Cloud native für den Enterprise Java Developer mit Quarkus

Version: 1.0

Ihr Sprecher

Serge Ndong

Trainer, Berater, Entwickler

Architektur

Java und Cloud

Persistenz

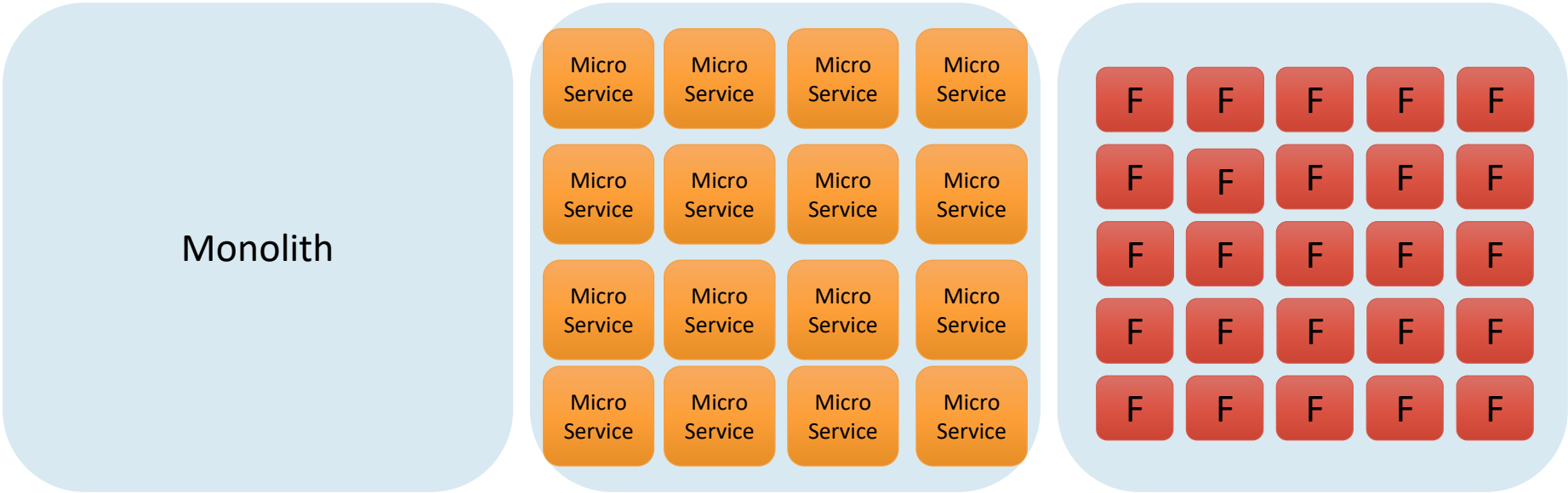
Gliederung

- Einführung
- Was ist Quarkus?
- Wie funktioniert Quarkus?
- Quarkus extensions
- Native Mode

Einführung

- JVM Warmup Problem
- Vom Monolith zu Microservices
- Unterschied bei der Skalierung
  - Monolith -> vertikale Skalierung
  - Microservice -> horizontale Skalierung
- Continuous Delivery
  - Bugfix, Features
- Elastizität, Skalierung auf der Cloud

