
QUARKUS
QUARKUS WORLD TOUR 2024
FOR
DEVELOPERS
ERIC DEANDREA
SR. PRINCIPAL DEVELOPER ADVOCATE
RED HAT
@edeandrea

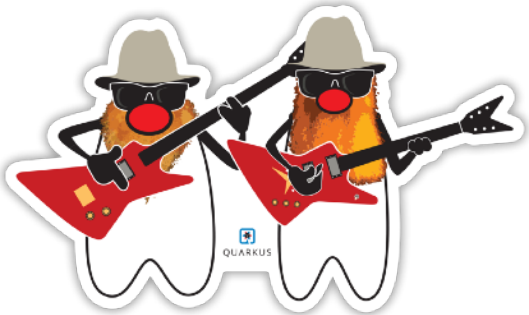
Who's on Stage Today?

Eric Deandrea

- 🤘 25+ years software development experience
- Java Champion
- 🤘 Contributor to Open Source projects
  - Quarkus
  - Spring Boot, Spring Framework, Spring Security
  - LangChain4j (& Quarkus LangChain4j)
  - Wiremock
  - Microcks
- Boston Java Users ACM Chapter Board Member
- 🤘 Published author

@edeandrea

- How Quarkus enables modern Java development & the Kubernetes-native experience
- Introduce familiar Spring concepts, constructs, & conventions and how they map to Quarkus
- Emphasis on testing patterns & practices



