

Cloud native für den Enterprise Java Developer mit Quarkus

Version: 1.0

BASEL | BERN | BRUGG | BUKAREST | DÜSSELDORF | FRANKFURT A.M. | FREIBURG I.B.R. | GENÈVE
HAMBURG | KOPENHAGEN | LAUSANNE | MANNHEIM | MÜNCHEN | STUTT GART | WIEN | ZÜRICH

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

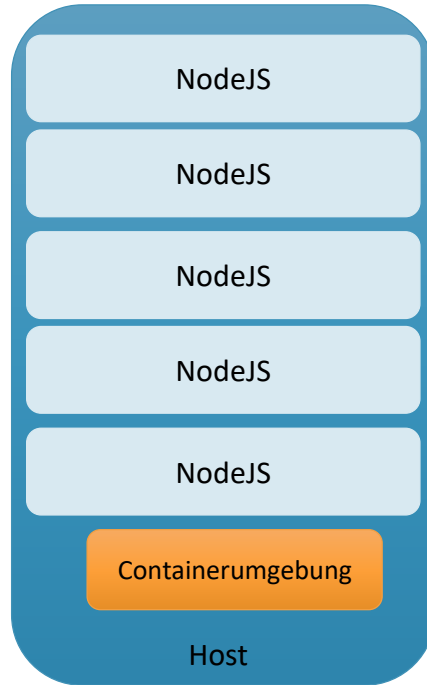
Monolith



Enterprise Java und Container

- Konzipiert für schnellen Durchsatz (Requests/s)
- Lange Startzeiten
- Speicher-Overhead
 - große JDK, Framework, App

Enterprise Java und Container



Was ist Quarkus?

- 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?

- 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

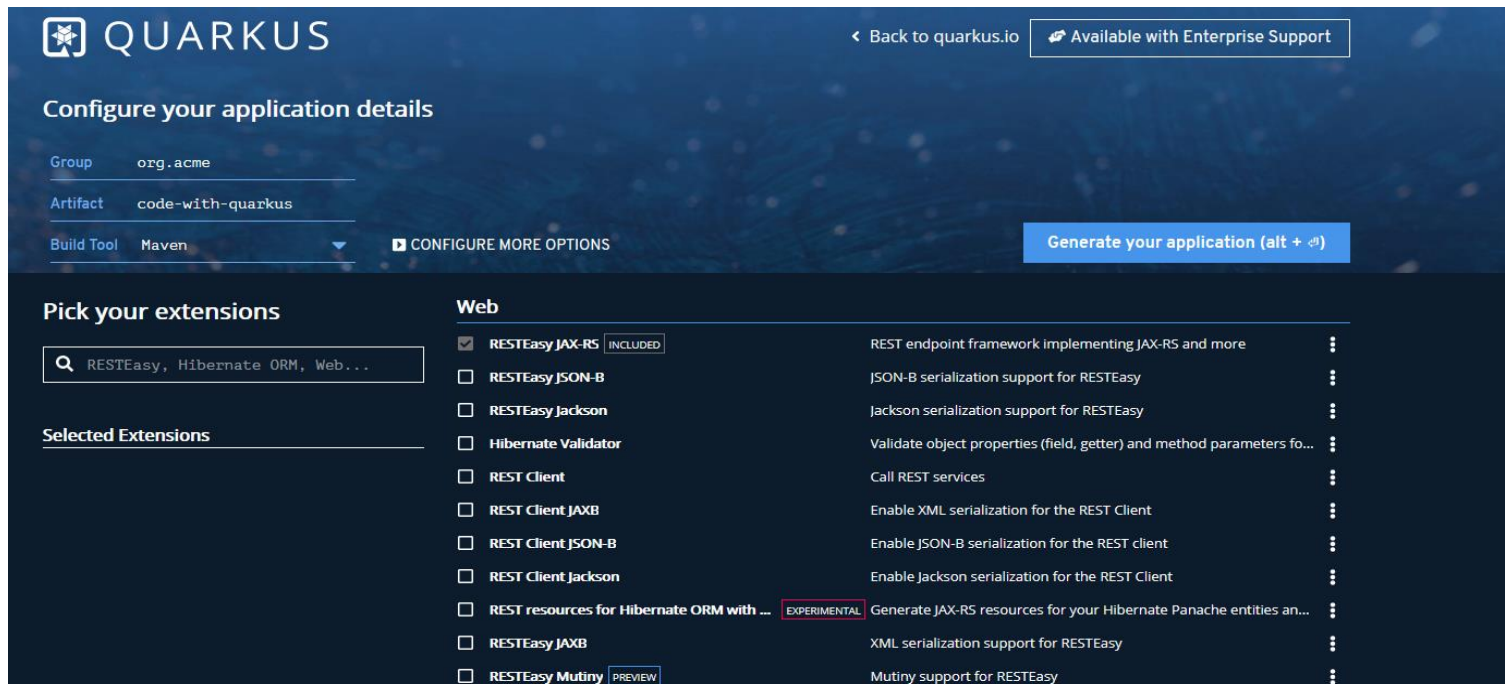
Was ist Quarkus?