Monolith zu



Enterprise Java und Container

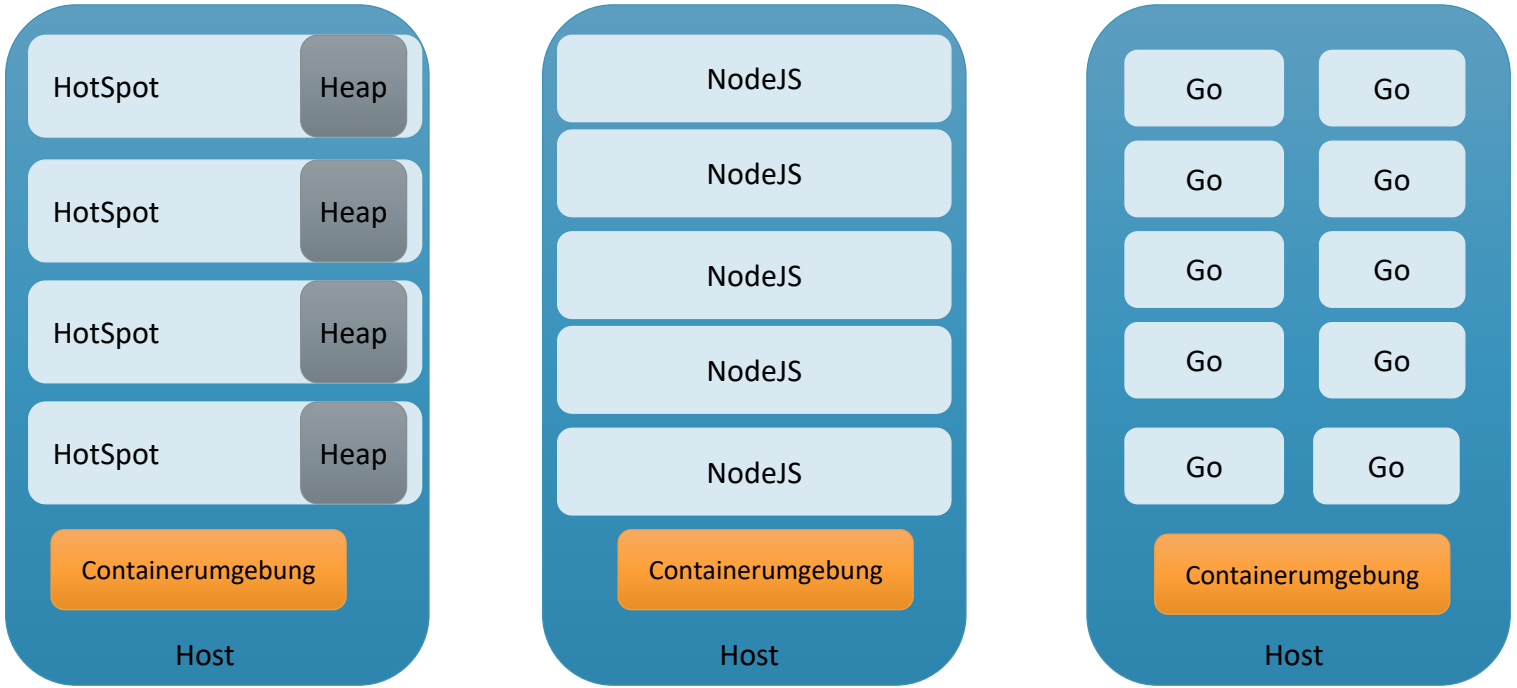
part of trivadis group

- Konzipiert für schnellen Durchsatz (Requests/s)

- Lange Startzeiten

- Speicher-Overhead
  - große JDK, Framework, App

Enterprise Java und Container



Was ist Quarkus?

part of trivadis group

- Supersonic Subatomic Java
- Toolkit und Framework zum Schreiben von Java, Kotlin und Scala Applikationen
- Leichtgewichtig und cloud-freundlich
- Design fokussiert auch den Einsatz auf GraalVM
- Extensions für „jedes“ Framework
- Vereinheitlicht imperative und reaktive Programmierung
- Best-of-Breed Ansatz
    - Frameworks und Standards

Was ist Quarkus?

part of trivadis group

- Benutzt vorhandene Standards
- Java EE
  - Servlet, JAX-RS
  - JPA, JDBC, Transactions
  - CDI, Bean Validation
- Microprofile
  - Fault Tolerance, Health, JWT, Metrics
  - OpenAPI, OpenTracing, Reactive Messaging, Rest Client
- Spring über einen Kompatibilitätslayer
  - Spring Dependency Injection
  - Spring Web
  - Spring Data

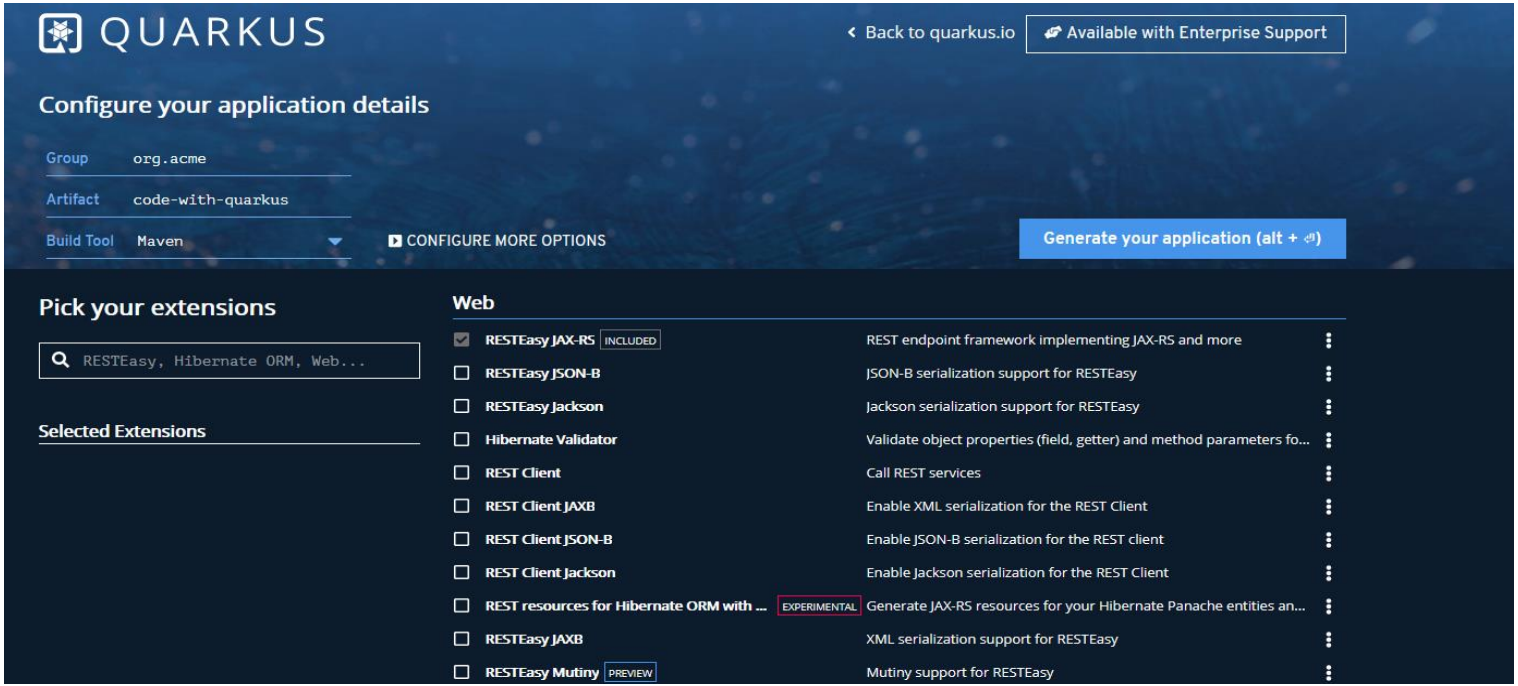
Was ist Quarkus?

