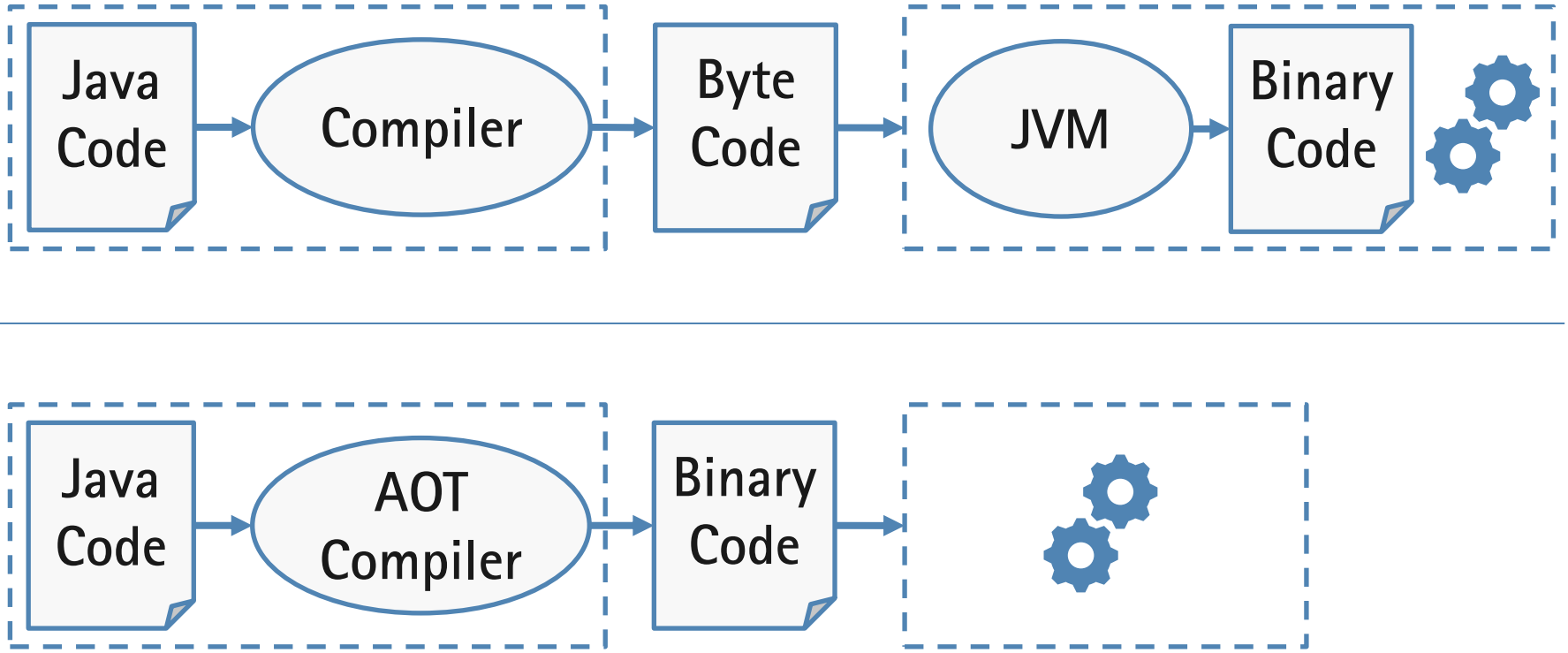
Quarkus Native

GEDOPLAN GmbH

Jan Pohlmeyer

Vom Java Code zur Ausführung



Was versprechen wir uns davon?

- Keine Java-Laufzeitumgebung notwendig
  - => kleinere Container Images
- Blitzschnelle Startzeit
- Geringere Ressourcennutzung

Native Java Anwendungen mit GraalVM/Mandrel

- GraalVM wurde von Oracle als Alternative zur JVM entwickelt
- Bietet zusätzlich eine Ahead-of-time (AOT) Kompilierung
- Resultierendes Artefakt ist eine direkt ausführbare Anwendung
- Keine Java-Laufzeitumgebung benötigt
- Quarkus Native => nativ ausführbare Quarkus-Anwendungen
- Anwendungsausführung in Linux Containern => Mandrel empfohlen
- Mandrel = Downstreamdistribution der GraalVM Community Version
- Hauptziel von Mandrel ist es Quarkus Native Builds durchzuführen
- Native Build kann in Quarkus-Projekten per Property aktiviert werden

Demo: Quarkus (Native) Anwendungen ausführen

- Dev-Mode (nicht native)

```
./mvnw quarkus:dev
```

- JAR Artefakt ausführen (nicht native)

```
java -jar target/quarkus-app/quarkus-run.jar
```

- Natives Artefakt ausführen

```
./target/quarkus-native-1.0.0-SNAPSHOT-runner
```

Demo: Vergleich Startzeiten und Ressourcennutzung

- Startzeit:
  - JVM: 1.429s
  - Native: 0.018s
- Ressourcennutzung:
  - JVM: CPU: 23% | Peak RSS: 150MB
  - Native: CPU: 0% 😳 | Peak RSS: 70MB

Demo: Laufzeit

- einfacher REST-Call
  - Klasseninitialisierung zur Buildtime
- mit JSON-Deserialisierung
- mit JSON-Deserialisierung aus Ressourcen
- Faces Expression Language

Demo: Quarkus Native Anwendungen bauen

- Einfacher nativer Build (für aktuelle Umgebung)

```
./mvnw package -Dnative
```

- Nativer Build im Container (Artefakt mit Linux ausführbar)

```
./mvnw package -Dnative -Dquarkus.native.container-build=true
```

- Container-Image als Build-Artefakt hinzufügen

```
./mvnw package -Dnative -Dquarkus.container-image.build=true
```

Demo: Vergleich Build

- Zeit:
  - JVM: 00:23 min
  - Native: 03:35 min
- Ressourcennutzung:
  - JVM: CPU: 405% | Peak RSS: 550MB
  - Native: CPU: 602% | Peak RSS: 3.6GB

Der Trade-Off (There is no such thing as a free lunch)

- Längere und ressourcenintensive Builds
- Platformabhängiges Build-Artefakt
- GraalVM/Mandrel (oder Containerengine) im Buildsystem notwendig
- Geringerer Durchsatz (keine Laufzeitoptimierungen)
- Komplexeres Debugging
- Vorbereiten der Anwendung
  - Ressourcen explizit einbinden
  - Mögliche Probleme bei der Nutzung von Reflection
  - Mögliche Probleme mit Bibliotheken (ohne Quarkus Integration)
  - ...

Anwendungsfälle

- Ideal für Serverless Functions und Microservices
- Wenn Startzeiten der Anwendung sehr kritisch sind
  - z.B. schnelle Skalierung der Service Instanzen on-request
- Bei kurzen Anwendungslaufzeiten und häufigen Restarts
  - JIT (Just-in-time) Compiler der JVM kann durch Laufzeitoptimierungen besseren Durchsatz ermöglichen (https://quarkus.io/blog/new-benchmarks/)
- Wenn geringere Arbeitsspeichernutzung kritisch ist

- Alternativ interessant: JDK AOT Caching (https://gedoplan.de/schneller-geht-immer-ahead-of-time-class-loading-mit-java-25/)

Noch Fragen?

- https://github.com/GEDOPLAN/ekj-quarkus-native