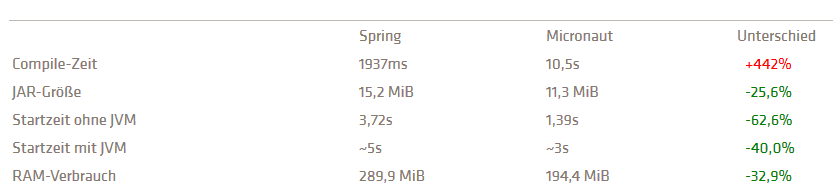
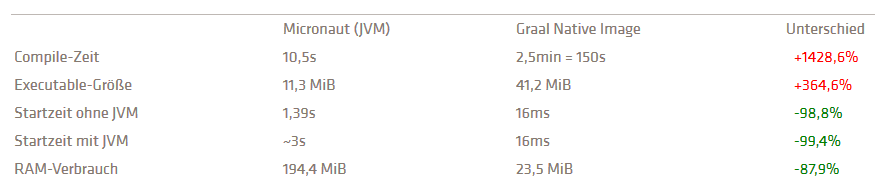
Micronaut

Falk Sippach, JUG Essen, 18.02.2020

* micronaut Konkurrent zu Quarkus, ktor, u.a.
* 8 Irrtümer verteilter Systeme
  + Das Netzwirk ist immer verfügbar
  + Die Latenz ist null
  + Die Brandbreite ist unendlich
  + Netzwerk ist sicher
  + DIe Topologie ändert sich nicht
  + Es gibt einen Admin
  + Die Transportkosten sind null
  + Das Netzwerk ist homogen
* micronaut ist konzipiert für Microservices, Cloud und Serverless
  + schnelle Startzeiten (<= 800ms)
  + geringer Speicherverbrauch
  + Kleine Deployment Artefakte (~20 MB)
  + Wenig externe Abhängigkeiten (d.h. geringer Dependency Download)
* Sprachen: Java, Groovy, Kotlin
* Installation
  + sdk install micronaut 1.1.0
  + mn
  + mv create-app hello-world
    - Liefert Dockerfile, Maven pom, Gradle Build
  + ./gradlew oder ./mvnw
  + sdk: Package Manager für Linux, Windows, Mac; verwaltet mehrere Versionen parallel
* Feature-Liste (ähnlich wie <https://start.spring.io>)
* Verschieben der Magie von Laufzeit auf Compilezeit für
  + Konfiguration
  + Dependency Injection
    - verwendet Dagger (Nachfolger von Guice)
  + …
* kein Classpath Scanning beim Start der Anwendung
* Kein Laufzeit-Cache von Reflection-Infos
* längere Compile-Zeit!
* Auto Konfig ähnlich wie bei Spring Boot
* Profile
* Persistenz
  + JDBC/JPA/Hibernate
  + NoSQL
  + ORM: GORM (nur für Groovy)
  + Micronaut Data (ehemals Predator): analog zu Spring Data
* Cross-cutting concerns: Logging, Monitoring 🡪 per AOP
* Eingebaute AOP-Mechanismen
  + Validation (JSR-303: Hibernate Validator)
  + Caching
  + Retry
  + Circuit Breaker (Eigenimplementation)
* Timer/Scheduler
* HTTP-Server
  + basierend auf Netty
  + asynchrony, reaktiv
  + Default: application/json
  + Fehlerbehandlung
* HTTP-Client (deklarativ)
  + analog zu Feign
  + mn create-client
* Testing
  + Empfehlung: kein Mocking; E2E anstelle Unit-Tests (da Startzeit schnell ist)
  + @MicronautTest
* Cloud Unterstützung
  + Distributed Config (Consul, AWS Parameter Store)
  + Service Discovery (Consul, Eureka, …)
  + Load Balancing: Ribbon, …
  + …
* über src/main/resources/application.yml
* Heapverbrauch (Beispielanwendung)
  + Spring: 50 MB
  + micronaut: ~7 MB



* Micronaut + GraalVM: kompiliert viel langsamer, aber nochmals schnellere Startzeit, geringerer RAM-Verbraucht, …
  + alle Vorteile erhält man nur, wenn man sich an Bedingungen hält: wenig Dependencies, …



<https://www.adesso.de/de/news/blog/micronaut-eine-alternative-zu-spring-4.jsp>

* Nachteile
  + keine inkrementellen Builds
  + erschwertes Debugging (wg. generierter Klassen)
  + Builds sind maschinen-abhängig (ARM64, …)