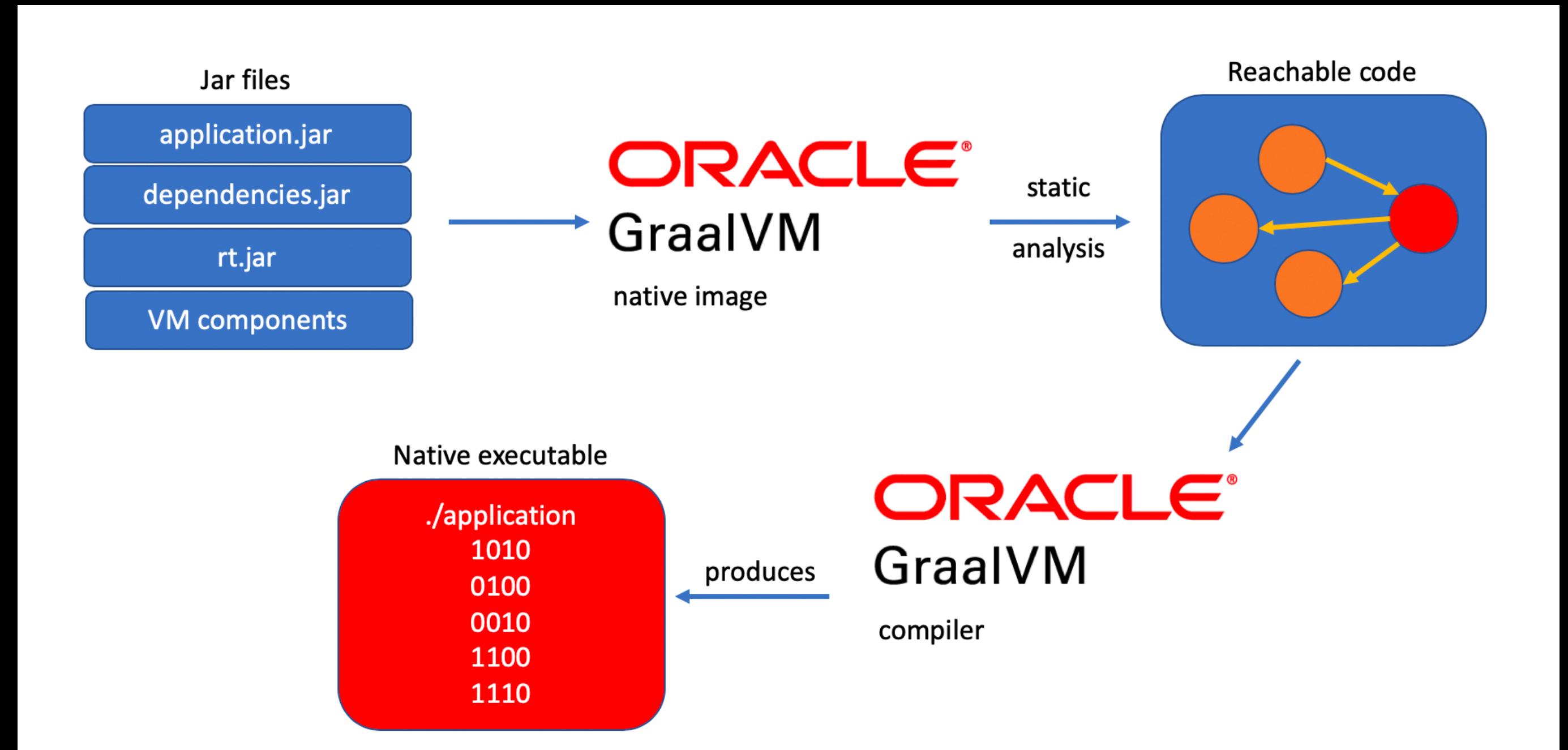
- JVM에서 **-XX:+PrintComilation** 옵션을 주면 Compilation 단계(3번째 열)를 볼 수 있음 (Compilation tier)
- 0단계 컴파일 X, 1~3단계가 C1 컴파일러에서 이루어진 것이고, 4단계가 C2 컴파일러에서 이루어진 것
- 즉, 시간이 갈수록 많이 실행되는 메서드는 최적화되어 더 빠르게 동작할 수 있다