- 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



The screenshot shows the Quarkus web application generator interface. At the top, there is a navigation bar with the Quarkus logo, a link to "Back to quarkus.io", and a button for "Available with Enterprise Support". Below this is a section titled "Configure your application details" with input fields for "Group" (org.acme), "Artifact" (code-with-quarkus), and "Build Tool" (Maven). A "CONFIGURE MORE OPTIONS" button and a "Generate your application (alt + ⌘)" button are also present.

The "Pick your extensions" section is active, showing a search bar with "RESTEasy, Hibernate ORM, Web..." and a list of "Selected Extensions". The "Web" section is expanded, showing a list of extensions with checkboxes and descriptions:

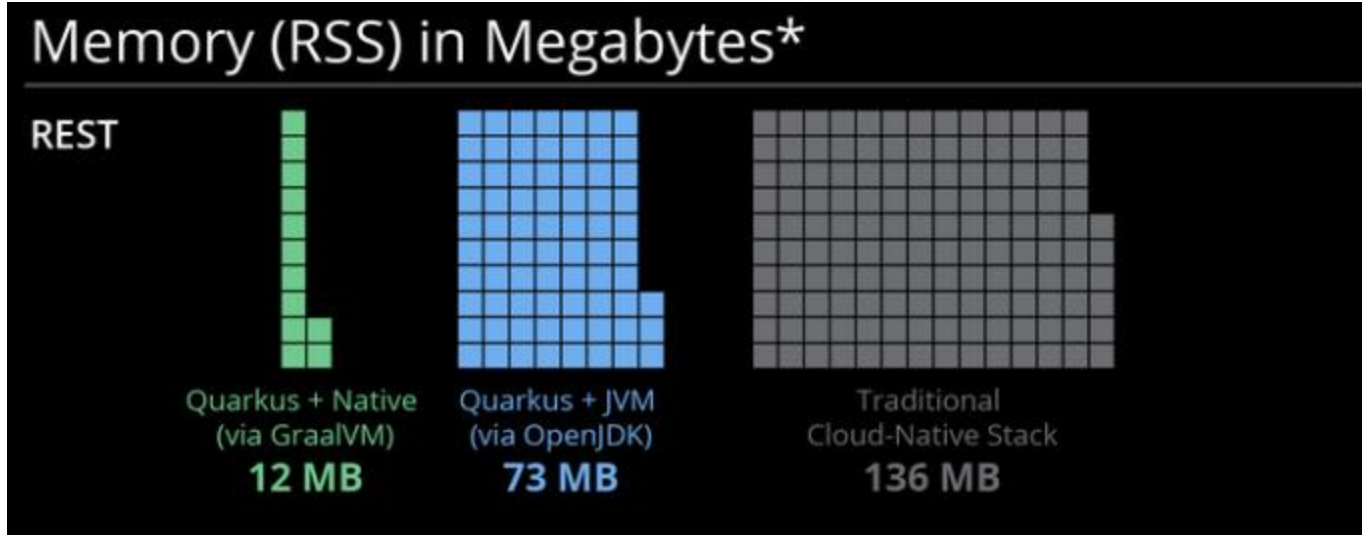
Extension	Status	Description
<input checked="" type="checkbox"/> RESTEasy JAX-RS	INCLUDED	REST endpoint framework implementing JAX-RS and more
<input type="checkbox"/> RESTEasy JSON-B		JSON-B serialization support for RESTEasy
<input type="checkbox"/> RESTEasy Jackson		Jackson serialization support for RESTEasy
<input type="checkbox"/> Hibernate Validator		Validate object properties (field, getter) and method parameters fo...
<input type="checkbox"/> REST Client		Call REST services
<input type="checkbox"/> REST Client JAXB		Enable XML serialization for the REST Client
<input type="checkbox"/> REST Client JSON-B		Enable JSON-B serialization for the REST client
<input type="checkbox"/> REST Client Jackson		Enable Jackson serialization for the REST Client
<input type="checkbox"/> REST resources for Hibernate ORM with ...	EXPERIMENTAL	Generate JAX-RS resources for your Hibernate Panache entities an...
<input type="checkbox"/> RESTEasy JAXB		XML serialization support for RESTEasy
<input type="checkbox"/> RESTEasy Mutiny	PREVIEW	Mutiny support for RESTEasy