https://red.ht/quarkus-spring-devs

@edeandrea

Java, The Enterprise Workhorse

@edeandrea

Java, The Enterprise Workhorse

@edeandrea

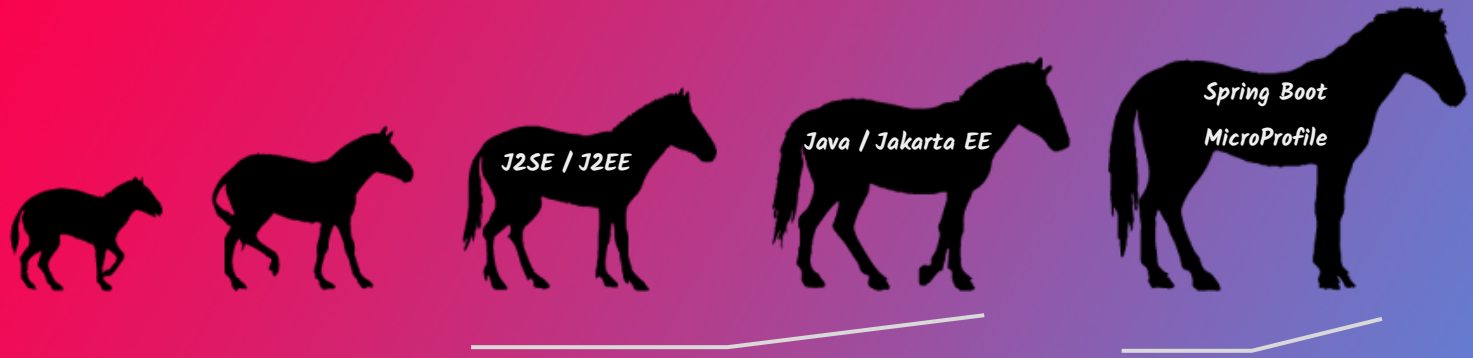
Java, The Enterprise Workhorse

Monolith

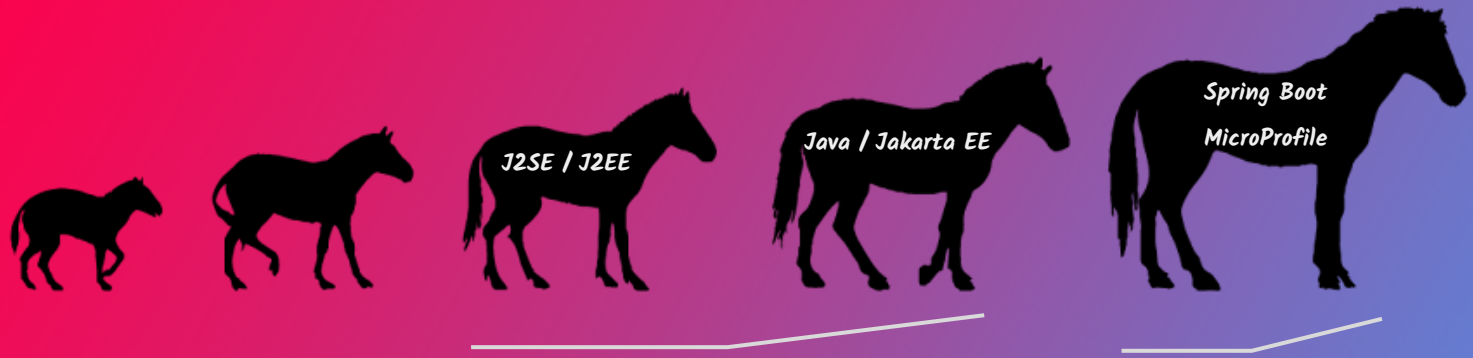
Java, The Enterprise Workhorse

Monolith

Java, The Enterprise Workhorse



Java, The Enterprise Workhorse



Java, The Enterprise Workhorse

Java, The Enterprise Workhorse

Monolith

Microservices
Cloud-Native

Java, The Enterprise Workhorse

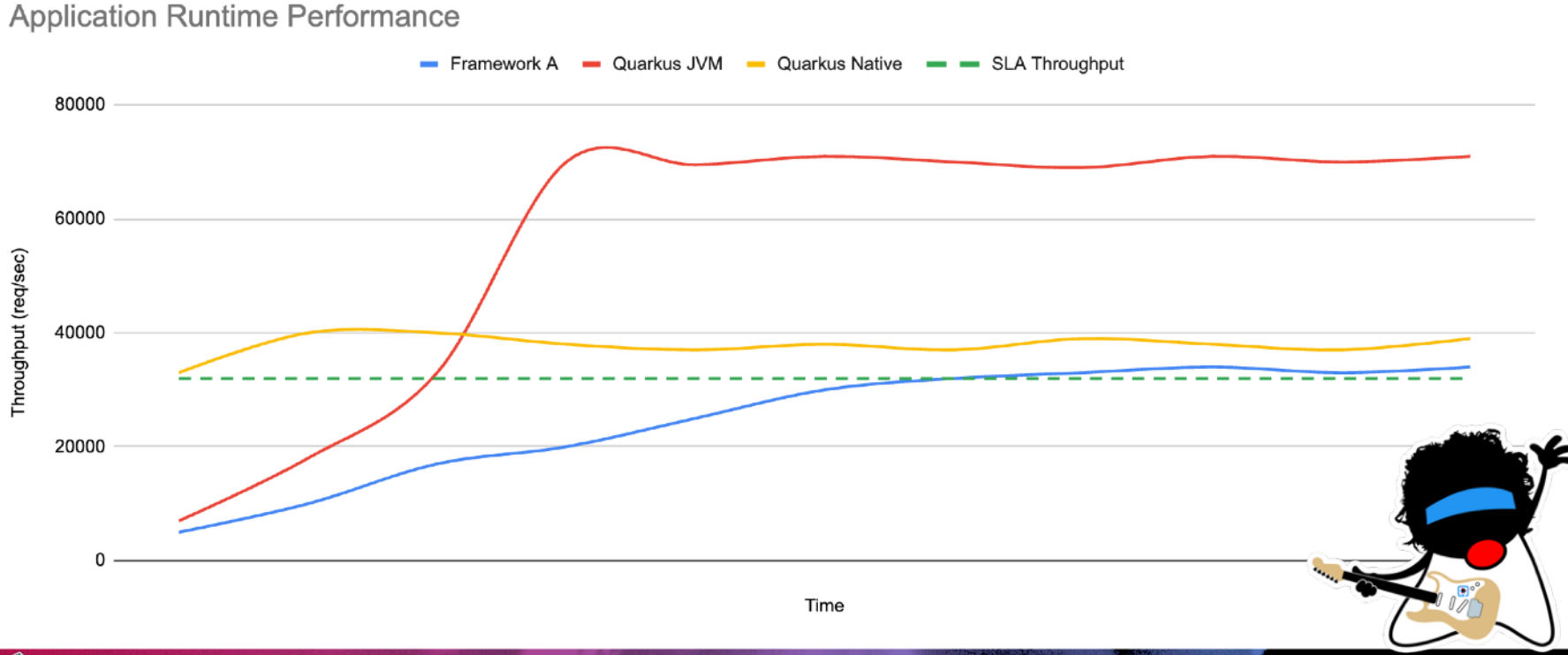
Monolith

Microservices
Cloud-Native

Microservices
Cloud-Native
Serverless
Event-Driven

@edeandrea

The Warmup Issue with Java



How Does a Framework Start?

Build Time

Runtime

Packaging (maven, etc)

Load config file from file system
Parse it

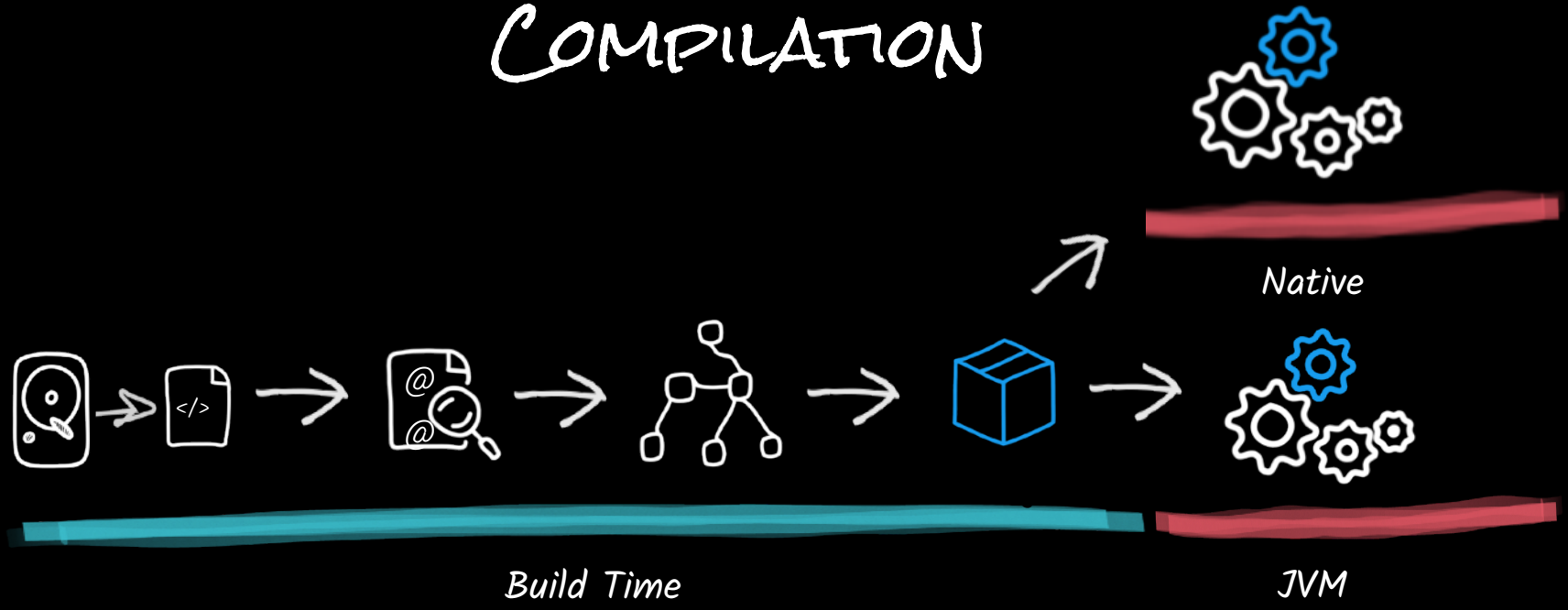
Classpath scanning to find annotated classes
Attempt to load class to enable/disable features

Build its model of the world.