JIT (Just-In-Time)



최적화 되지 않은 바이트 코드(노란색) // C1 네이티브 코드(녹색) // C2 네이티브 코드(파란색-최적 상태)

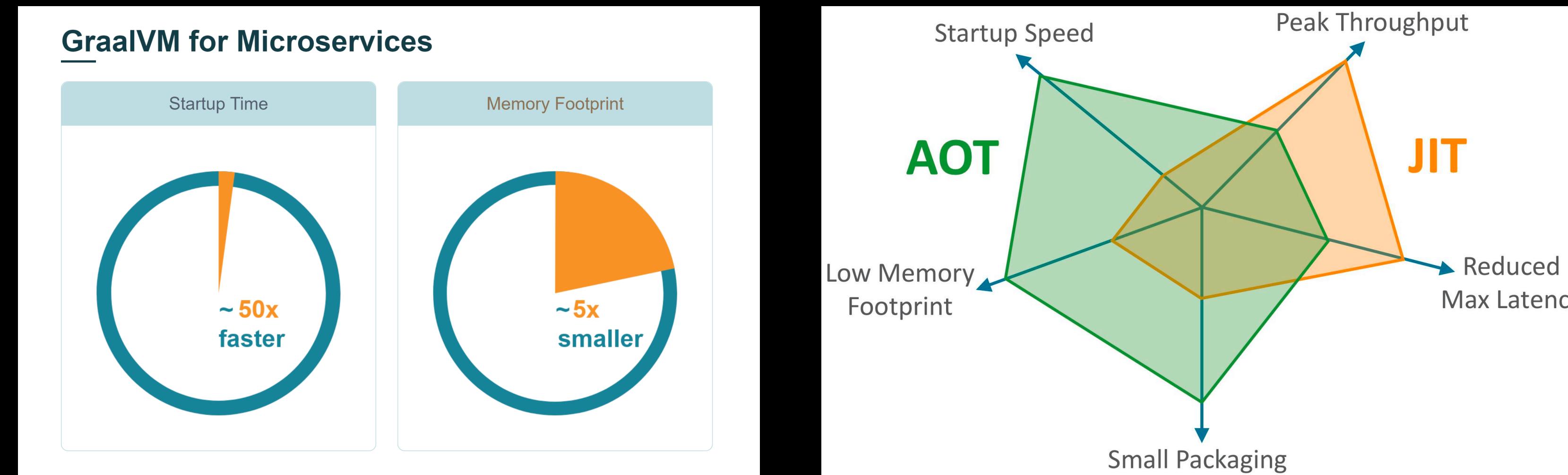
AOT (Ahead-Of-Time)



