Kubernetes Native Java and Eclipse MicroProfile (plus other things)

Mark Little, VP Engineering, Red Hat

Who am I?

- Research into fault-tolerant distributed systems since 1986
  - Arjuna, Argus, Isis/Horus, Emerald, Xerox, ...
  - DCE, DCOM, CORBA, HTTP, Web Services, ...
- Implemented world's first 100% Java transaction service in 1995
  - Yes, we're still using it today!
- Active in OMG, OASIS, W3C, JCP, GGF and others
  - Co-author of a number of specifications and standards
- Visiting Professor at Newcastle and Lyon
- Industry ...
  - Various startups
  - Bluestone Distinguished Engineer, HP Distinguished Engineer
  - JBoss CTO in 2009
- These days spend far too much time on conference calls, meetings etc!

Overview

- Why Java in the age of Kubernetes and Linux Containers?
  - Java is dead, right?
- Why did Enterprise Java need to change?
  - Kubernetes and Immutability
  - Incompatibilities with Java and Application Containers
- Eclipse MicroProfile
  - Eclipse Jakarta EE
- Quarkus and principles behind it may be a game changer
  - Optimised for immutable architectures
  - Can work in constrained environments, e.g., low memory footprint

Why is Java still important?

- Still a de-facto language for enterprise developers*
- Large skills base (7-10 million Java devs.)**
- Large and diverse ecosystem
  - Amazon, Fujitsu, Google, IBM, Microsoft, Netflix, Oracle, Pivotal, Red Hat, ...
- Large, resilient community
- Much more than just the language!
- The innovation continues
  - Eclipse MicroProfile
  - Eclipse Jakarta EE
  - Java SE faster schedule
  - SubstrateVM

Sources:

*Tiobe Index : https://www.tiobe.com/tiobe-index/

* IEEE Spectrum : https://spectrum.ieee.org/static/interactive-the-top-programming-languages-2018

I** SlashData - 7.6 million active Java developers (State of Developer Nation, 16th Edition, Q4 2018)

Historical Enterprise Java Stack

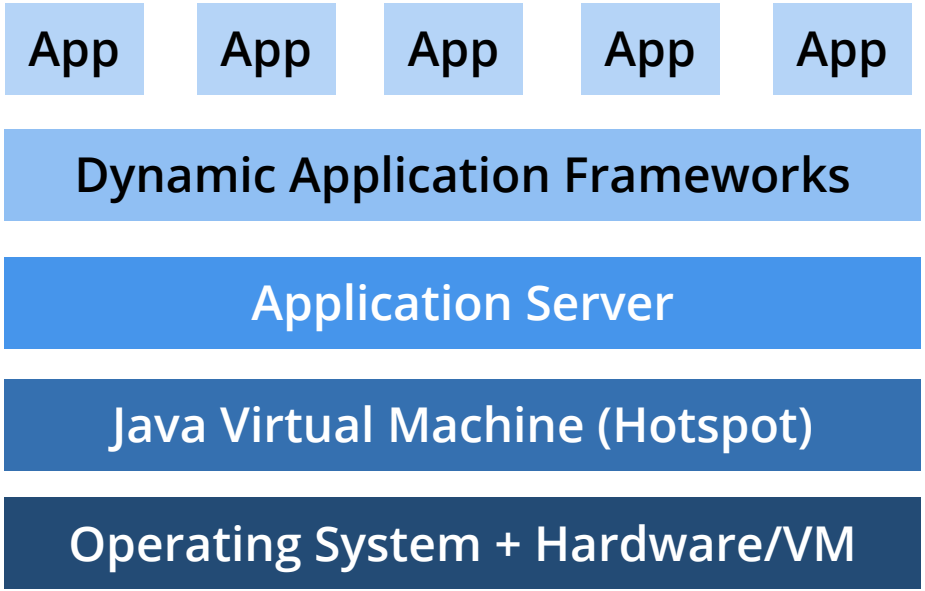
Architecture: Monoliths

Deployment: multi-app, appserver

App Lifecycle: Months

Memory: 1GB+ RAM

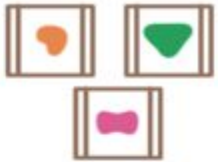
Startup Time: 10s of sec



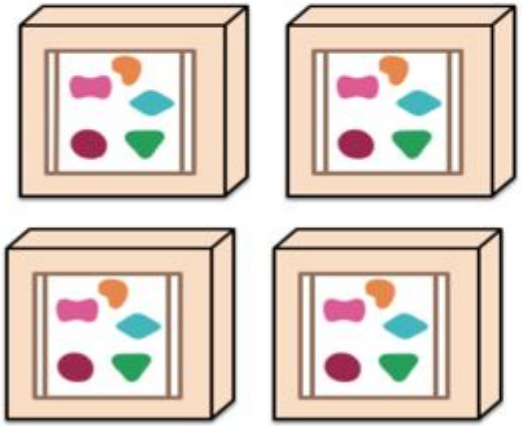
And then along came ...

A monolithic application puts all its functionality into a single process...

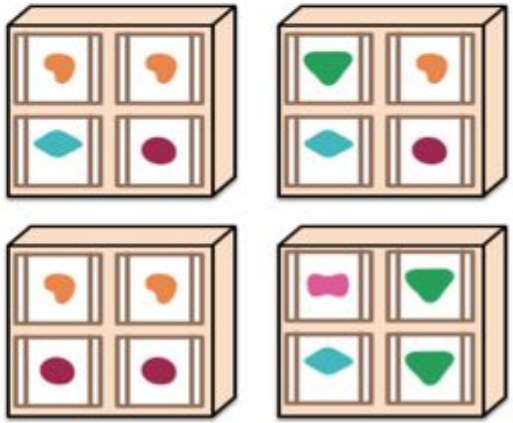
A microservices architecture puts each element of functionality into a separate service...



