NTT DATA

AWS Serverless for Java Developers with Quarkus and GraalVM

AGILE/
DEVOPS
GLOBAL
CONFERENCE

Alexander Burkard

Technical Consultant @ NTT DATA

Alexander Burkard

Technical Consultant
NTT DATA

- Studied Computer Science
- Located near munich
- Software Engineer, Architect, Trainer, Teamlead, …
- Married and father of two girls
- Likes
  - being in company
  - coding
  - playing ball sports (kind of on hold)
  - reading books (definitely on hold)
- Does not like so much
  - running
  - romantic movies

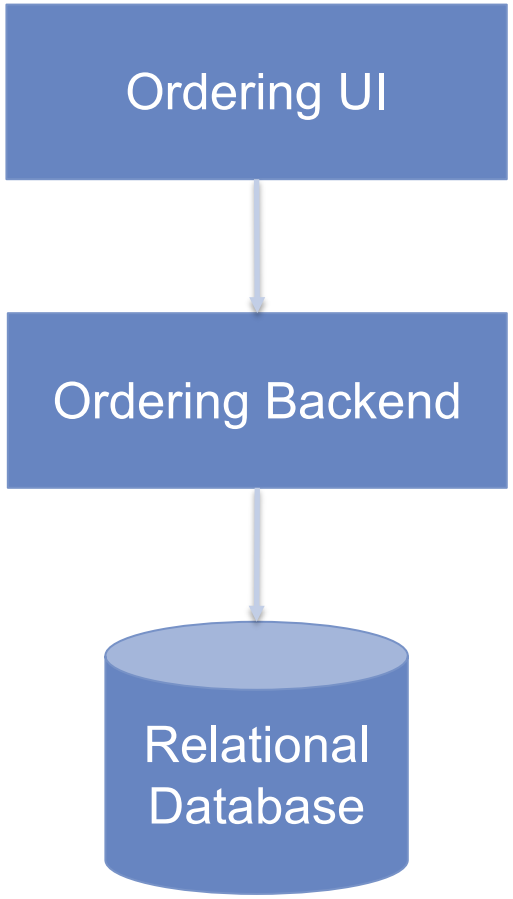
AWS Serverless for Java Developers with Quarkus and GraalVM

GraalVM™

Is it really that simple?