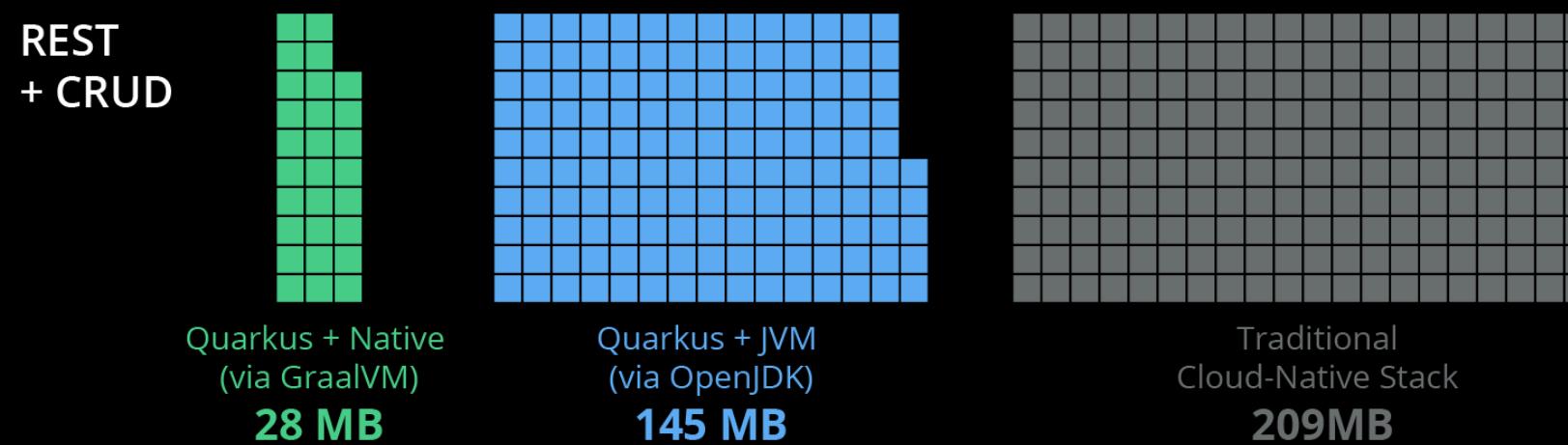
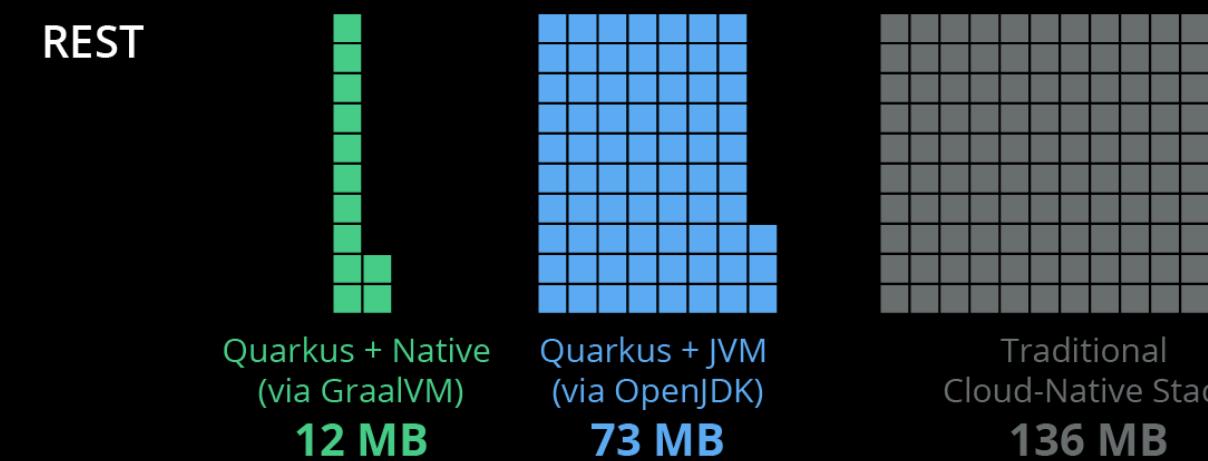
Java 코드를 정적으로 컴파일하고, 컴파일러가 특정 플랫폼용 Native machine code로 이루어진 Executable file을 생성

- 런타임 중 바이트 코드를 해석할 필요가 없고, 핫스팟을 식별할 필요가 없으며, 컴파일에 CPU 부하가 필요하지 않다. 즉, 실행시키면 바로 최적의 상태, 최고의 속도로 시작할 수 있다.
- 바로 실행파일을 만드는 것이기에 실행하는 환경에 JDK가 존재하지 않아도 된다. (SubstrateVM 이용)