part of trivadis group

- Datenbanken
  - MySQL,
  - MariaDB
  - MS SQL Server
  - H2
  - FlyWay
  - Amazon DynamoDB
  - MongoDB
  - Neo4j
  - …

Developer Joy

- Einfach zu starten
  - `mvn io.quarkus:quarkus-maven-plugin:1.7.2.Final:create`
  - code.quarkus.io



Quelle: https://code.quarkus.io/

Developer Joy

part of trivadis group

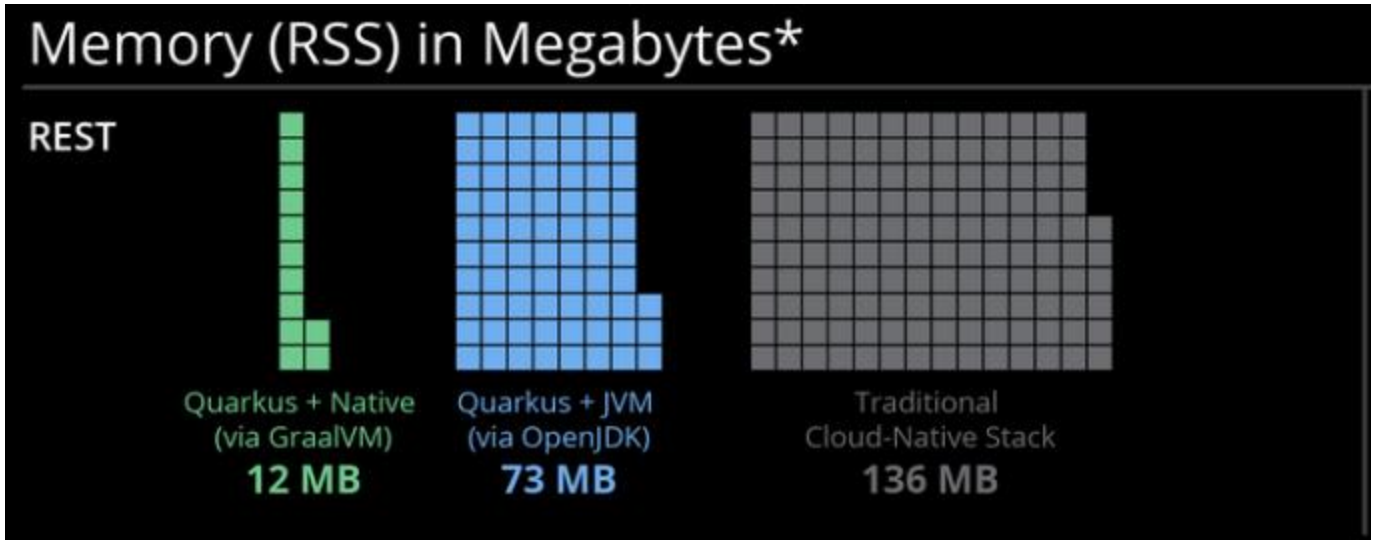
- Kaum Konfiguration
- Live Reload
  - `mvn compile quarkus:dev`
- Extensions for Junit 5
  - `@QuarkusTest`
  - `@NativeImageTest`
- Isoliert von der GraalVM CLI/API
- Maven oder Gradle Plugins
- Java, Kotlin oder Scala