... and scales by replicating the monolith on multiple servers

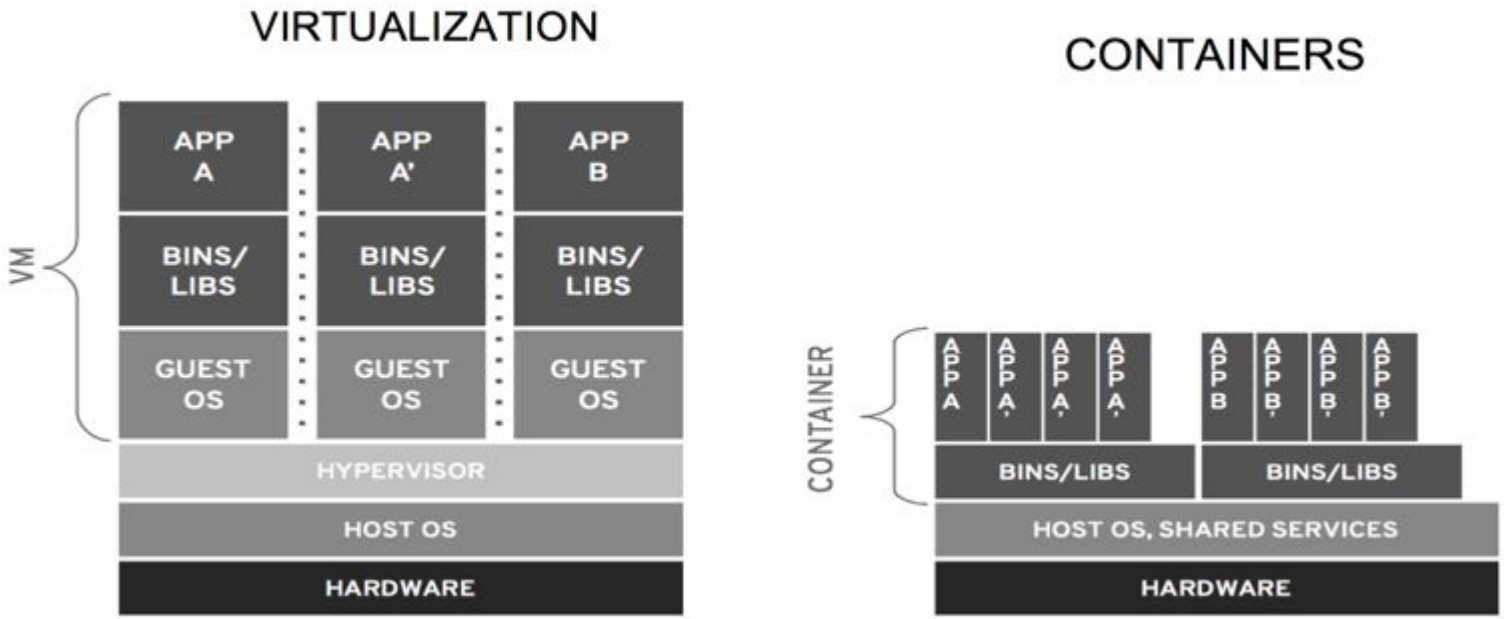


... and scales by distributing these services across servers, replicating as needed.



Monoliths are Evil? Microservices ..?

Followed closely by …

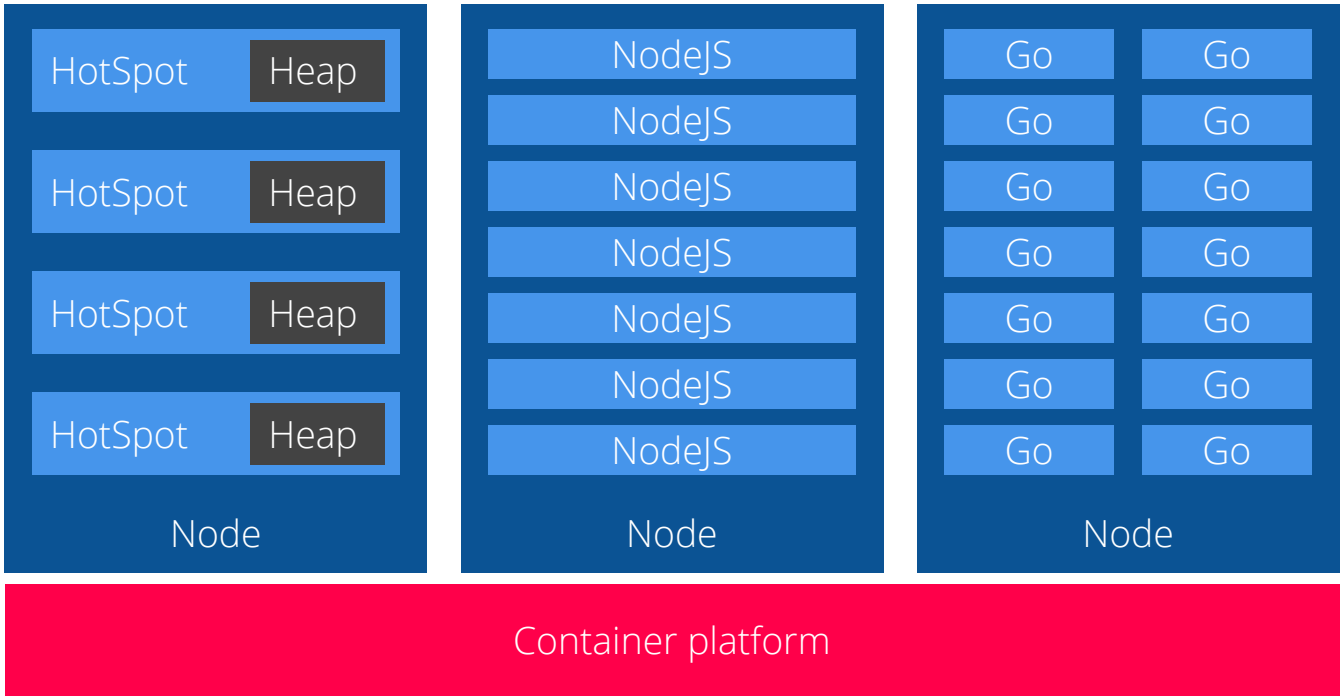


And then … Kubernetes

- Open source project from Google
- The de facto standard for cluster management for Linux containers
- Packages Orchestration, service discovery, load balancing
  - All behind a simple REST API
- Immutable architectures

The “Java Problem”

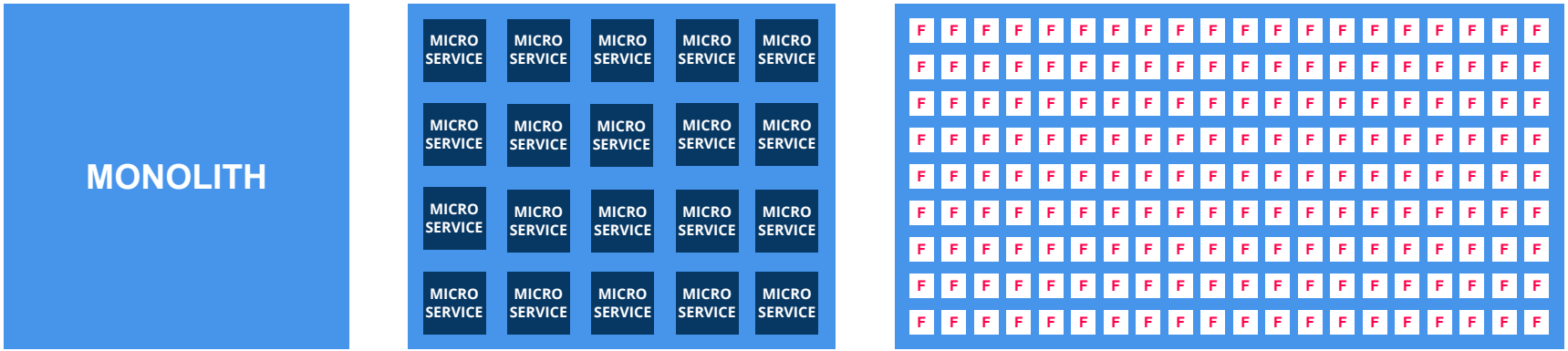
- Designed for throughput at the expense of footprint
- Intended to be long running, less focus on startup speed
- Rich dynamic behavior built around mutable bare-metal systems
  - … yet Linux containers are primarily immutable
  - … frameworks and stacks built to leverage key Java capabilities such as dynamism
- Java is trying to pivot (JPMS, AOT, Graal, etc)
  - … but architectural changes to frameworks are required to truly benefit