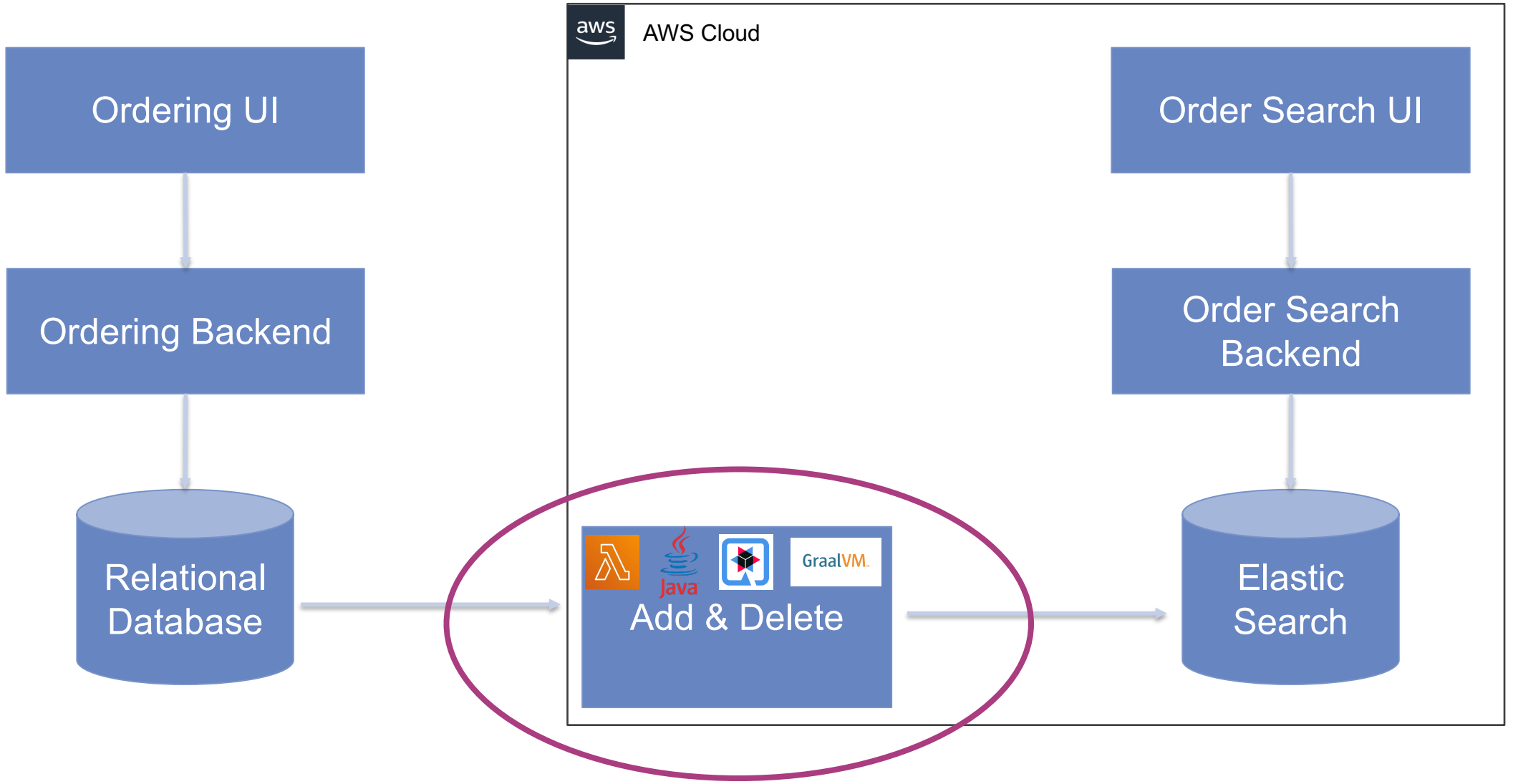
Let's find out!

The Ordering Application - Automotive industry



- Java Enterprise Application
- Monolithic 3-Tier Architecture
- Runs on premises
- Rolled out globally
- Many Interfaces

The Ordering Application - Cloud Migration



What is AWS Lambda?

A serverless compute with the following characteristics

Pay per request	Scaling and load balancing built-in	Availability and fault tolerance built-in	No servers to provision or manage	Sustainable	Focus on business logic

Memory [MB]	Price per Millisecond [$]
512	0.0000000083
1024	0.0000000167

Example: 500.000 request * 1.000 ms = 8$

What Programming Languages (Runtimes) are used out there for AWS Lambda?

Java ~5%	Python ~25%	Node.js ~65%	Others ~5%

Why is this, when Java is one of the most popular programming languages?

Why is Java not a Popular Lambda Runtime?