Container-freundlich

part of trivadis group

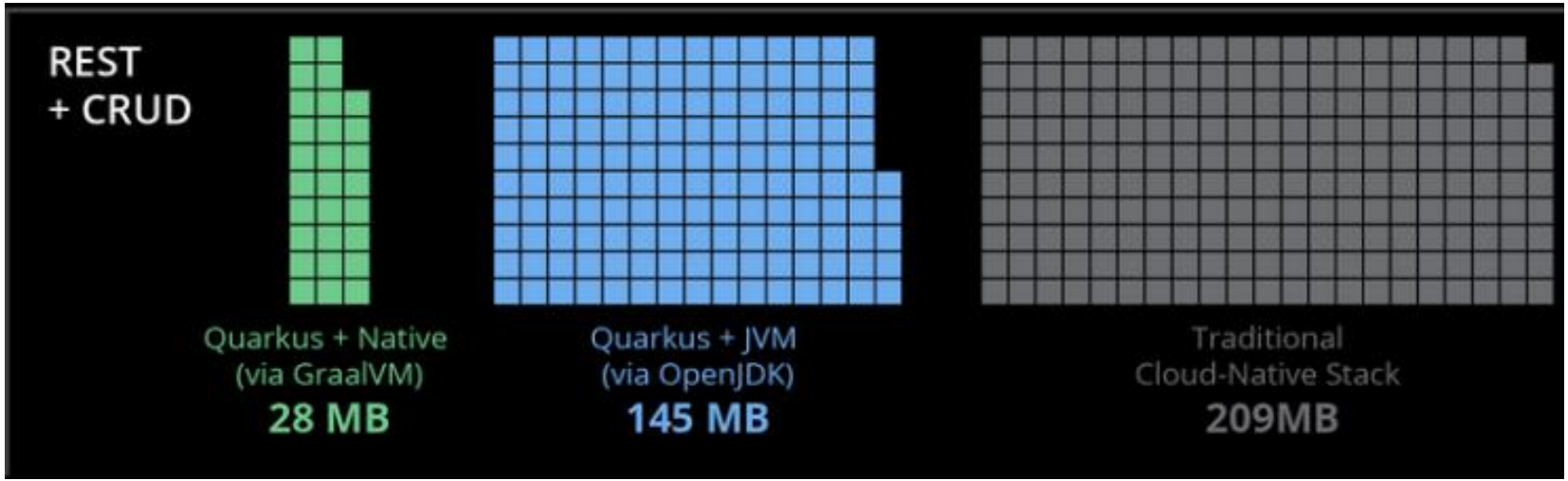
- Kleine Container-Images
- Schnelle Start/Bootzeiten
- Geringer Speicherbedarf
- Geringer Resident Set Size (RSS) Speicher
- RSS = Gesamte Speicherverbrauch des Prozesses > Heap Size