HotSpot
Heap
HotSpot
Heap
HotSpot
Heap
HotSpot
Heap
Node
NodeJS
NodeJS
NodeJS
NodeJS
NodeJS
NodeJS
NodeJS
Node
Go
Go
Go
Go
Go
Go
Go
Go
Go
Go
Go
Go
Go
Go
Node
Container platform

Why footprint matters in the cloud

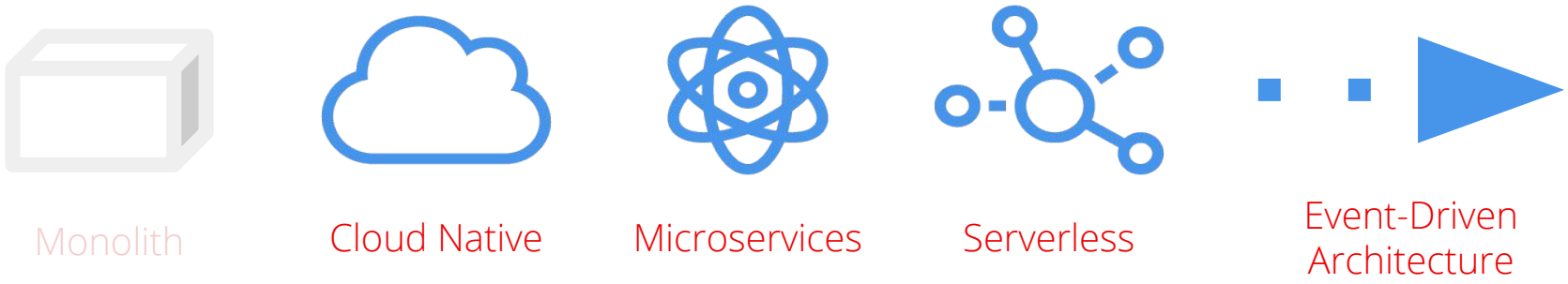
- Memory is more important than throughput on containers
  - It’s more expensive (requires permanence), unlike CPU cycles
- Microservices multiply overhead cost
  - One app becomes N microservices (e.g. 20 microservices ~= 20GB today!)
- If we do nothing Java alternatives will take over (eventually)
  - Go, Python, Node, PHP, Rust etc. do not have this problem

What we really want ...



- 1 monolith ≈ 20 microservices ≈ 200 functions
- Scale to 1 vs scale to 0
- Start up time

We need a Kubernetes native stack

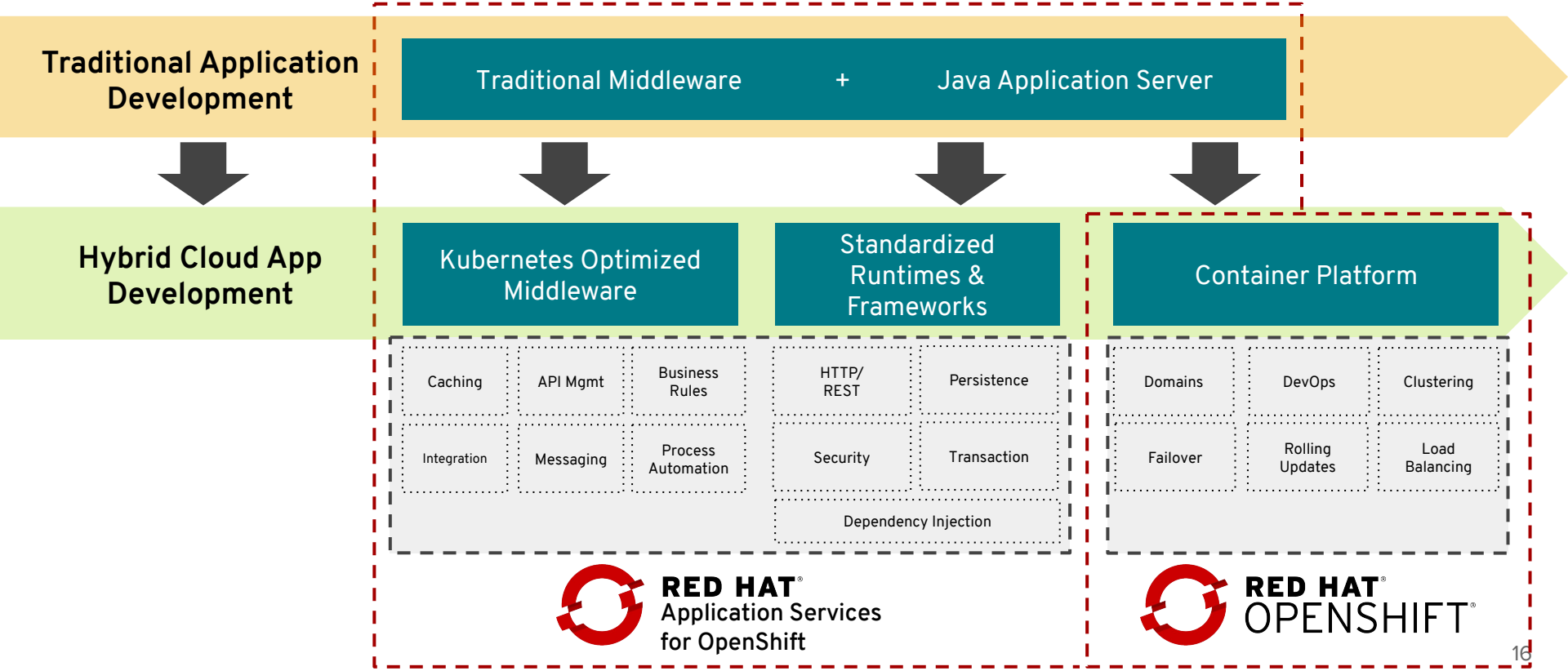


14

It starts with the JVM

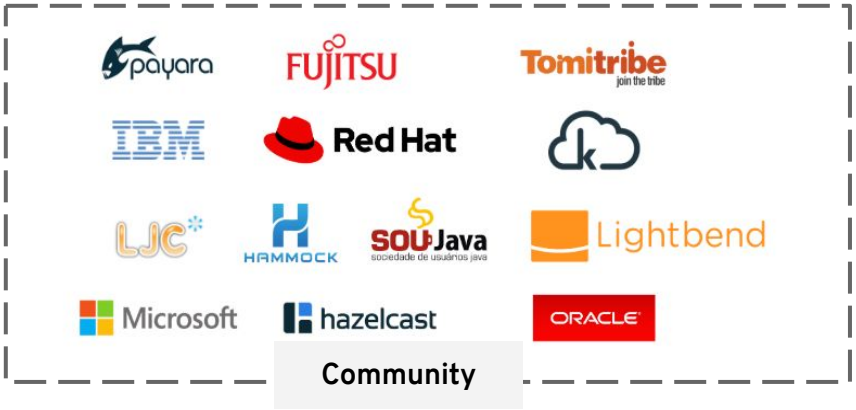
- Linux container aware JVM efforts
  - Memory utilisation
  - Processor utilisation
- OpenJDK evolved to work better with Linux Containers
  - Shenandoah GC
- Eclipse OpenJ9 performance improvements
  - JIT-as-a-Service
- Compiled Java?
  - gcj
  - ... Dalvik?
  - Avian
  - Excelsior JET
  - GraalVM
- JVM improvements are necessary but not sufficient