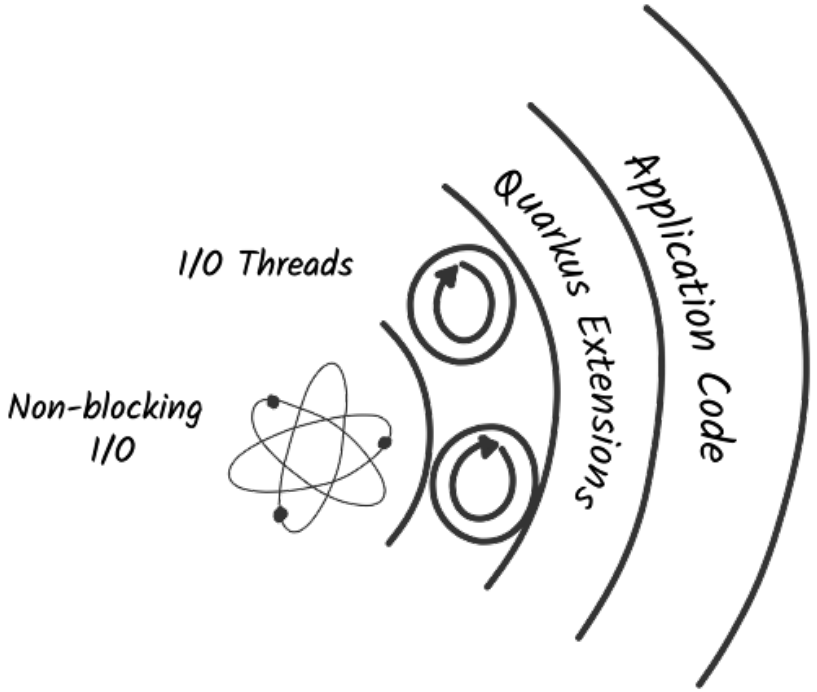
Start the management (thread, pool...)