Quarkus Benchmarks I



Quelle: https://quarkus.io/

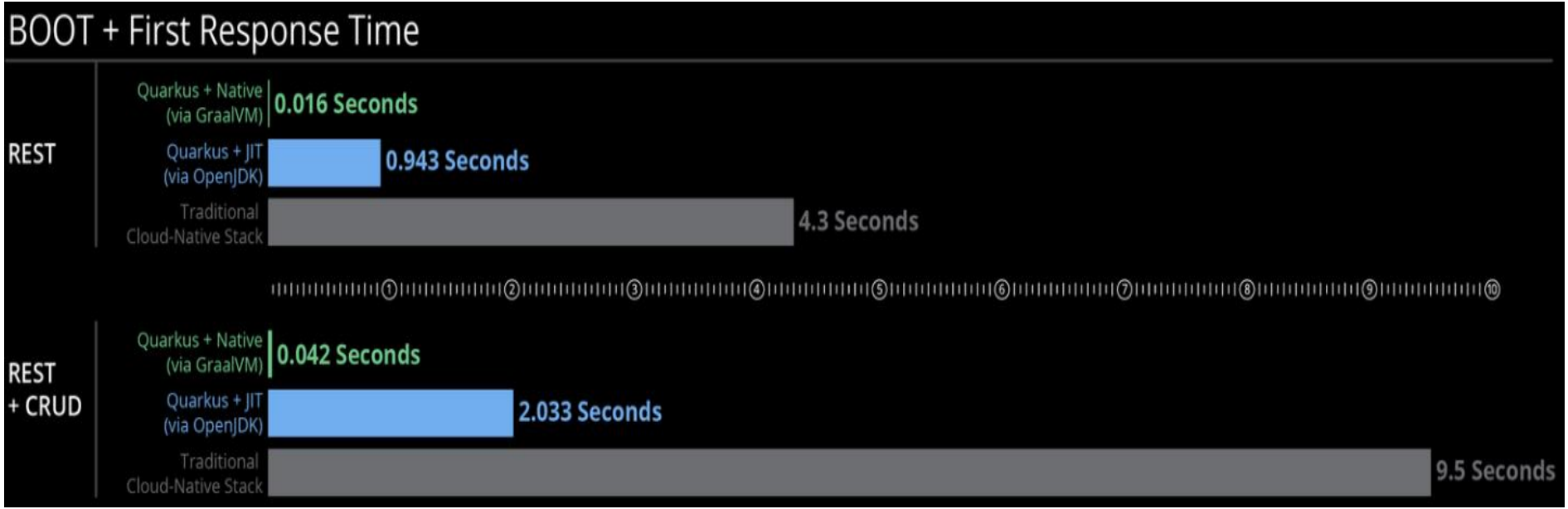
Quarkus Benchmarks II



Quelle: https://quarkus.io/

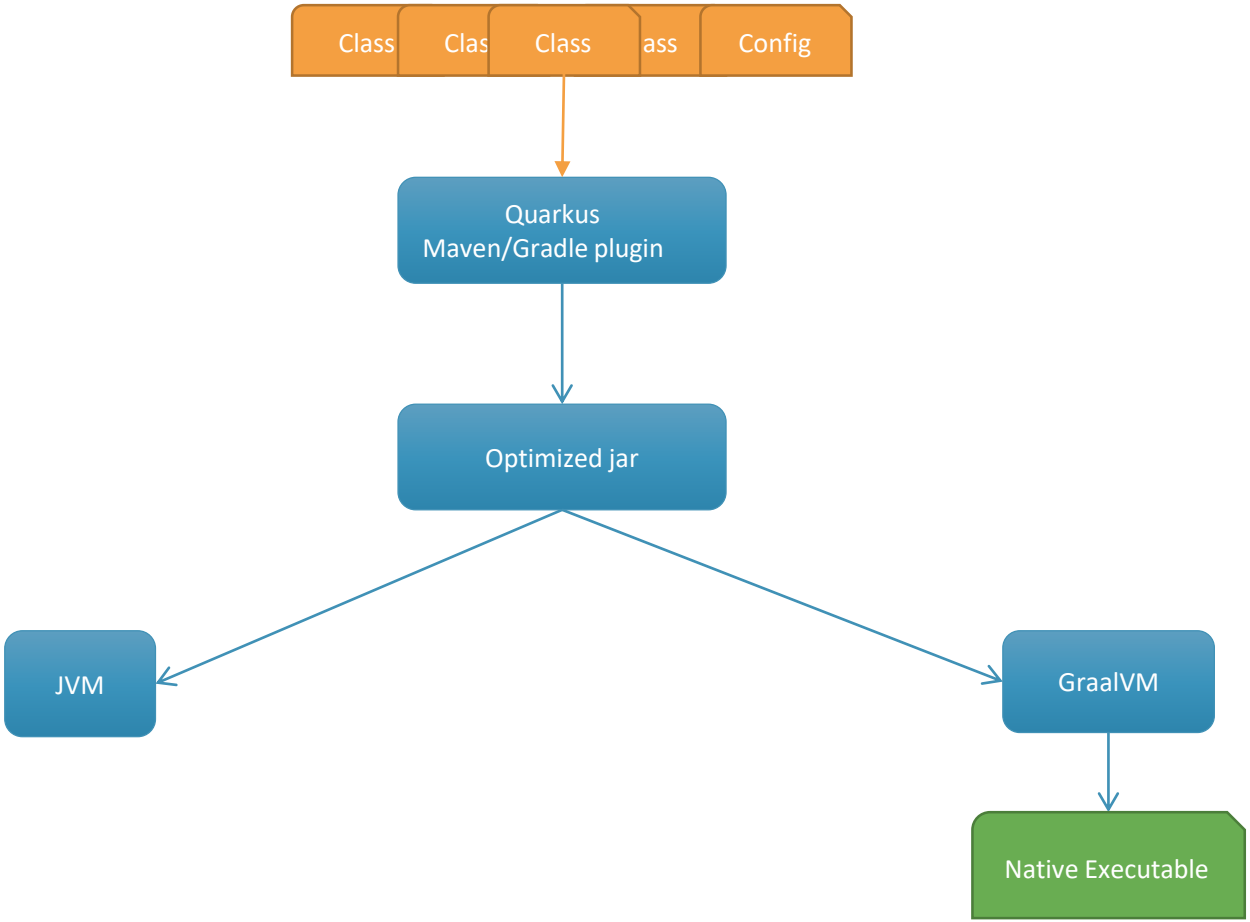
Quarkus Benchmarks III

- Startzeit
  - Lazy Loading
  - Zeit zum ersten Request

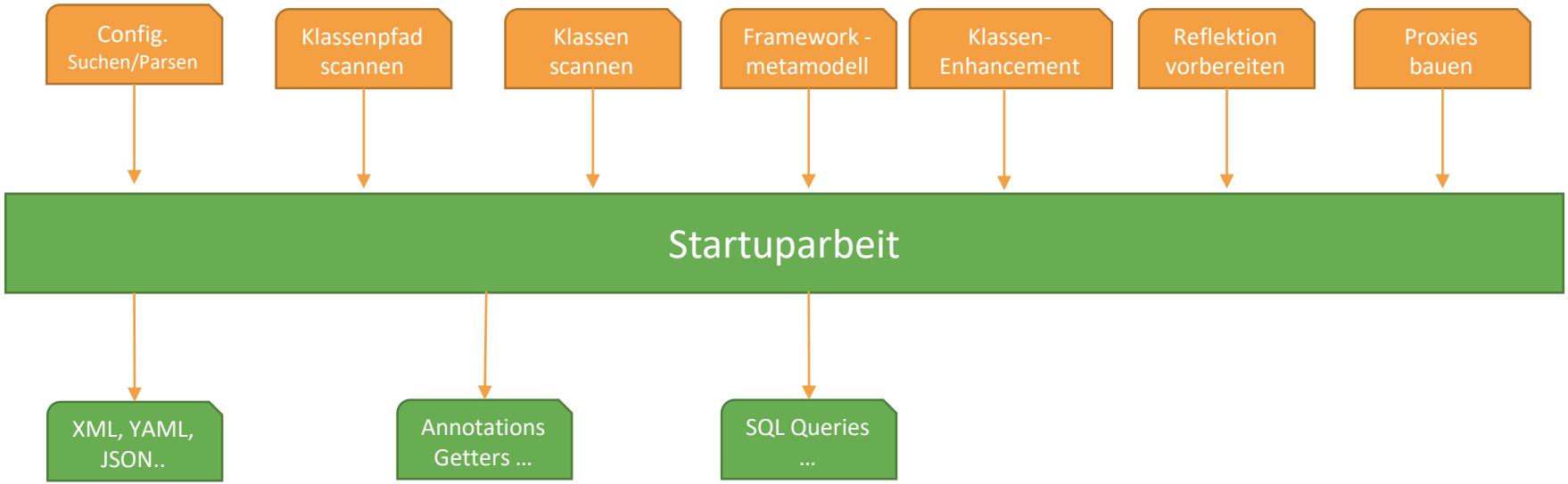


Quelle: https://quarkus.io/

Wie funktioniert Quarkus?



Startuparbeit



Vorteil Startuparbeit verschieben

part of trivadis group

- Die Arbeit wird einmal gemacht
- Die Bootstrap-Klassen werden nicht geladen
- Kurze Startzeiten
- Weniger Speicherverbrauch
- Wenig bis keine Reflektion
- Keine dynamischen Proxies
- Output: Bytecode