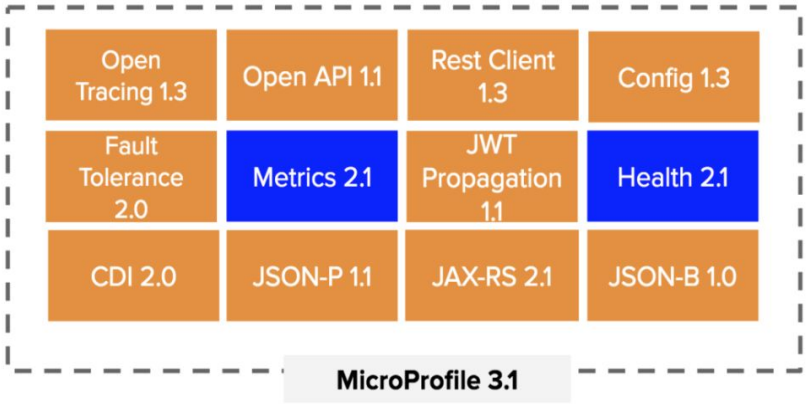
APPLICATION SERVER TRANSITION



Eclipse MicroProfile

- Open Source community specifications for Enterprise Java microservices
- 9 releases in 3 years
- 5 specifications in the pipeline