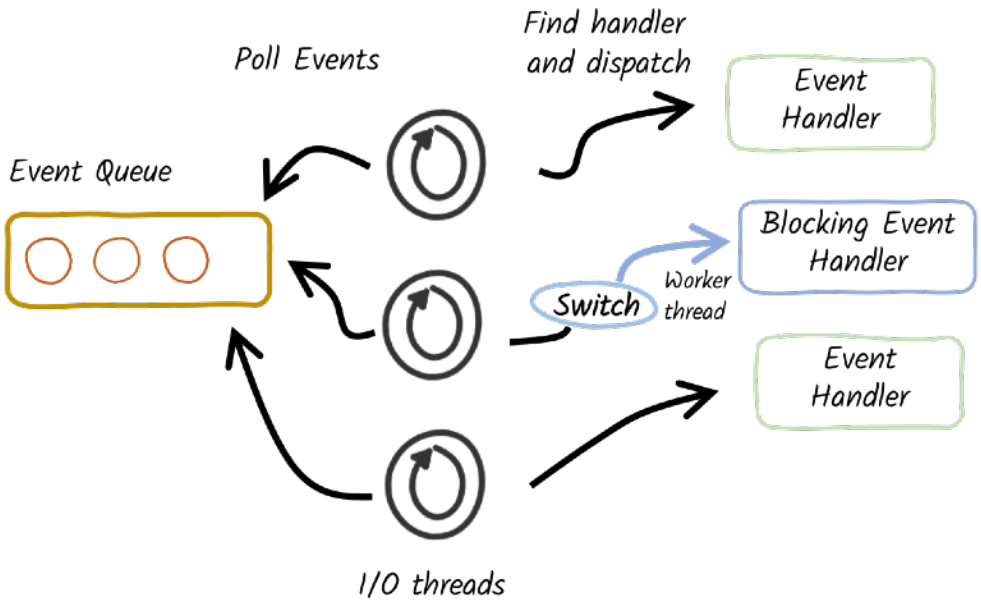
The Quarkus Way

The Quarkus Way enables Native Compilation



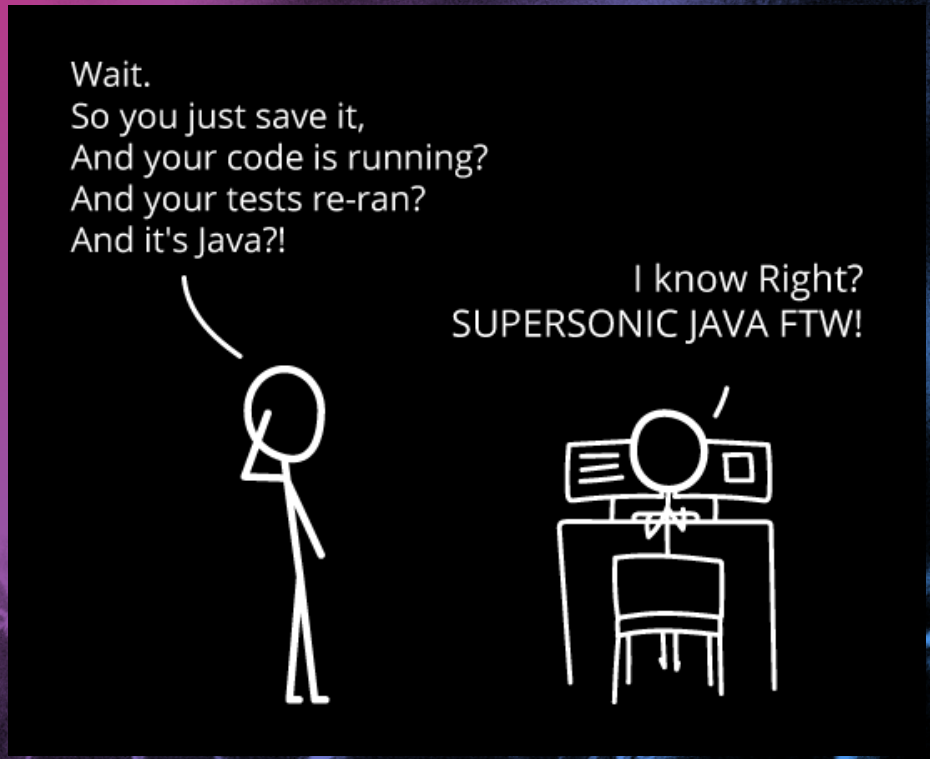
Unification of Imperative and Reactive