GraalVM

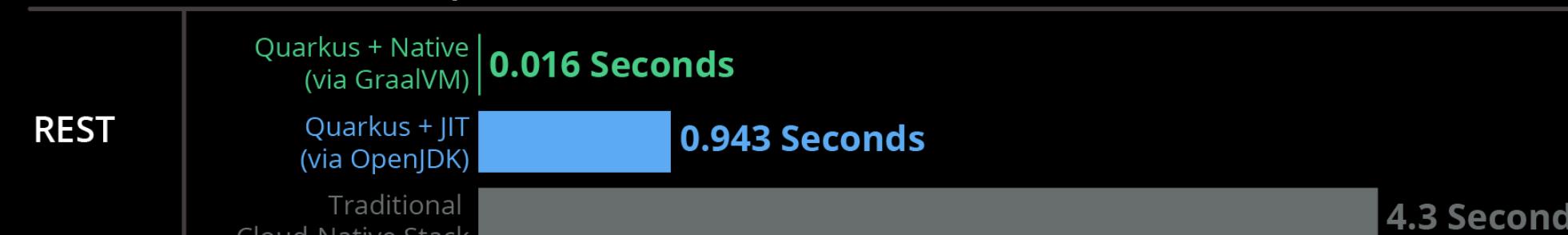


Memory (RSS) in Megabytes*



*Tested on a single-core machine

BOOT + First Response Time



No Silver Bullet 2222

GraalVM - native image

바로 실행파일을 만드는 것이기 만드는 플랫폼에 종속된다.

- Mac에서 빌드? Linux에서 돌아가지 않을 수 있다. => 어떻게 해? Docker, DevContainer 이용해야 함

처음부터 Native machine code로 컴파일을 해야하기 때문에 오래걸린다.

- 오래 걸릴 뿐더러 작업이 무겁다. (인텔 맥 아룩하니깐 조심)

런타임 중 동적으로 사용되는 클래스(reflection, resources, serialization, proxy usage etc)들은 미리 컴파일하지 못한다.

- 그래서 부가적으로 hint를 통해 알려주어야 한다.