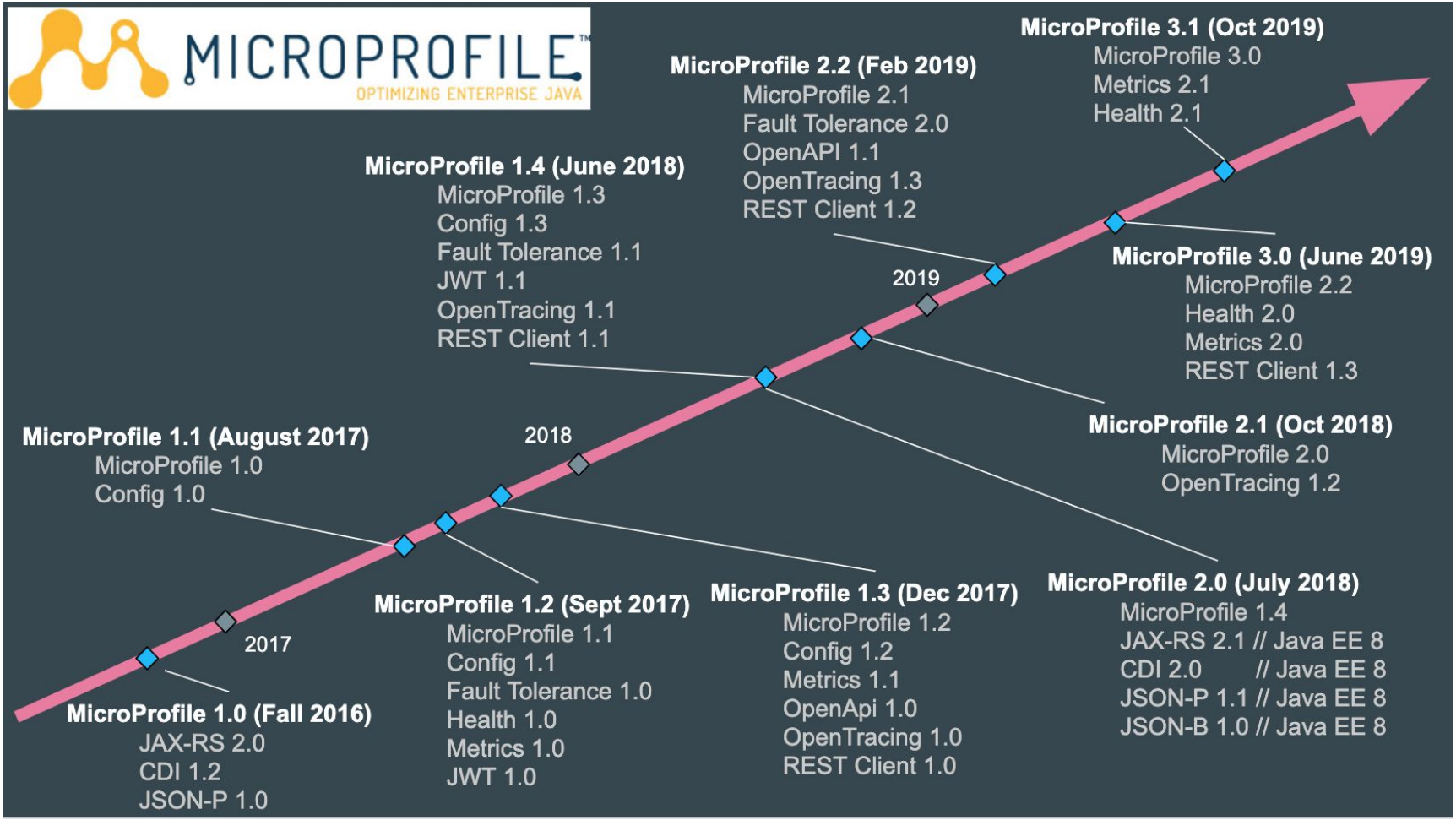


MICROPROFILE
OPTIMIZING ENTERPRISE JAVA

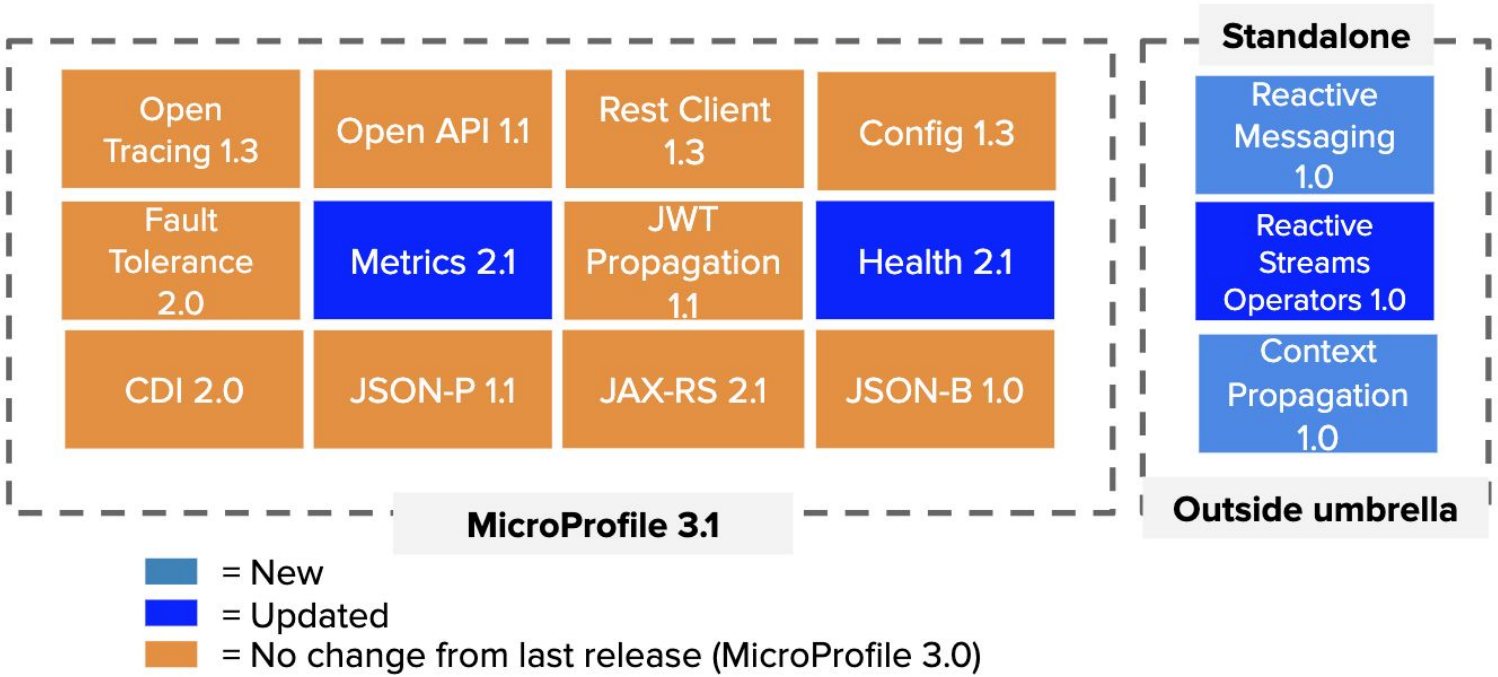
Projects Presentations FAQ Blog Contributors Join The Discussion See The Code

Eclipse MicroProfile

Optimizing Enterprise Java for a Microservices Architecture

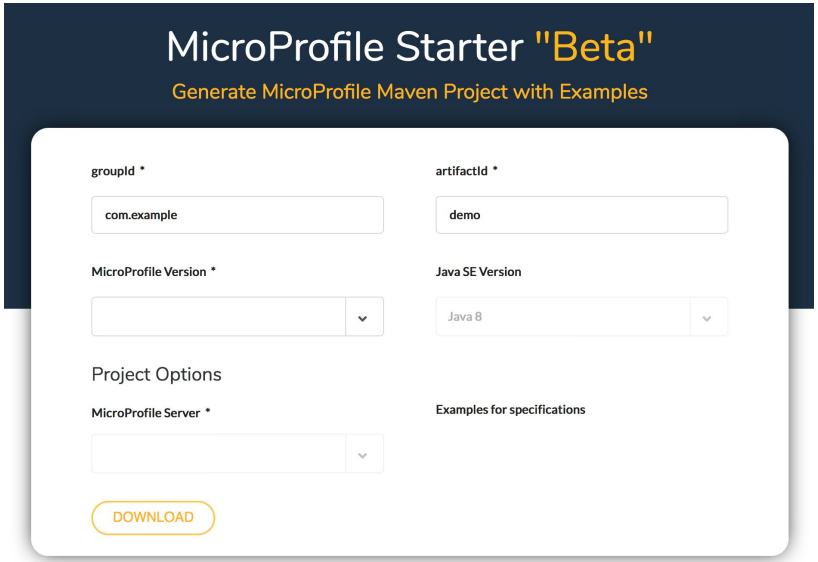

MICROPROFILE
OPTIMIZING ENTERPRISE JAVA
MicroProfile 3.1 (Oct 2019)
MicroProfile 3.0
Metrics 2.1
Health 2.1
MicroProfile 2.2 (Feb 2019)
MicroProfile 2.1
Fault Tolerance 2.0
OpenAPI 1.1
OpenTracing 1.3
REST Client 1.2
MicroProfile 1.4 (June 2018)
MicroProfile 1.3
Config 1.3
Fault Tolerance 1.1
JWT 1.1
OpenTracing 1.1
REST Client 1.1
2019
MicroProfile 3.0 (June 2019)
MicroProfile 2.2
Health 2.0
Metrics 2.0
REST Client 1.3
MicroProfile 2.1 (Oct 2018)
MicroProfile 2.0
OpenTracing 1.2
MicroProfile 1.1 (August 2017)
MicroProfile 1.0
Config 1.0
2018
MicroProfile 2.0 (July 2018)
MicroProfile 1.4
JAX-RS 2.1 // Java EE 8
CDI 2.0 // Java EE 8
JSON-P 1.1 // Java EE 8
JSON-B 1.0 // Java EE 8
MicroProfile 1.2 (Sept 2017)
MicroProfile 1.1
Config 1.1
Fault Tolerance 1.0
Health 1.0
Metrics 1.0
JWT 1.0
MicroProfile 1.3 (Dec 2017)
MicroProfile 1.2
Config 1.2
Metrics 1.1
OpenApi 1.0
OpenTracing 1.0
REST Client 1.0
2017
MicroProfile 1.0 (Fall 2016)
JAX-RS 2.0
CDI 1.2
JSON-P 1.0

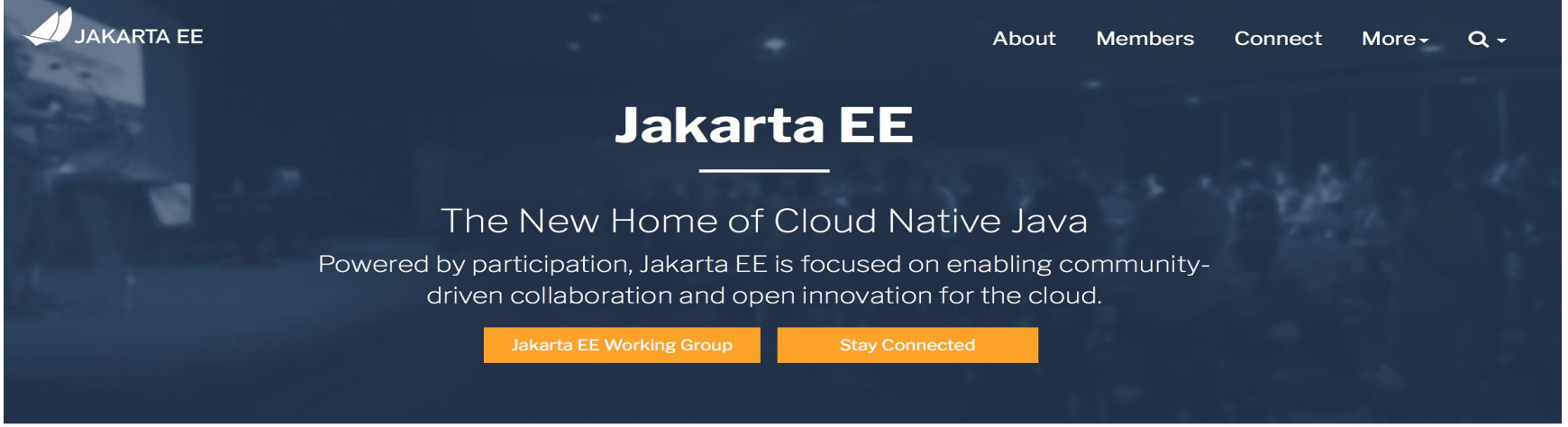
Eclipse MicroProfile 3.1 (Oct 2019)