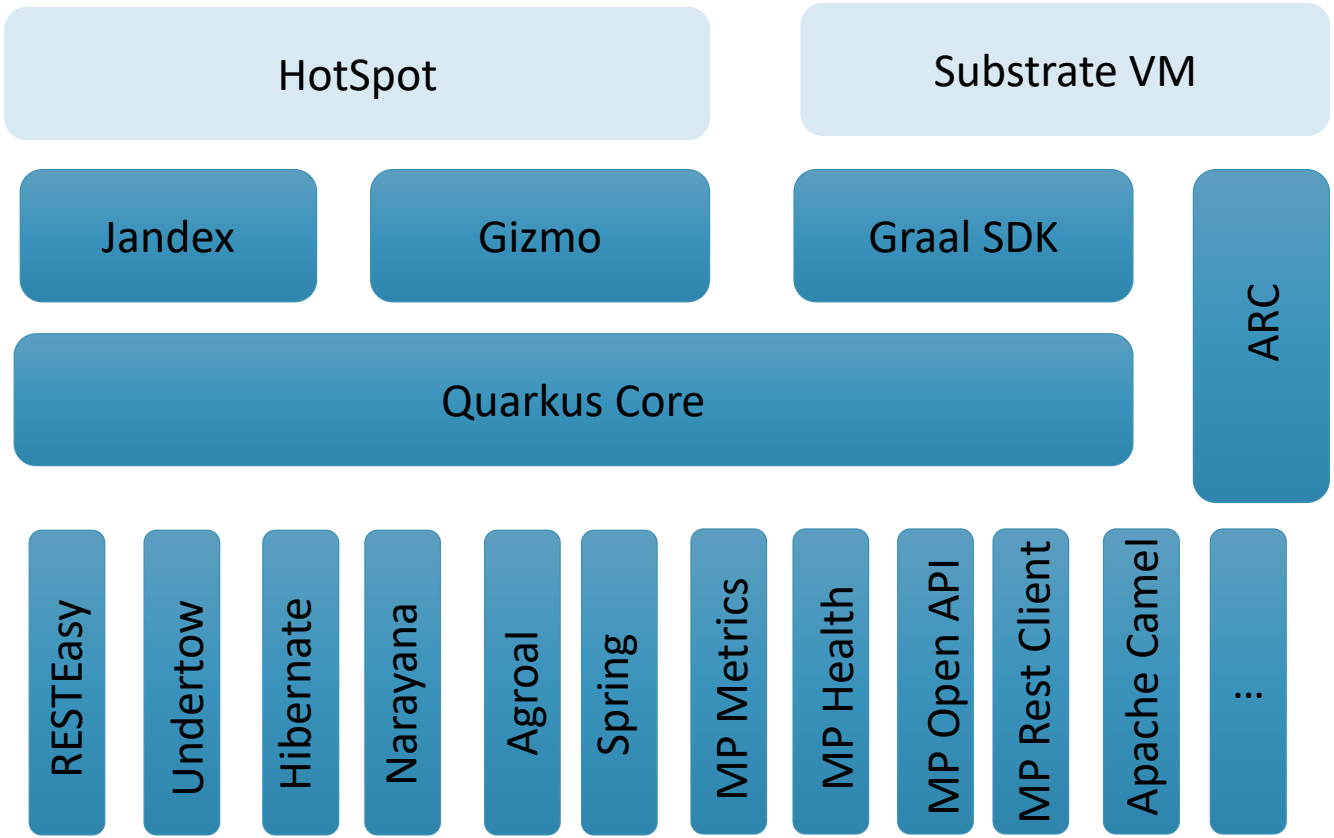
Extensions

- Jedes Framework bzw. Bibliothek braucht eine eigene Extension
- Machen den Code leichtgewichtig
- Machen den Code bereit für die GraalVM
- Physische Unterscheidung zwischen:
  - Deployment Maven Model
  - Runtime Maven Model
- Jandex
  - Hochperformante „classpath scanner and indexer“
  - Vermeidet die Initialisierung von Klassen
- ARC
  - CDI basierte DI zu Buildtzeit
- GIZMO
  - Bibliothek zur Bytecode-Generierung
- Custom Extensions möglich

Designfolgen

- Es werden weniger Klassen geladen
- Bootstrap-Klassen werden nicht zur Verfügung gestellt
- Aufwand wird nicht bei jedem Containerboot wiederholt
- Einfacher in GraalVM native Images laufen zu lassen
- Developer‘s Joy