Developer Joy

- 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

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



Quelle: <https://quarkus.io/>

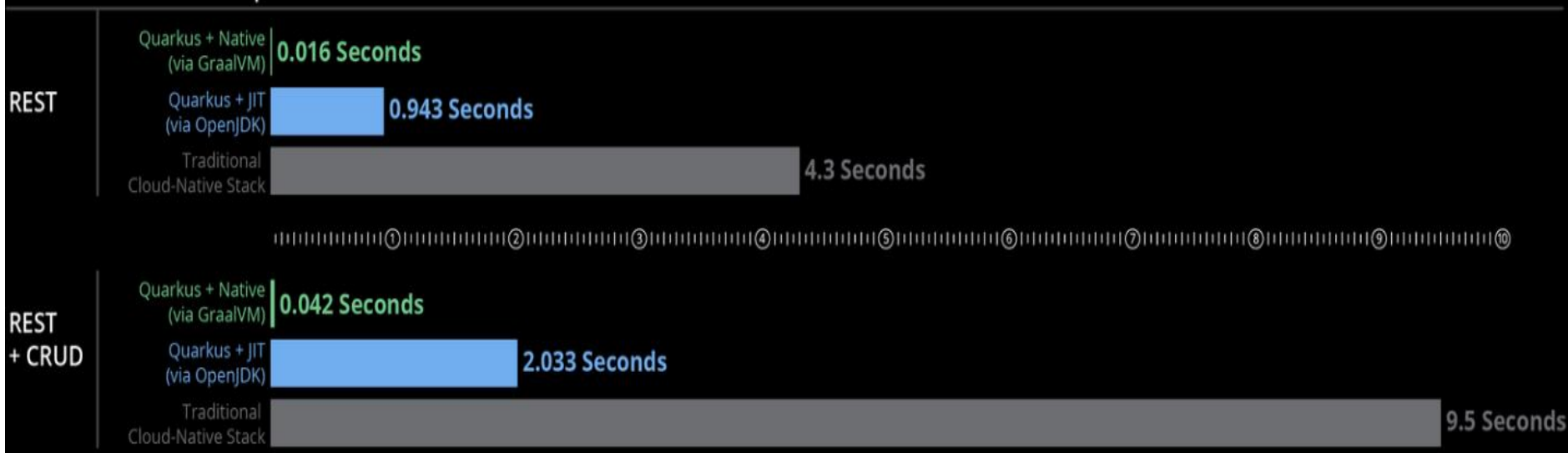
Quarkus Benchmarks II