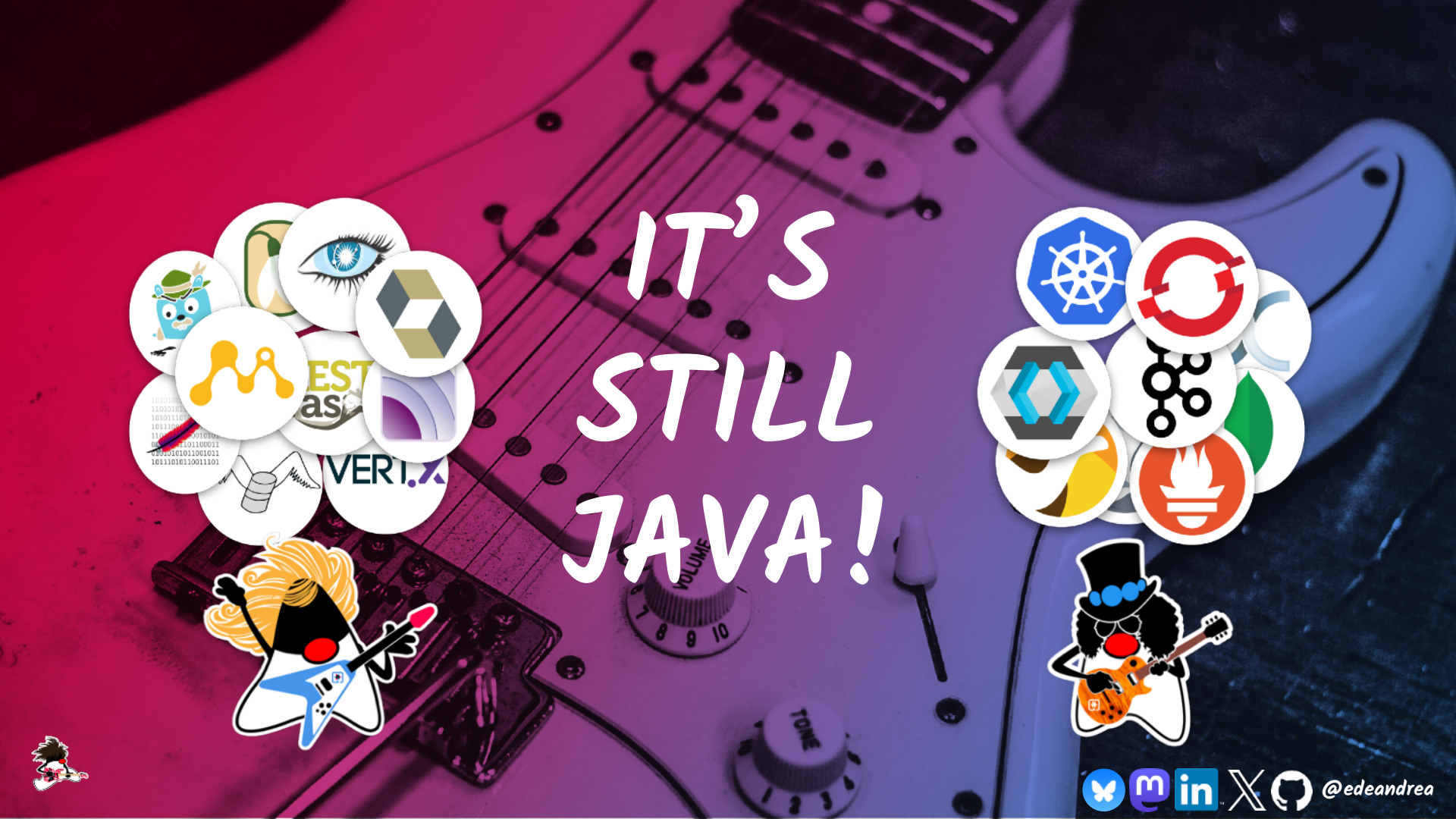


Enhancing Developer Joy

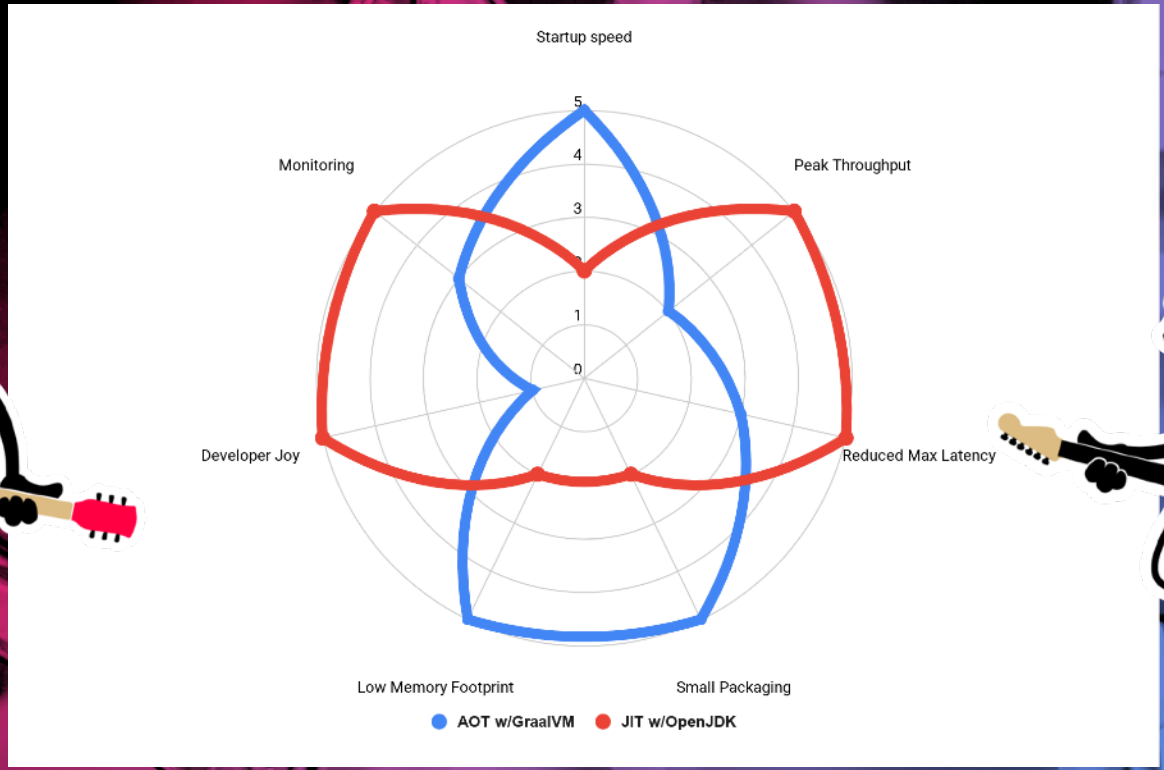
- 🤘 Zero-config Live coding
- 🤘 Auto-provision services
- 🤘 Continuous testing
- 🤘 Dev UI
- 🤘 CLI