Roadmap items

- Long Running Actions (yes, transactions for microservices!)
- GraphQL
- Reactive Relational Database Access
- Event Data
- Service meshes
- start.microprofile.io





Strategic Members

FUJITSU IBM ORACLE payara redhat Tomitribe

Participating Members

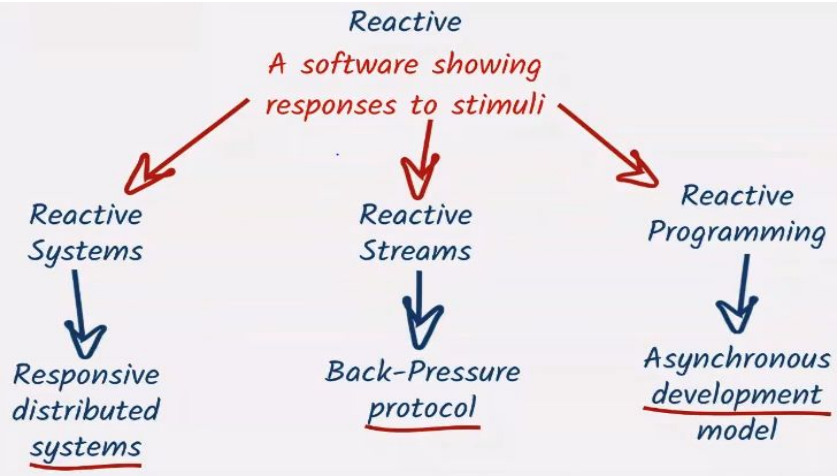
cloudbees DOCDOKU Genuitec IncQueryLabs Liferay Lightbend LJC

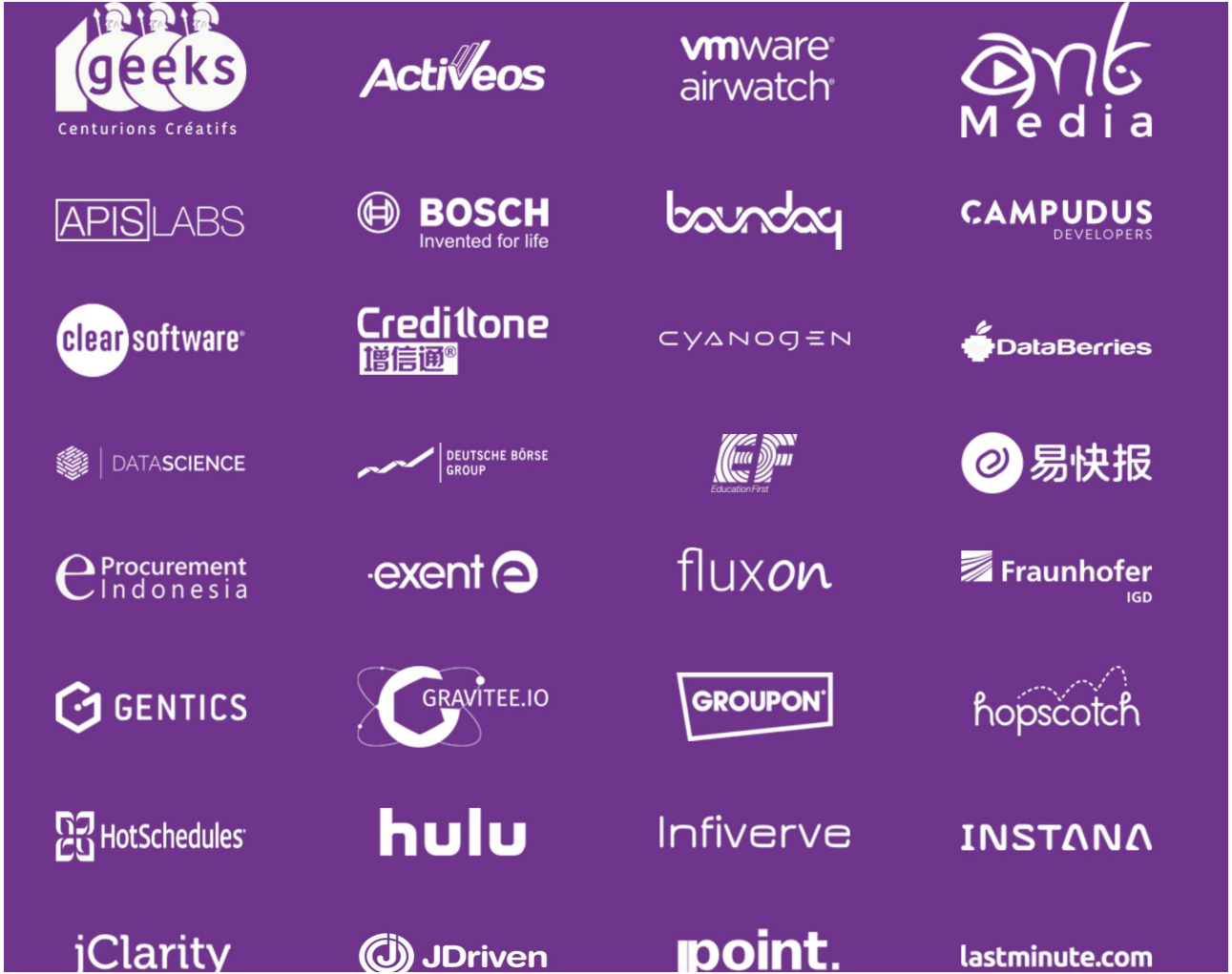
Microsoft MIZUHO Pivotal RCPVision SAP USEOPEN vaadin

webtide

Eclipse Vert.x

- Responsive: fast, is able to handle a large number of events / connections
- Elastic: scale up and down by just starting and stopping nodes, round-robin
- Resilient: failure as first-class citizen, self-healing
- Asynchronous message-passing: asynchronous non-blocking development model
- 2014 JAX Innovations Award Winner




geeks
Centurions Créatifs
ActiVeos
vmware airwatch
Media
APIS LABS
BOSCH
Invented for life
baunday
CAMPUDUS
DEVELOPERS
clear software
Creditone
增信通
CYANOGEN
DataBerries
DATASCIENCE
DEUTSCHE BÖRSE GROUP
Education First
易快报
eProcurement Indonesia
exent
fluxon
Fraunhofer IGD
GENTICS
GRAVITEE.IO
GROUPON
hopscotch
HotSchedules
hulu
Infiverve
INSTANA
jClarity
JDriven
point.
lastminute.com

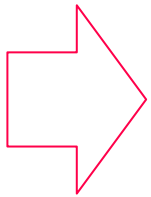
WHAT IS QUARKUS?