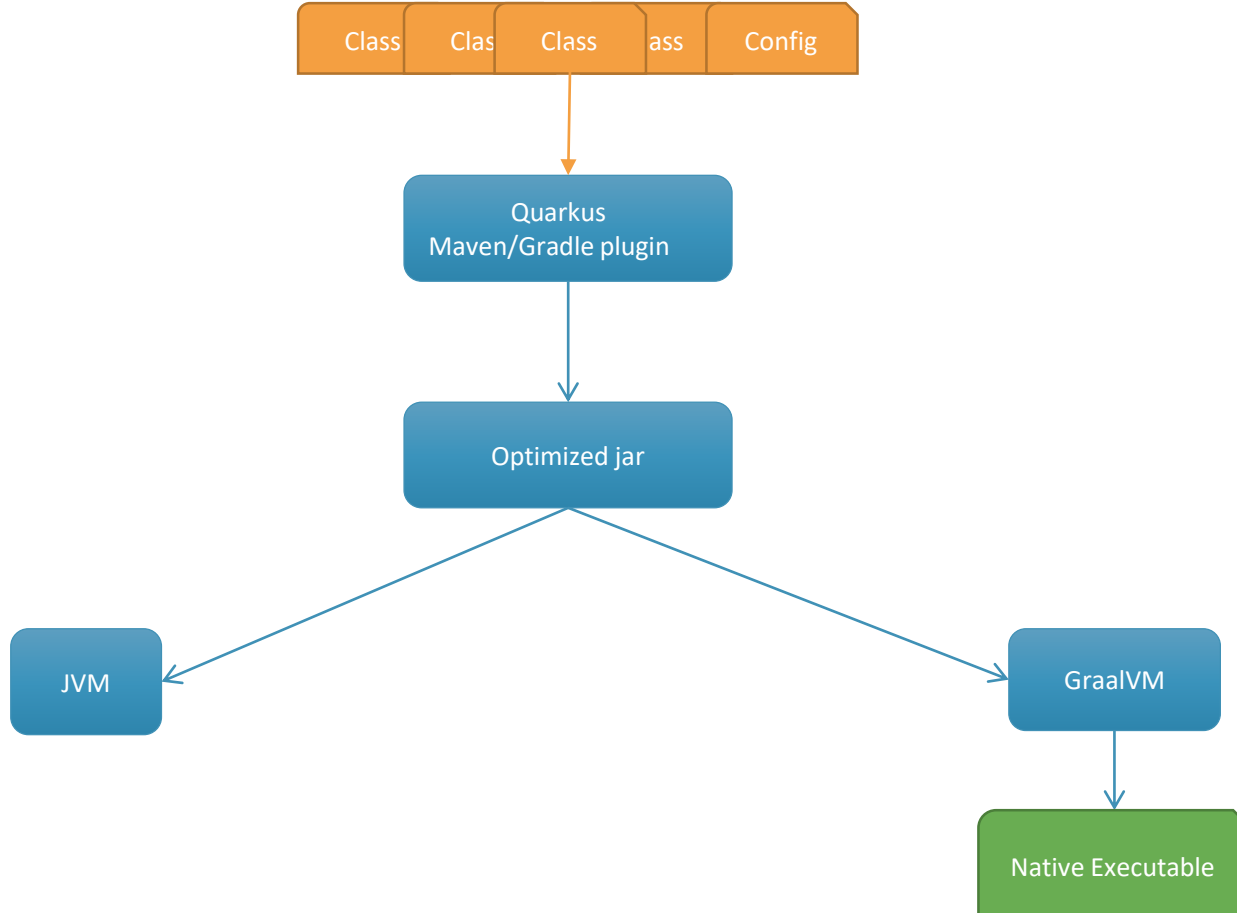
Quarkus Benchmarks III

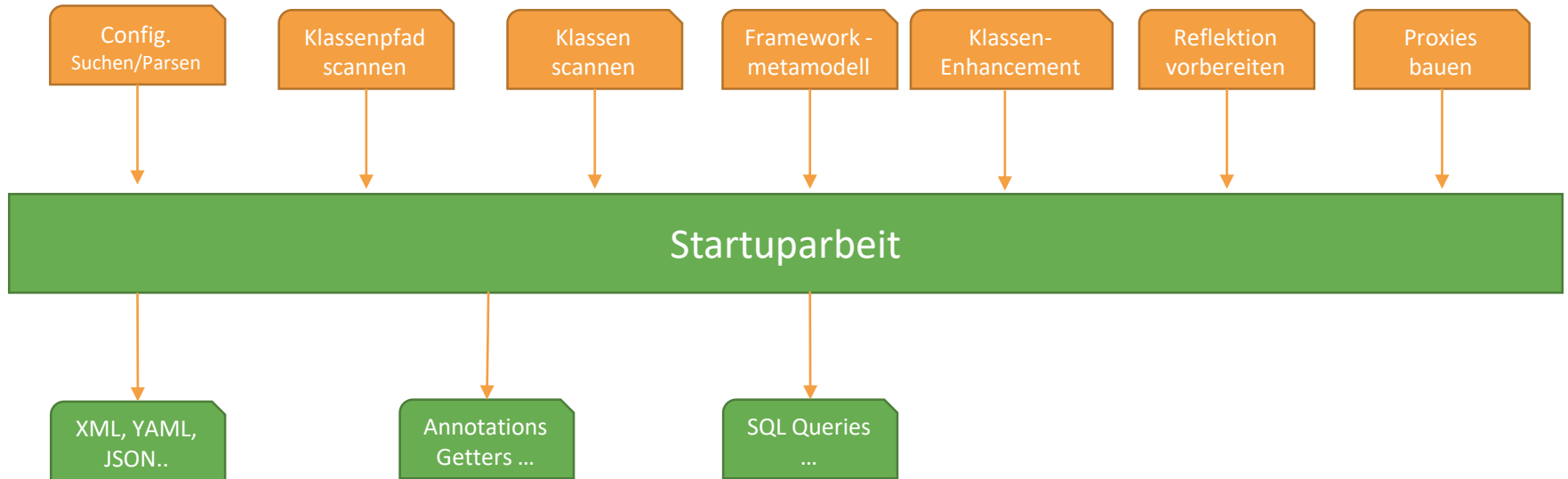
- Startzeit
 - Lazy Loading
 - Zeit zum ersten Request

BOOT + First Response Time



Wie funktioniert Quarkus?





Vorteil Startuparbeit verschieben