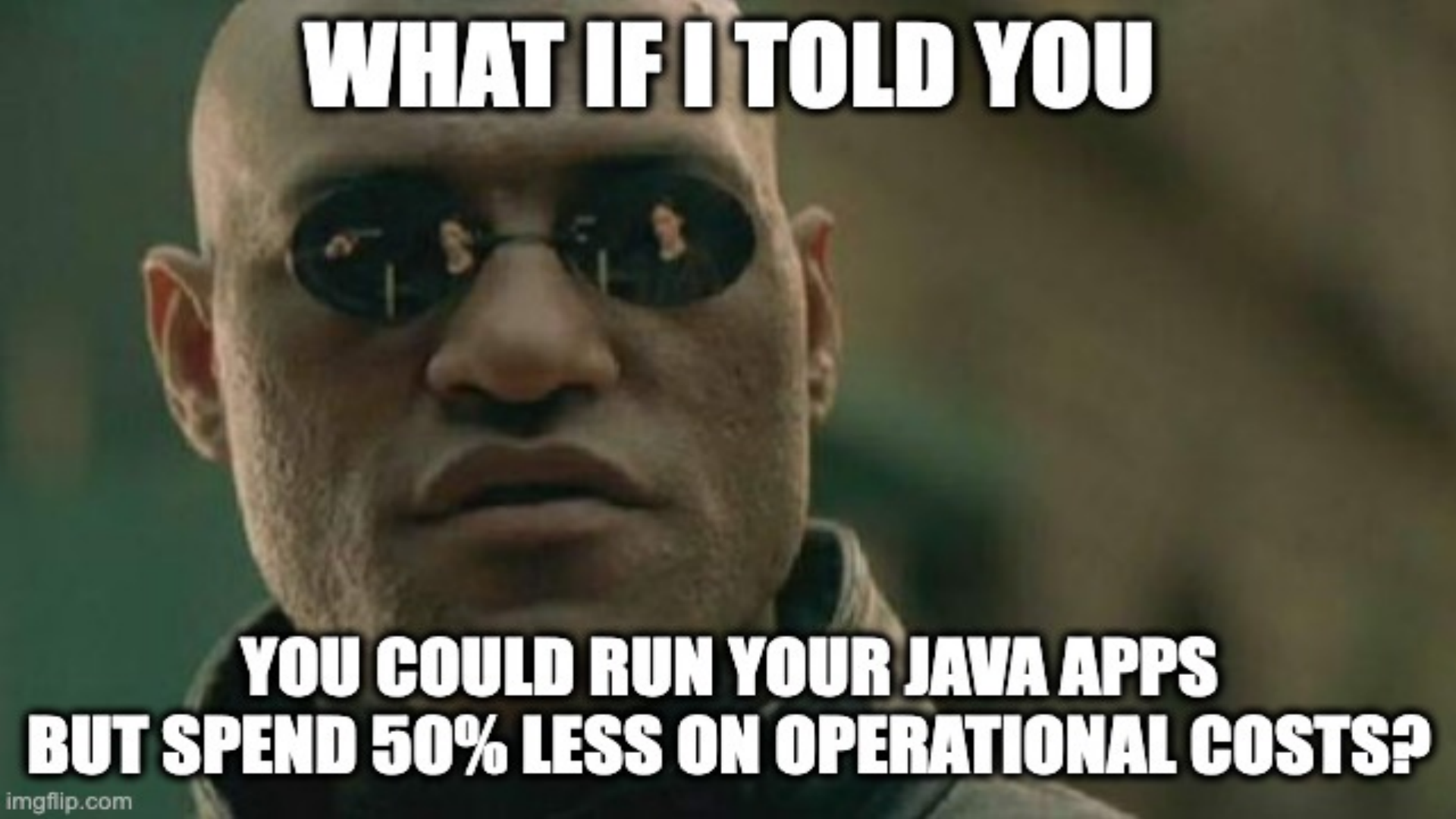


@edeandrea


IT'S
STILL
JAVA!
VERT.X
VOLUME
TONE
@edeandrea

AOT/Native vs JVM Mode



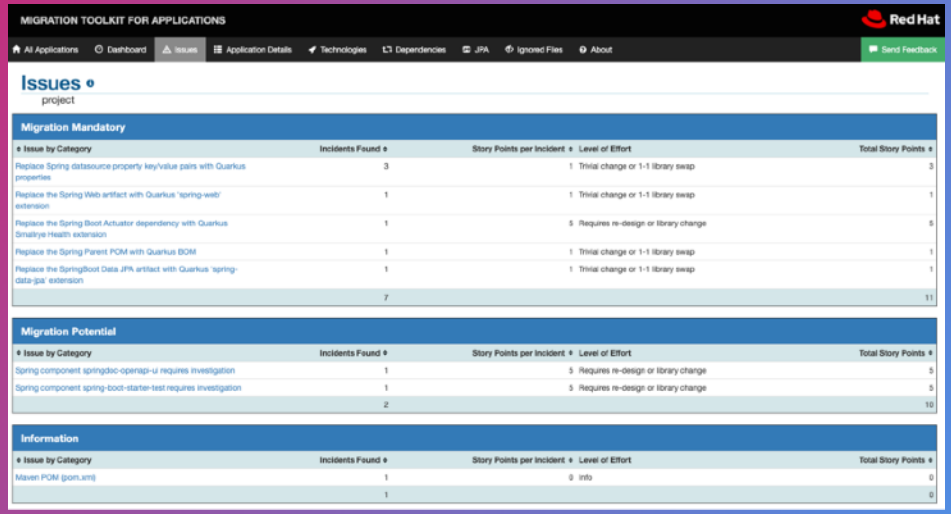

WHAT IF I TOLD YOU
YOU COULD RUN YOUR JAVA APPS
BUT SPEND 50% LESS ON OPERATIONAL COSTS?
imgflip.com

Jam Time!

@edeandrea

Migration Toolkit For Applications

"Simplifies the Migration of Spring Apps to Quarkus"



Proprietary Java EE runtimes	Java EE & JDK upgrades	JBoss EAP migrations
Spring Boot, Quarkus, Tomcat (JWS)	Cloud readiness, containerization	Pluggable: add your own rules
OracleJDK to OpenJDK Migrations		

Command Line Interface	Web Console	IDE Plugins	Maven Plugin

- Automate Application Analysis
- Estimate Level of Effort
- Accelerate Code Transformation & Migration
- Includes Rules for DI, Metrics, Security, Web, Shell, & More

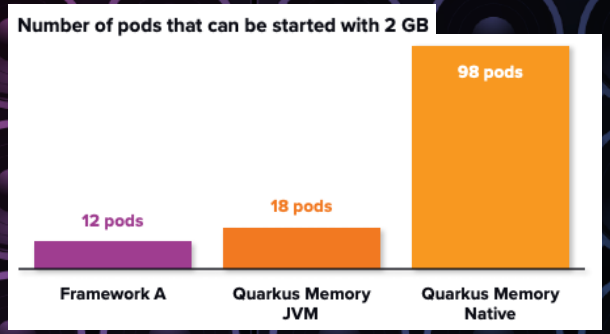
https://developers.redhat.com/products/mta

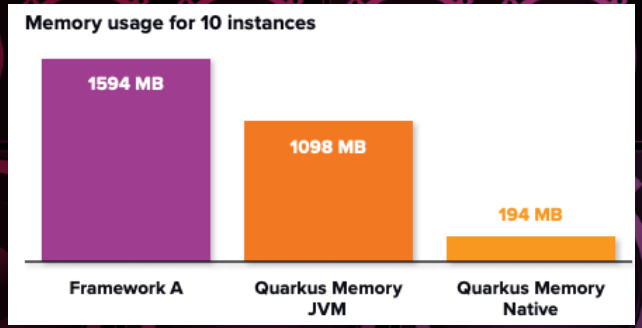
@edeandrea

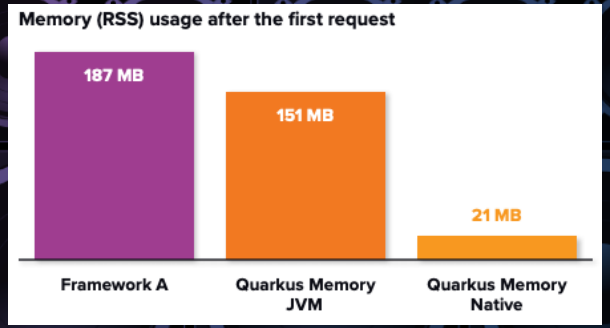
Don't Take My Word For It!