Architektur



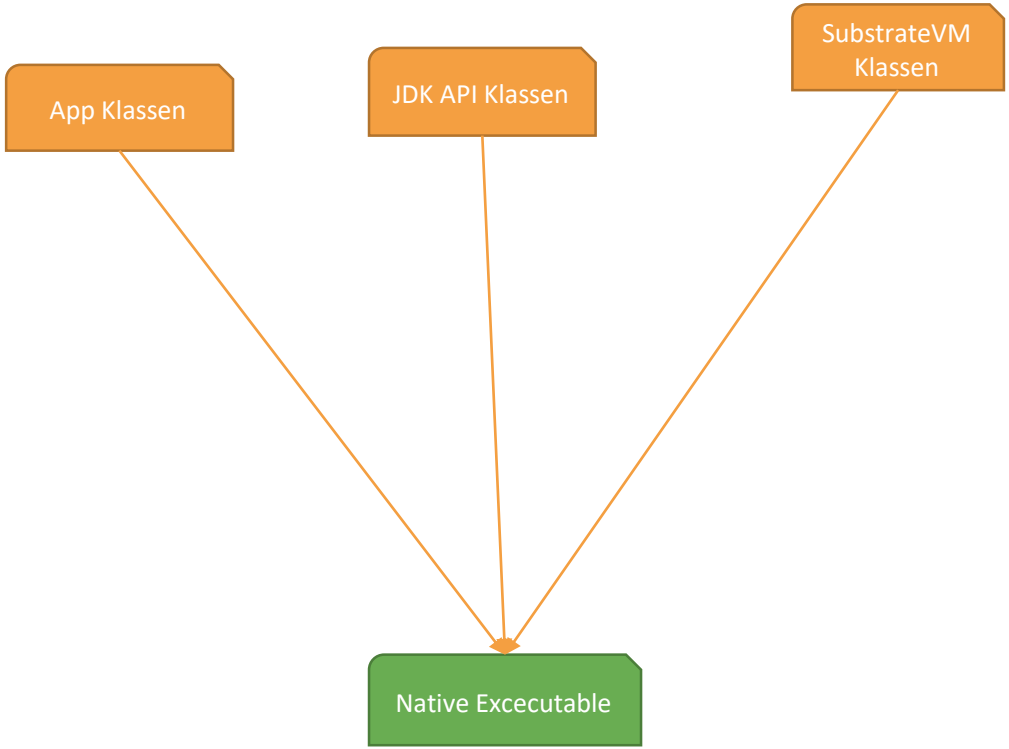
GraalVM

- Graal Compiler
  - AOT oder
  - JIT
- GraalVM – polyglotte VM

- Quarkus
  - GraalVM = Graal Compiler(AOT) + SubstrateVM
  - ```
    mvn package -Pnative
    ```


* Ja es gibt auch noch: Truffle Framework, Sulong, …
    ```

Ahead-of-time Compilierung



Quarkus - GraalVM

part of trivadis group

- Wann Native Image?
  - Schnelles Starten
    - Serverless Functions
    - Microservices

- Was macht der Start schneller?
  - Kein Classloading
  - Kein interpretierter Code
  - Einfacher Garbage Collector
  - Keine CPU-Zeit für Profiling and JIT-Arbeiten
  - Generierung Image-Heaps während der Erstellung des nativen Images

Quarkus - GraalVM

- Niedrigerer Speicherbedarf
  - Keine Metadaten für die geladenen Klassen
  - Keine Profilingdaten für die JIT-Optimierung
  - Kein interpretierter Code
  - Keine JIT-Strukturen

Quarkus - GraalVM

- Der Preis für native Image
  - Kein JVMTI
  - Keine Java Agents
  - Kein JMX
  - JFR Support

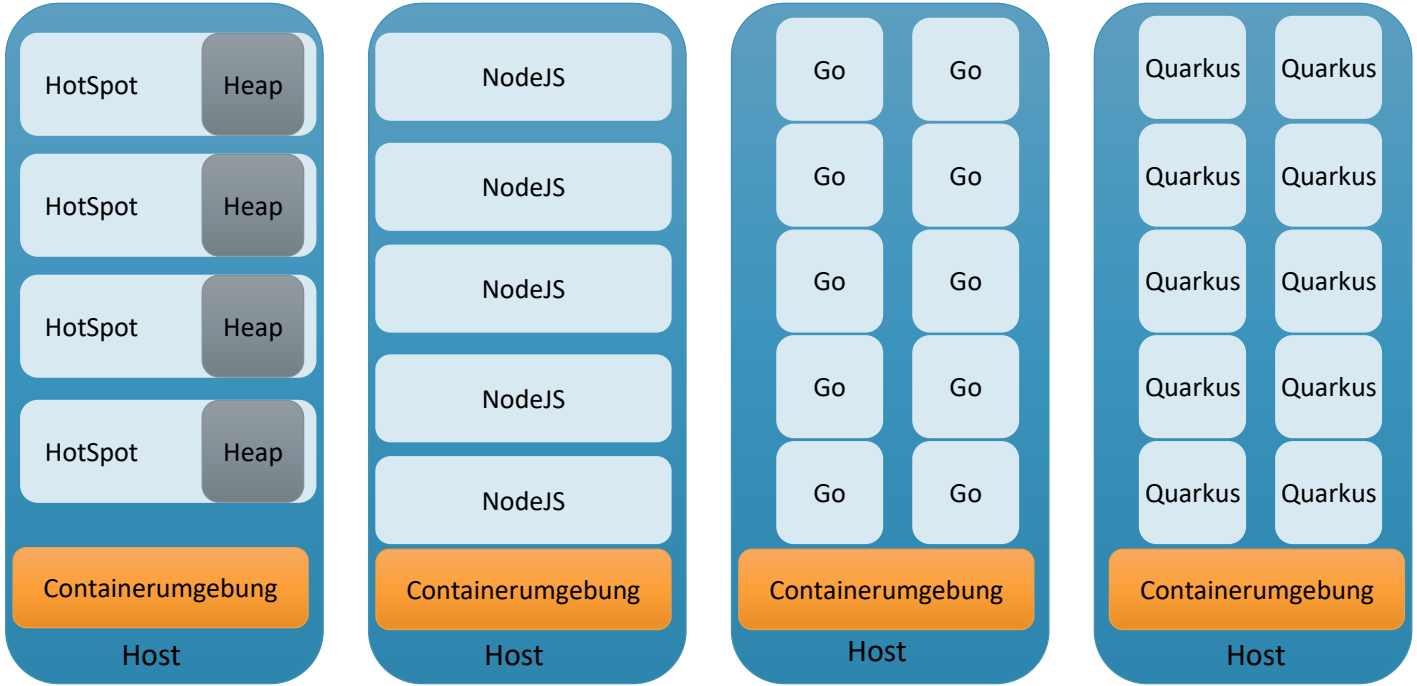
- Effizient nur für kleinere Heaps

- Native Code ist nicht „vollständig“ effizient

- Reflection

- Keine Unterstützung für Threads und Heap Dumps

Enterprise Java und Containers



Ausblick

part of trivadis group

JIT - OpenJDK	AOT – GraalVM native image
Hohe Speicherdichte	Höchste Speicherdichte
Hohe request/s/MB	Höchste request/s/MB
Schnelle Startzeit	Schnellste Startzeit
Beste Raw Performance(JIT vorhanden)	
Bester Garbage Collector	
Nutzung größerer Heaps	
Bekannte Monitoring Tools	
Compile once, Run anywhere	
Bibliotheken, die nur auf Standard JDK laufen	


Fragen?


Vielen Dank für Ihre
Aufmerksamkeit!