QUARK: elementary particle / **US**: hardest thing in computer science

Moving to Compile-Time Boot

What does a framework do at startup time?

- Parse config files
- Classpath & classes scanning
  - for annotations, getters or other metadata
- Build framework metamodel objects
- Prepare reflection and build proxies
- *Start and open IO, threads etc*

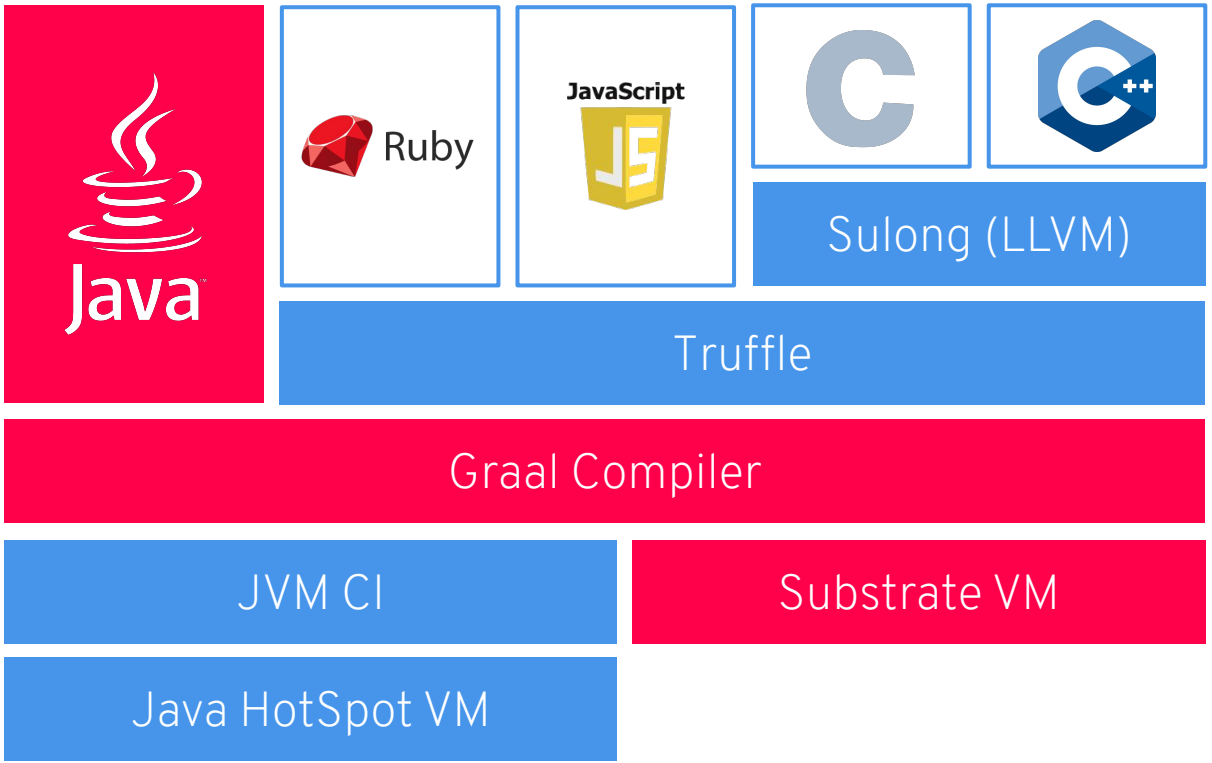


Framework Optimizations

- Moved as much as possible to build phase
- Minimized runtime dependencies
- Maximize dead code elimination
- Introduced clear metadata contracts
- Spectrum of optimization levels
  (all → some → no runtime reflection)

What about MicroProfile?

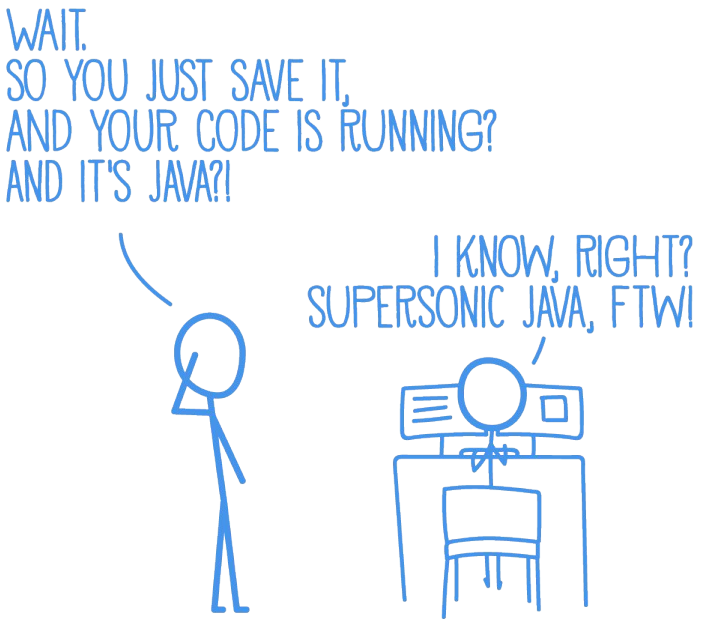
- Quarkus implements MicroProfile
- We all know the benefits of open standards ...
  - No vendor lock-in so applications can be ported across implementations
  - Don't like something then come in and help evolve it
- MicroProfile in Quarkus enables skills to be immediately brought to the problem
  - And applications from other implementations too!
- Aim to feed more innovations back to MicroProfile and beyond
  - Remember ... the JVM needs love too, not just frameworks


Java
Ruby
JavaScript
C
C++
Sulong (LLVM)
Truffle
Graal Compiler
JVM CI
Substrate VM
Java HotSpot VM

Benefit No. 1: Developer Joy

A cohesive platform for optimized developer joy:

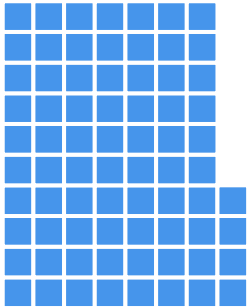
- Based on standards (e.g., **MicroProfile**), but not limited
- Unified configuration
- Zero config, live reload in the blink of an eye
- Streamlined code for the 80% common usages, flexible for the 20%
- No hassle native executable generation
- Unifies imperative and reactive programming
  - Vert.x FTW!
- Re-architected many projects
  - Hibernate, Narayana, Netty, Infinispan, ...