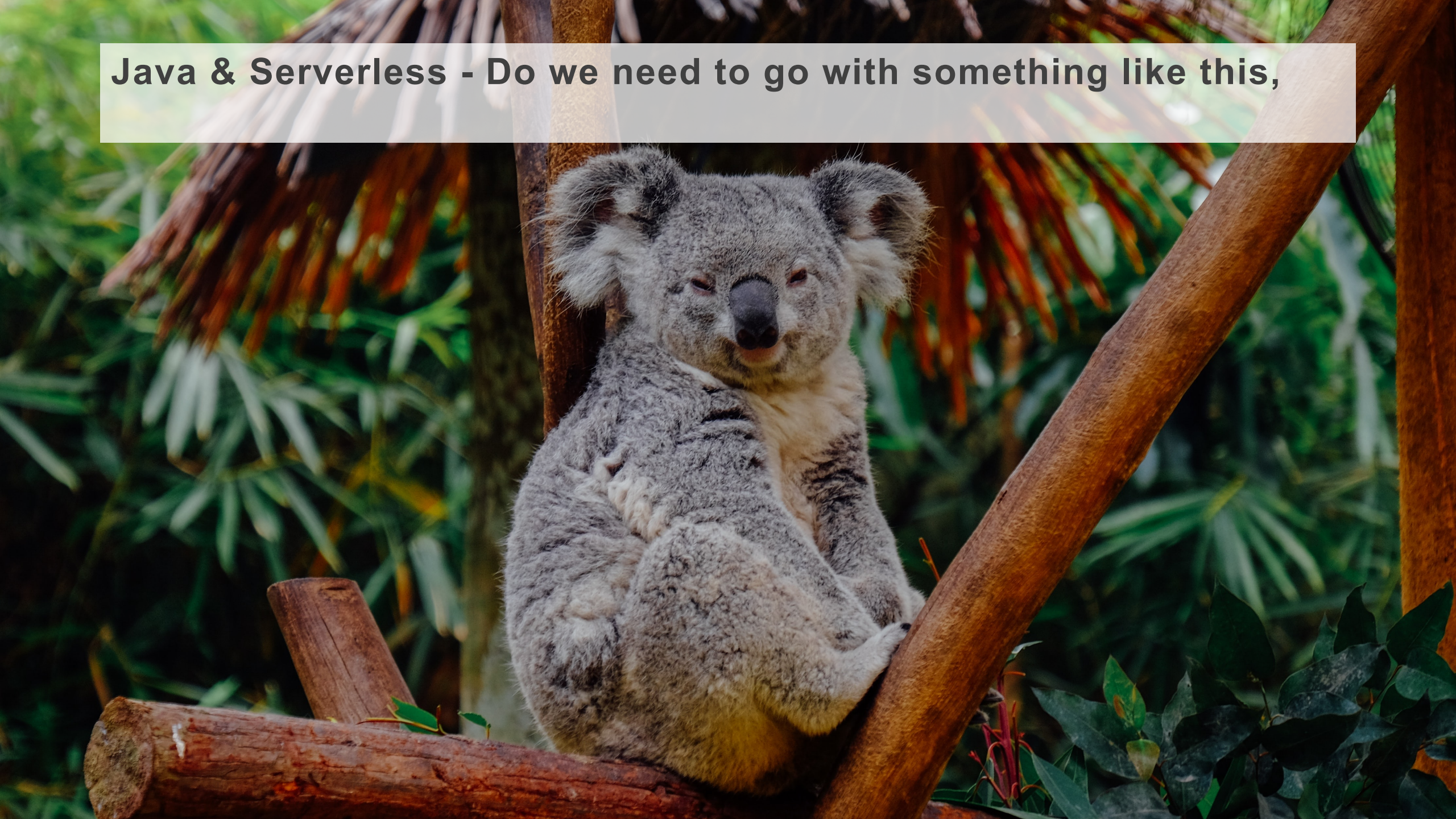
Runtime	Cold start [ms]
Java	2000-6000
Python	500
Node.js	700
Top Tier (Go, Rust, …)	12