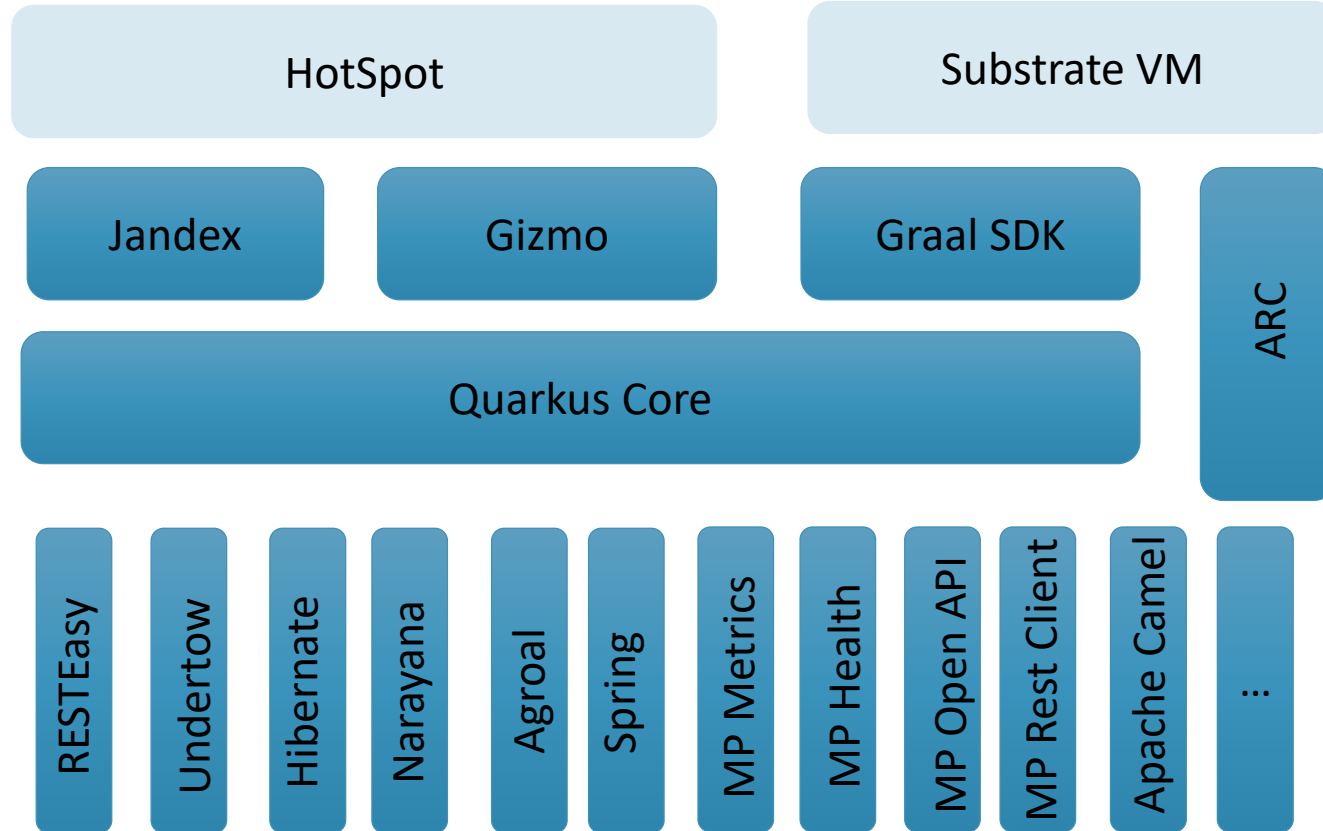
- Die Arbeit wird einmal gemacht
- Die Bootstrap-Klassen werden nicht geladen
- Kurze Startzeiten
- Weniger Speicherverbrauch
- Wenig bis keine Reflektion
- Keine dynamischen Proxies
- Output: Bytecode