Best database-access responses per second, single query, Dell R440 Xeon Gold + 10 GbE (17 tests)

Rnk	Framework	Best performance (higher is better)	Errors	Cls
1	quarkus resteasy reactive + hibernate	318,897 — 100.0% (50.7)	0	Ful
2	quarkus	307,488 — 96.4% (48.9)	0	Ful
3	helidon 4	268,833 — 84.3% (42.8)	0	Mcr
4	helidon	247,545 — 77.6% (39.4)	0	Mcr
5	micronaut jdbc	221,741 — 69.5% (35.3)	0	Mcr
6	micronaut-graalvm	205,664 — 64.5% (32.7)	0	Mcr
7	micronaut	149,830 — 47.0% (23.8)	0	Mcr
8	spring	147,907 — 46.4% (23.5)	0	Ful
9	micronaut data mongodb	129,777 — 40.7% (20.6)	0	Mcr
10	micronaut jdbc graalvm	114,035 — 35.8% (18.1)	0	Mcr
11	spring-mongo	104,656 — 32.8% (16.7)	0	Ful
12	spring-jpa	104,318 — 32.7% (16.6)	0	Ful
13	micronaut vertx pg client graalvm	78,831 — 24.7% (12.5)	0	Mcr
14	micronaut data jdbc graalvm	77,663 — 24.4% (12.4)	0	Mcr
15	micronaut data mongodb graalvm	62,632 — 19.6% (10.0)	0	Mcr
16	micronaut r2dbc	62,495 — 19.6% (9.9)	0	Mcr
17	micronaut r2dbc graalvm	27,470 — 8.6% (4.4)	0	Mcr







https://www.techempower.com/benchmarks/#section=data-r22&test=db&f=zijunz-zik0zj-zik0zj-zik0zj-zik0zj-zik0zj-zik0zj-v2qiv3-xamxa7-zik0zj-zik0zj-zik0zj-zik0zj-zik0zj-lekf&hw=ph&l=zik0vz-cn3
https://www.redhat.com/en/blog/key-findings-idc-red-hat-quarkus-lab-validation
https://www.reddit.com/r/java/comments/o0ewar/do_quarkus_performance_benefits_scale

@edeandrea

Don't Take My Word For It!

Some real numbers (8 cores, 14GB RAM, GraalVM CE 21.0.2)

JVM			
	Quarkus	Spring Boot	Ratio (Quarkus / SB)
Framework version	3.15.1	3.3.4	
Build time (s)	9.02	5.29	170.57%
Av. RSS after startup (MB)	199.6	321.07	62.17%
Av. time to 1st req (ms)	3,195.33	6,109	52.31%
Av. RSS after 1st req (MB)	251.03	320.79	78.25%
Av. throughput (req/sec)	33,369.1	8,190.43	407.42%
Max throughput density (req/sec/MB)	60.55	12.67	477.97%

Native			
	Quarkus	Spring Boot	Ratio (Quarkus / SB)
Framework version	3.15.1	3.3.4	
Build time (s)	127.27	203.1	62.66%
Av. RSS after startup (MB)	72.56	192	37.27%
Av. time to 1st req (ms)	106	459.67	23.06%
Av. RSS after 1st req (MB)	79.26	194.71	40.71%
Av. throughput (req/sec)	19,128.37	6,742.24	283.71%
Max throughput density (req/sec/MB)	74.15	17.68	419.47%
Build RSS (GB)	6.28	7.79	80.62%
Binary Size (MB)	89.97	154.66	58.17%

Don't Take My Word For It!

Some real numbers (8 cores, 14GB RAM, GraalVM CE 21.0.2)

JVM			
	Quarkus	Spring Boot	Ratio (Quarkus / SB)
Framework version	3.15.1	3.3.4	
Build time (s)	9.02	5.29	170.57%
Av. RSS after startup (MB)	199.6	321.07	62.17%
Av. time to 1st req (ms)	3,195.33	6,109	52.31%
Av. RSS after 1st req (MB)	251.03	320.79	78.25%
Av. throughput (req/sec)	33,369.1	8,190.43	407.42%
Max throughput density (req/sec/MB)	60.55	12.67	477.97%

Native			
	Quarkus	Spring Boot	Ratio (Quarkus / SB)
Framework version	3.15.1	3.3.4	
Build time (s)	127.27	203.1	62.66%
Av. RSS after startup (MB)	72.56	192	37.27%
Av. time to 1st req (ms)	106	459.67	23.06%
Av. RSS after 1st req (MB)	79.26	194.71	40.71%
Av. throughput (req/sec)	19,128.37	6,742.24	283.71%
Max throughput density (req/sec/MB)	74.15	17.68	419.47%
Build RSS (GB)	6.28	7.79	80.62%
Binary Size (MB)	89.97	154.66	58.17%

Don't Take My Word For It!

Some real numbers (8 cores, 14GB RAM, GraalVM CE 21.0.2)

JVM			
	Quarkus	Spring Boot	Ratio (Quarkus / SB)
Framework version	3.15.1	3.3.4	
Build time (s)	9.02	5.29	170.57%
Av. RSS after startup (MB)	199.6	321.07	62.17%
Av. time to 1st req (ms)	3,195.33	6,109	52.31%
Av. RSS after 1st req (MB)	251.03	320.79	78.25%
Av. throughput (req/sec)	33,369.1	8,190.43	407.42%
Max throughput density (req/sec/MB)	60.55	12.67	477.97%

Native			
	Quarkus	Spring Boot	Ratio (Quarkus / SB)
Framework version	3.15.1	3.3.4	
Build time (s)	127.27	203.1	62.66%
Av. RSS after startup (MB)	72.56	192	37.27%
Av. time to 1st req (ms)	106	459.67	23.06%
Av. RSS after 1st req (MB)	79.26	194.71	40.71%
Av. throughput (req/sec)	19,128.37	6,742.24	283.71%
Max throughput density (req/sec/MB)	74.15	17.68	419.47%
Build RSS (GB)	6.28	7.79	80.62%
Binary Size (MB)	89.97	154.66	58.17%

Don't Take My Word For It!