Runtime	Warm start [ms]
Java	11
Python	11
Node.js	17
Top Tier (Go, Rust, …)	9

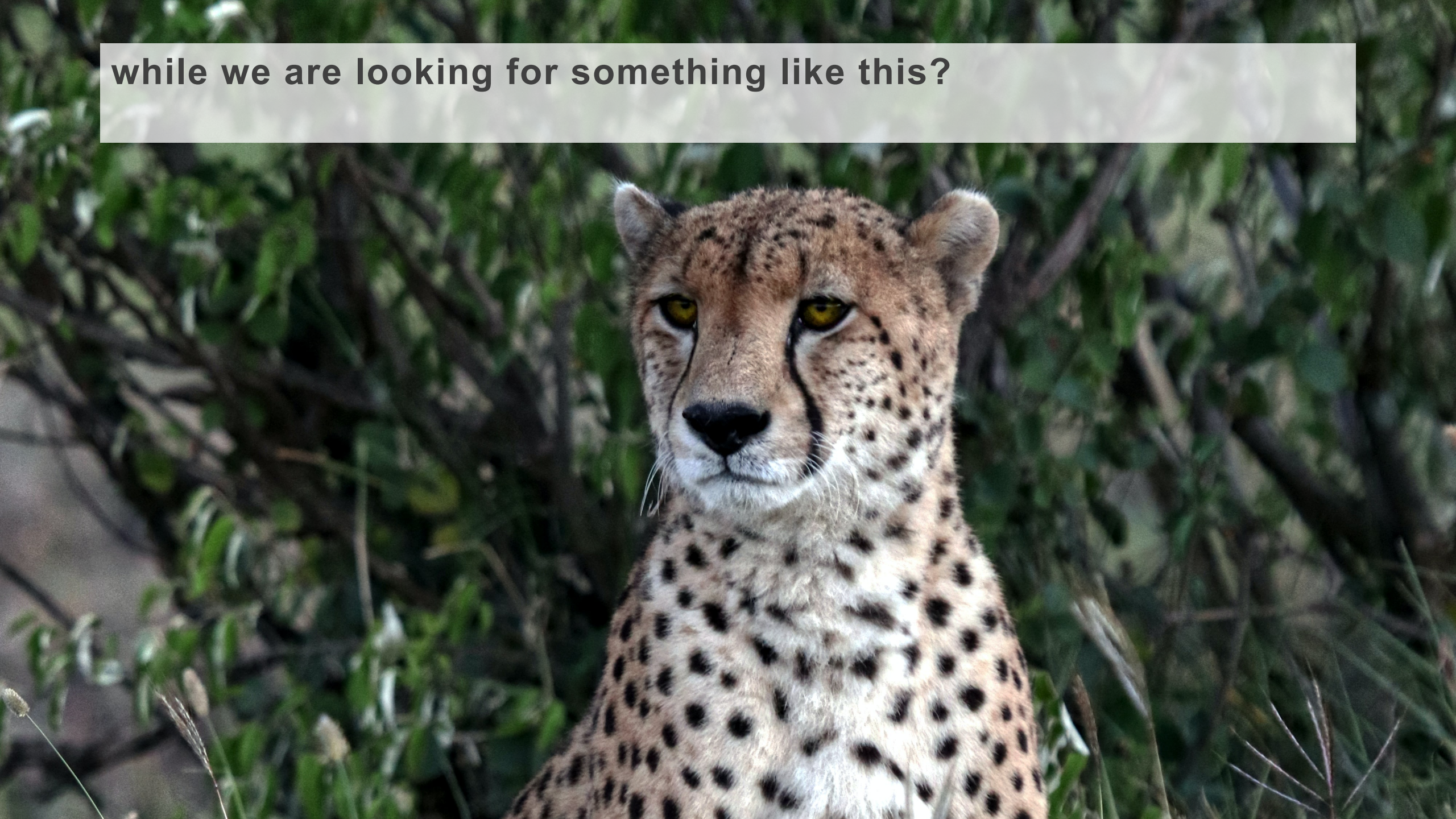
- Java memory usage is up to 7-times higher than other runtimes ($)
- Java deliverable is up to 15-times bigger than other runtimes

Why would anybody choose Java?