- 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 Buildzeit
- 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

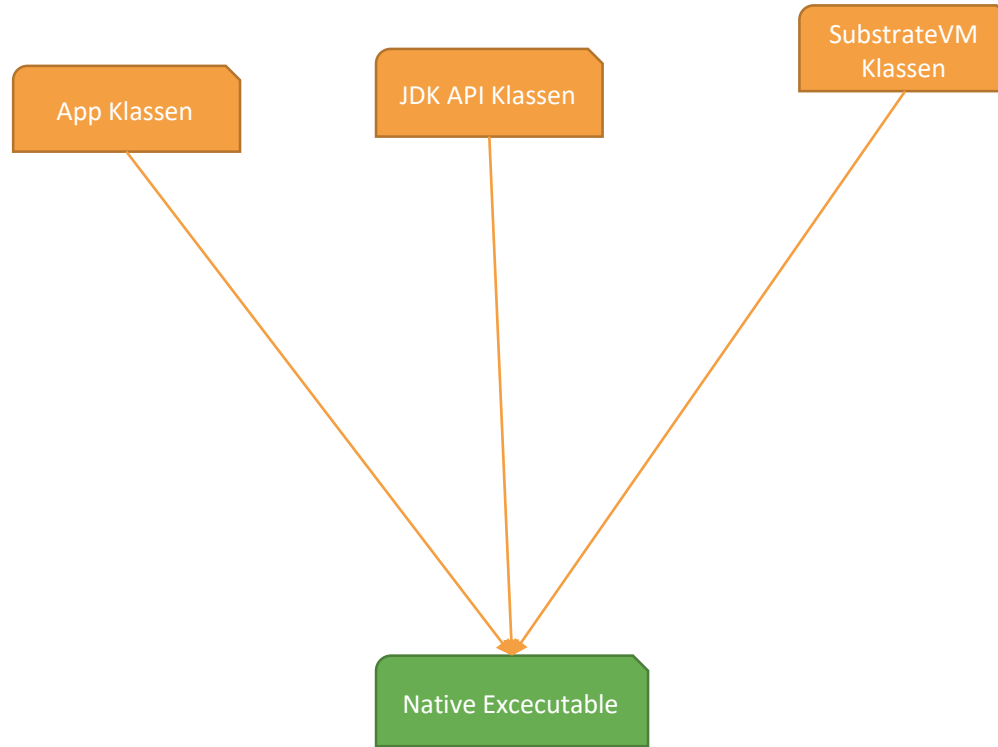
Architektur



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

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

Ahead-of-time Compilierung



- 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

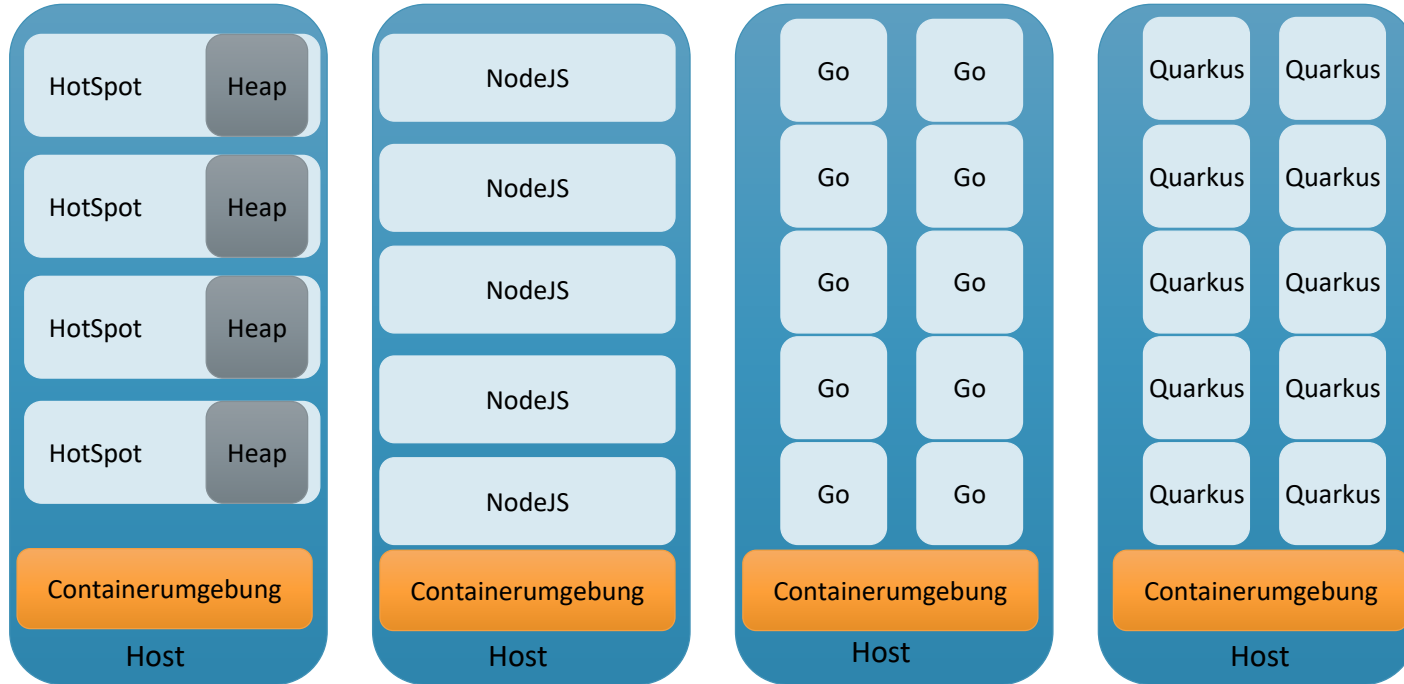
Quarkus - GraalVM

- 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



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?

BASEL | BERN | BRUGG | BUKAREST | DÜSSELDORF | FRANKFURT A.M. | FREIBURG I.B.R. | GENÈVE
HAMBURG | KOPENHAGEN | LAUSANNE | MANNHEIM | MÜNCHEN | STUTTGART | WIEN | ZÜRICH



**Vielen Dank für Ihre
Aufmerksamkeit!**

BASEL | BERN | BRUGG | BUKAREST | DÜSSELDORF | FRANKFURT A.M. | FREIBURG I.B.R. | GENÈVE
HAMBURG | KOPENHAGEN | LAUSANNE | MANNHEIM | MÜNCHEN | STUTTGART | WIEN | ZÜRICH