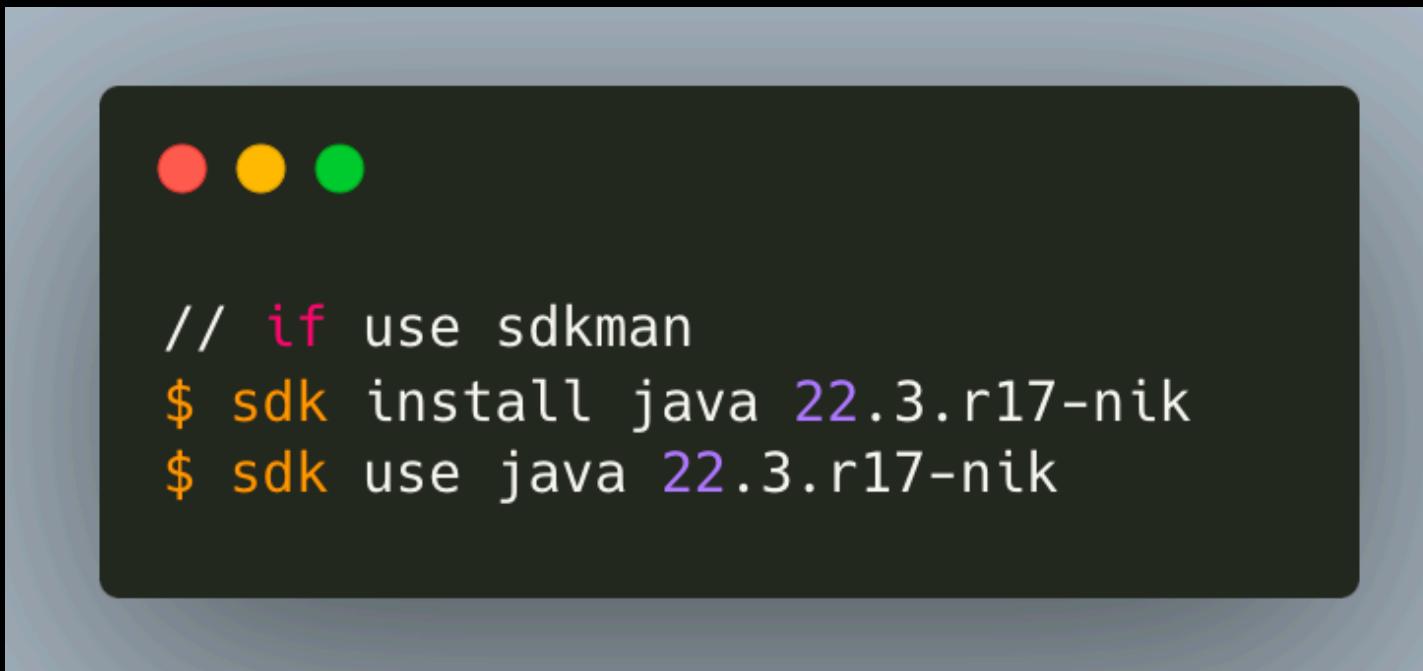
GraalVM Native Support

좋아! 사용해보자

문서 잘 되어있으니 참고

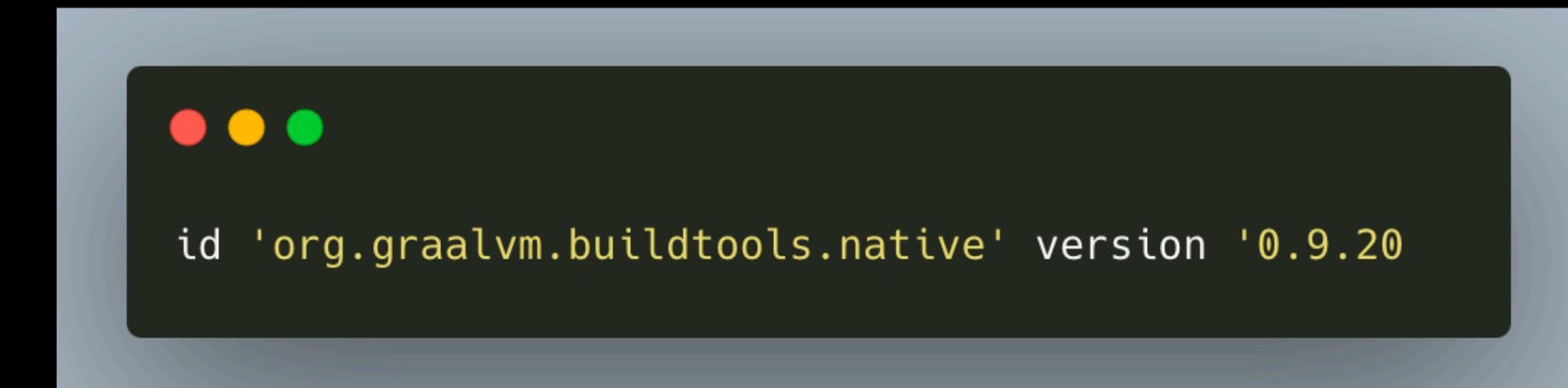
<https://docs.spring.io/spring-boot/docs/current/reference/html/native-image.html>