Java & Serverless - Do we need to go with something like this,



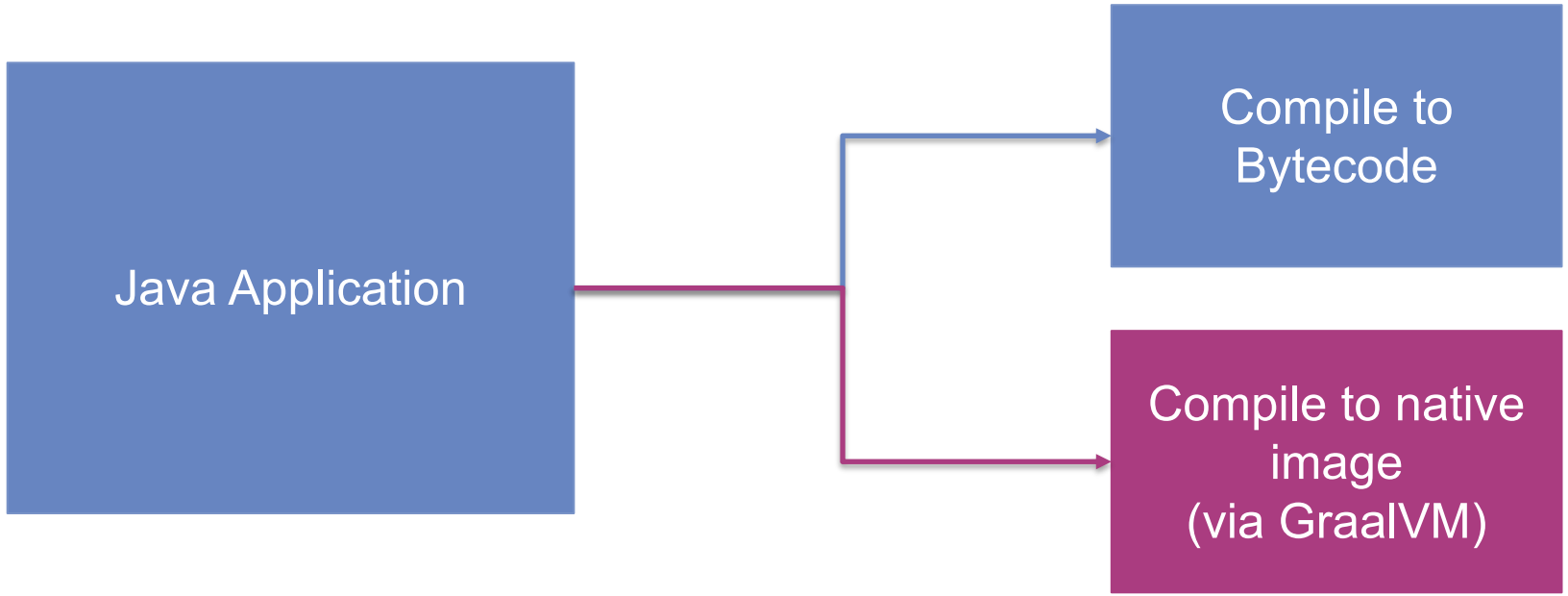
while we are looking for something like this?



Quarkus with Graal VM to the Rescue

Quarkus was created to enable Java developers to create applications for a modern, cloud-native world. Quarkus is crafted from best-of-breed Java libraries and standards (including Microprofile and Spring).

Quarkus + GraalVM address the Java issues in the serverless world.