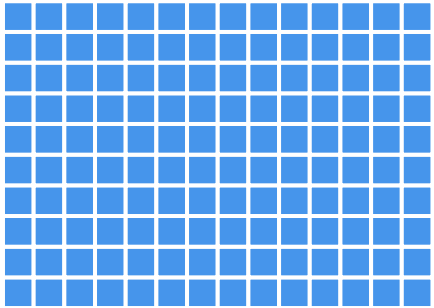
Benefit No. 2: Supersonic Subatomic Java

REST

Quarkus + SubstrateVM
13 MB



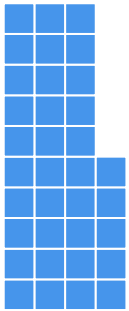
Quarkus + OpenJDK Hotspot
74 MB



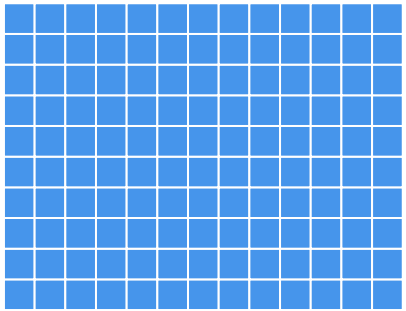
Traditional Cloud Native Stack
+ OpenJDK Hotspot
140 MB

Benefit No. 2: Supersonic Subatomic Java

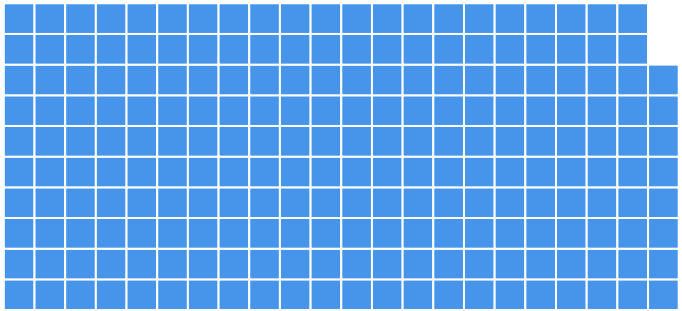
REST + CRUD



Quarkus + SubstrateVM
35 MB

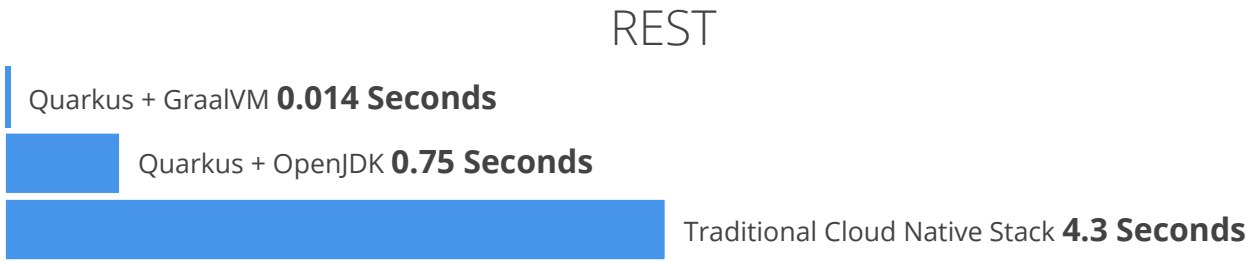


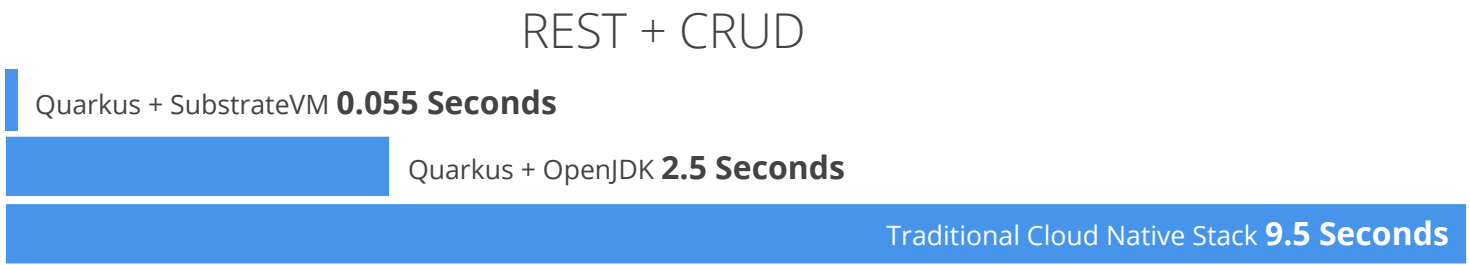
Quarkus + OpenJDK Hotspot
130 MB



Traditional Cloud Native Stack
+ OpenJDK Hotspot
218 MB

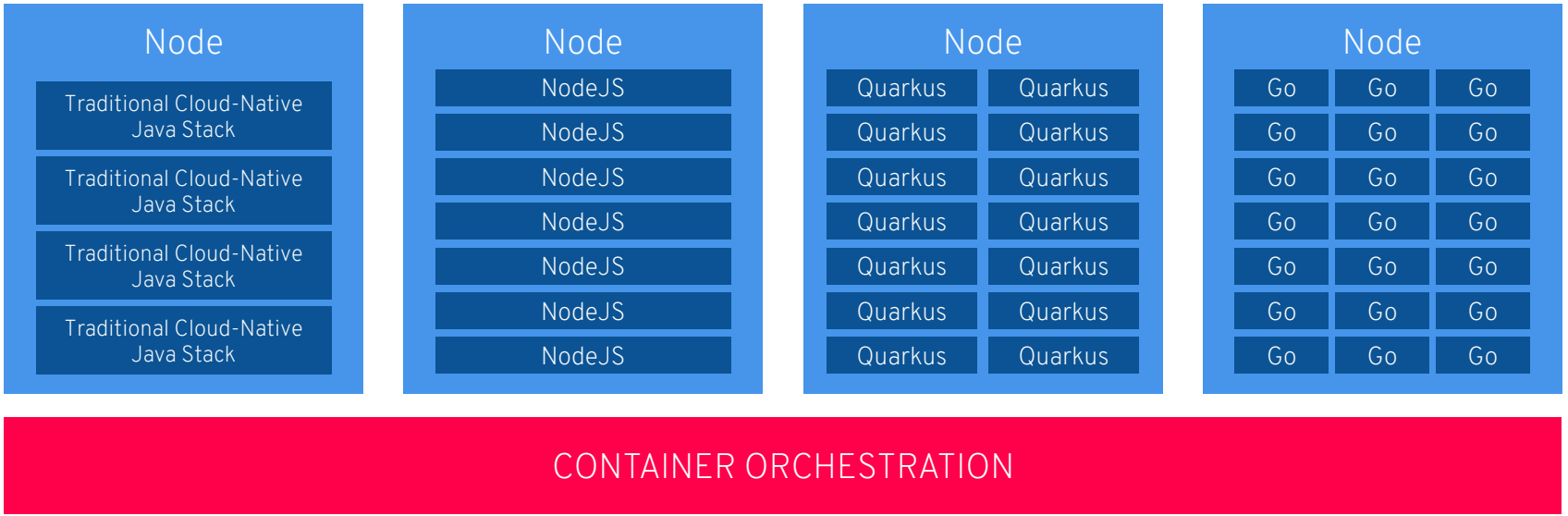
Benefit No. 2: Supersonic Subatomic Java





Time to first response

The New Truth about Java + Containers



Conclusions: rethinking the problem

- Our problems are not the same as they were 10 years ago
  - We can't expect the same solutions to make sense!
  - Containers are small and primarily immutable
- To adapt, we must truly understand conditions and make *different* trade-offs
  - Eclipse MicroProfile offers a standard way to trim down services
  - But there's still a lot more work to do across the entire stack
- Enable millions of Java developers to become truly cloud native