1. GraalVM 설치



```
// if use sdkman
$ sdk install java 22.3.r17-nik
$ sdk use java 22.3.r17-nik
```

2. GraalVM native support plugin 추가 (Gradle/Maven)



```
id 'org.graalvm.buildtools.native' version '0.9.20'
```

GraalVM Native Support

Native executable file 생성하기

Bean 한 개 등록한 아주 작은 프로젝트

일반 build하는 경우 (`./gradlew build`) 4초 소요

native compile 1분 4초 소요

GraalVM Native Support

Native executable file 실행하기

Bean 한 개 등록한 아주 작은 프로젝트

```
.
/\ \ / _____.----_ \ \ \ \ \ \
( ()\__| |' | | | | \ \ \ \ \
\ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \
' |____| .__| | | | | | \ \ \ \ \
=====|_|=====|_|/_=/_/_/_/
:: Spring Boot ::          (v3.0.4)

2023-03-25T03:36:38.309+09:00 INFO 21816 --- [           main] c.k.serverless.ServerlessApplication : Starting ServerlessApplication v0.0.1-SNAPSHOT using Java 17.0.6 with PID 21816
-spring-learning/serverless/build/libs/serverless-0.0.1-SNAPSHOT.jar started by user in /Users/user/Documents/open-source/spring-learning/serverless/build/libs)
2023-03-25T03:36:38.311+09:00 INFO 21816 --- [           main] c.k.serverless.ServerlessApplication : No active profile set, falling back to 1 default profile: "default"
2023-03-25T03:36:38.809+09:00 INFO 21816 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8080 (http)
2023-03-25T03:36:38.817+09:00 INFO 21816 --- [           main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2023-03-25T03:36:38.818+09:00 INFO 21816 --- [           main] o.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/10.1.5]
2023-03-25T03:36:38.866+09:00 INFO 21816 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[] : Initializing Spring embedded WebApplicationContext
2023-03-25T03:36:38.867+09:00 INFO 21816 --- [           main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 528 ms
2023-03-25T03:36:39.109+09:00 INFO 21816 --- [           main] o.s.c.f.web.mvc.FunctionHandlerMapping : FunctionCatalog: org.springframework.cloud.function.context.catalog.BeanFacto
f4
2023-03-25T03:36:39.134+09:00 INFO 21816 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8080 (http) with context path ''
2023-03-25T03:36:39.161+09:00 INFO 21816 --- [           main] c.k.serverless.ServerlessApplication : Started ServerlessApplication in 1.066 seconds (process running for 1.333)
```

일반 build하는 경우, 1.066 초 소요

native compile은 0.119 초 소요

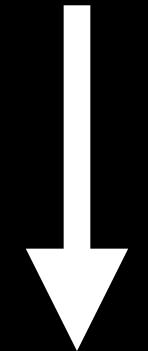
```
.
/\ \ / _____.----_ \ \ \ \ \ \
( ()\__| |' | | | | \ \ \ \ \
\ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \
' |____| .__| | | | | | \ \ \ \ \
=====|_|=====|_|/_=/_/_/_/
:: Spring Boot ::          (v3.0.4)

2023-03-25T03:29:49.267+09:00 INFO 18057 --- [           main] c.k.serverless.ServerlessApplication : Starting AOT-processed ServerlessApplication using Java 17.0.5 with PID 18057
-source/spring-learning/serverless/build/native/nativeCompile/serverless started by user in /Users/user/Documents/open-source/spring-learning/serverless/build/native/nativeCompile)
2023-03-25T03:29:49.269+09:00 INFO 18057 --- [           main] c.k.serverless.ServerlessApplication : No active profile set, falling back to 1 default profile: "default"
2023-03-25T03:29:49.303+09:00 INFO 18057 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8080 (http)
2023-03-25T03:29:49.306+09:00 INFO 18057 --- [           main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2023-03-25T03:29:49.306+09:00 INFO 18057 --- [           main] o.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/10.1.5]
2023-03-25T03:29:49.320+09:00 INFO 18057 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[] : Initializing Spring embedded WebApplicationContext
2023-03-25T03:29:49.320+09:00 INFO 18057 --- [           main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 51 ms
2023-03-25T03:29:49.346+09:00 INFO 18057 --- [           main] o.s.c.f.web.mvc.FunctionHandlerMapping : FunctionCatalog: org.springframework.cloud.function.context.catalog.BeanFacto
0a
2023-03-25T03:29:49.349+09:00 INFO 18057 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8080 (http) with context path ''
2023-03-25T03:29:49.349+09:00 INFO 18057 --- [           main] c.k.serverless.ServerlessApplication : Started ServerlessApplication in 0.119 seconds (process running for 0.167)
```