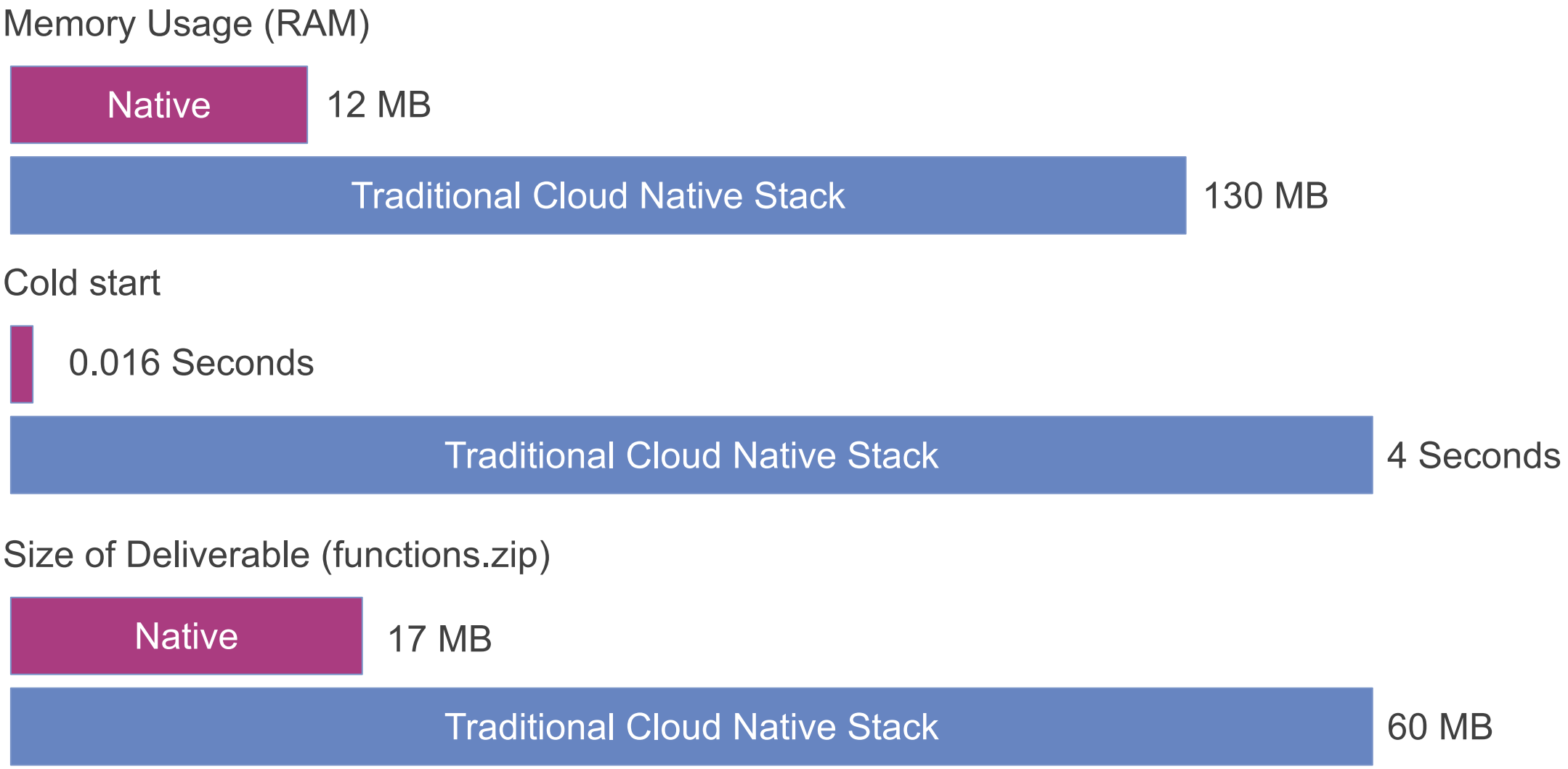


Use a Dockerized Native Build in your Quarkus Project

```
## JVM Build
build:
   mvn clean package

## Native Build with docker
build-native-using-docker:
   mvn clean package -Pnative -Dquarkus.native.container-build=true
```

Quarkus with Graal VM – The Numbers



Java & Serverless!

Trade-offs

Massive increase in Build Time	Build + Unit Test via maven • JVM 10 seconds • Native 3 minutes	• Turn around time for local Development • CICD pipeline feedback
Closed World Principle	Everything needs to be known at compile time	• No reflection, dynamic class loading, dynamic proxy, JNI(Java Native Interface) at runtime • These need to be registered at build time • Crashes at runtime
No JVM	When you compile to native you are not running on a JVM	• No JVM based thread and memory analysis • No Java Debugging (you need to go with GDB, Vtune, etc.) • No platform independence • No Just In Time Compiler

Conclusion – There is no silver bullet

You need to ask yourself some questions and make the decision for your project

Is it Java?	Do I care about the cold start delay?	Can I compensate the cold start delay?	Is there a price reduction?	Is there a new feature that solves my issue?

-> The native build adds complexity (Tooling, Local Development, Building, Testing, …) to your project.

NTT DATA

THANK YOU

AGILE/
DEVOPS
GLOBAL
CONFERENCE