GraalVM Native Support

앞선 Slack API 호출 프로젝트

- [1] INFO 8 --- [main] lambdainternal.AWSLambda: Started AWSLambda in 4.145 seconds (JVM running for 5.659)
- [2] INFO 8 --- [main] lambdainternal.AWSLambda: Started AWSLambda in 3.977 seconds (JVM running for 5.486)
- [3] INFO 8 --- [main] lambdainternal.AWSLambda: Started AWSLambda in 4.097 seconds (JVM running for 5.67)
- [4] INFO 8 --- [main] lambdainternal.AWSLambda: Started AWSLambda in 3.906 seconds (JVM running for 5.385)
- [5] INFO 8 --- [main] lambdainternal.AWSLambda: Started AWSLambda in 3.941 seconds (JVM running for 5.495)



GraalVM Native Support를 이용해보자

GraalVM Native Support

BUILD SUCCESSFUL in 11m 36s 😂 → **in 0.15 seconds (JVM running for 0.169)** ✓

Build 11분 36초 -> Start 시간은 0.15초 소요

과연,,, Lambda 에서는 ?

GraalVM Native Support

과연,,, Lambda 에서는 ?

6.4 S

```
-----  
\\ / _'-_- -_( )_ -- -- - \ \\ \\  
(( ))\\_ | '_ | '_| | '_ \\\` | \\ \\ \\  
\\ \\_)| |_)| | | | | | | | | | | ) ) ) ) )  
' |____| .__|_|_|_|_|_\_, | / / / /  
=====|_|=====|_|/_=/_/_/_/  
:: Spring Boot ::  
  
2021-06-20 18:15:49.509 INFO 5 --- [ main] pe.viner.bot.VinerBotApplication : Starting VinerBotApplication  
2021-06-20 18:15:49.509 INFO 5 --- [ main] pe.viner.bot.VinerBotApplication : No active profile set, falling back to default profiles: default  
2021-06-20 18:15:49.709 INFO 5 --- [ main] o.s.s.c.ThreadPoolTaskScheduler : Initializing ExecutorService 'taskScheduler'  
2021-06-20 18:15:49.715 INFO 5 --- [ main] o.s.b.web.embedded.netty.NettyWebServer : Netty started on port 8080  
2021-06-20 18:15:49.716 INFO 5 --- [ main] pe.viner.bot.VinerBotApplication : Started VinerBotApplication in 0.309 seconds (JVM running for 0.556)  
✓
```

0.5 s

Cold Start 해결

그래서 이거 써야하나?

Serverless가 필수 일 때, 정말로 Java/Spring 환경이 필요하다면야 사용하겠지만 …

하지만 계속 드는 생각 ‘크흠… 굳이… Java 를…?’

그래서 이거 써야하나?

Serverless와 별개로 GraalVM Native Image은 향후 더 크게 발전할 수도 있을 것 같다

- 일단 Spring 프로젝트 커지면 Cold start + Warm up 시간만 장난 아니다
- Kubernetes 환경에 가면서 Scale out이 더 쉽게 일어나고 이에 따라 빠르게 동작할 수 있는 어플리케이션이 필요하지 않을까
- Kubernetes 환경의 Serverless (like AWS Fargate) 역할로도 잘 활용될 수 있을 것 같다

이 쪽을 타겟팅한 다른 Java framework도 많다. 국내에선 많이 활용하진 않지만 오히려 더 발전한 부분이 있는 듯



감-사합니다