Some real numbers (8 cores, 14GB RAM, GraalVM CE 21.0.2)

JVM			
	Quarkus	Spring Boot	Ratio (Quarkus / SB)
Framework version	3.15.1	3.3.4	
Build time (s)	9.02	5.29	170.57%
Av. RSS after startup (MB)	199.6	321.07	62.17%
Av. time to 1st req (ms)	3,195.33	6,109	52.31%
Av. RSS after 1st req (MB)	251.03	320.79	78.25%
Av. throughput (req/sec)	33,369.1	8,190.43	407.42%
Max throughput density (req/sec/MB)	60.55	12.67	477.97%

Native			
	Quarkus	Spring Boot	Ratio (Quarkus / SB)
Framework version	3.15.1	3.3.4	
Build time (s)	127.27	203.1	62.66%
Av. RSS after startup (MB)	72.56	192	37.27%
Av. time to 1st req (ms)	106	459.67	23.06%
Av. RSS after 1st req (MB)	79.26	194.71	40.71%
Av. throughput (req/sec)	19,128.37	6,742.24	283.71%
Max throughput density (req/sec/MB)	74.15	17.68	419.47%
Build RSS (GB)	6.28	7.79	80.62%
Binary Size (MB)	89.97	154.66	58.17%

Don't Take My Word For It!

Some real numbers (8 cores, 14GB RAM, GraalVM CE 21.0.2)

JVM			
	Quarkus	Spring Boot	Ratio (Quarkus / SB)
Framework version	3.15.1	3.3.4	
Build time (s)	9.02	5.29	170.57%
Av. RSS after startup (MB)	199.6	321.07	62.17%
Av. time to 1st req (ms)	3,195.33	6,109	52.31%
Av. RSS after 1st req (MB)	251.03	320.79	78.25%
Av. throughput (req/sec)	33,369.1	8,190.43	407.42%
Max throughput density (req/sec/MB)	60.55	12.67	477.97%

Native			
	Quarkus	Spring Boot	Ratio (Quarkus / SB)
Framework version	3.15.1	3.3.4	
Build time (s)	127.27	203.1	62.66%
Av. RSS after startup (MB)	72.56	192	37.27%
Av. time to 1st req (ms)	106	459.67	23.06%
Av. RSS after 1st req (MB)	79.26	194.71	40.71%
Av. throughput (req/sec)	19,128.37	6,742.24	283.71%
Max throughput density (req/sec/MB)	74.15	17.68	419.47%
Build RSS (GB)	6.28	7.79	80.62%
Binary Size (MB)	89.97	154.66	58.17%

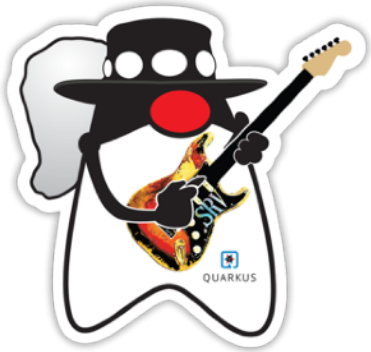
Don't Take My Word For It!

Some real numbers (8 cores, 14GB RAM, GraalVM CE 21.0.2)

JVM			
	Quarkus	Spring Boot	Ratio (Quarkus / SB)
Framework version	3.15.1	3.3.4	
Build time (s)	9.02	5.29	170.57%
Av. RSS after startup (MB)	199.6	321.07	62.17%
Av. time to 1st req (ms)	3,195.33	6,109	52.31%
Av. RSS after 1st req (MB)	251.03	320.79	78.25%
Av. throughput (req/sec)	33,369.1	8,190.43	407.42%
Max throughput density (req/sec/MB)	60.55	12.67	477.97%

Native			
	Quarkus	Spring Boot	Ratio (Quarkus / SB)
Framework version	3.15.1	3.3.4	
Build time (s)	127.27	203.1	62.66%
Av. RSS after startup (MB)	72.56	192	37.27%
Av. time to 1st req (ms)	106	459.67	23.06%
Av. RSS after 1st req (MB)	79.26	194.71	40.71%
Av. throughput (req/sec)	19,128.37	6,742.24	283.71%
Max throughput density (req/sec/MB)	74.15	17.68	419.47%
Build RSS (GB)	6.28	7.79	80.62%
Binary Size (MB)	89.97	154.66	58.17%

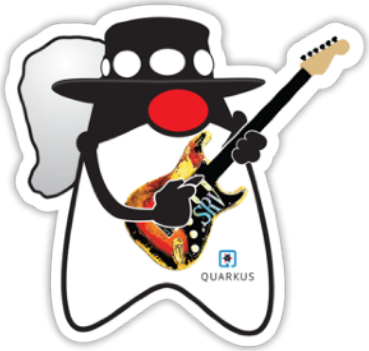
A Real Example



A Real Example

🤘 **Spring Boot 3 application contents:**

- 38,741 classes
- 83,538 fields
- 276,952 methods
- **Reflection:**
  - 10,976 classes
  - 1,025 fields
  - 12,885 methods



A Real Example

🤘 **Spring Boot 3 application contents:**

- 38,741 classes
- 83,538 fields
- 276,952 methods
- **Reflection:**
  - 10,976 classes
  - 1,025 fields
  - 12,885 methods

🤘 **Quarkus 3 application contents:**

- 23,707 classes
- 48,142 fields
- 189,563 methods
- **Reflection:**
  - 6,656 classes
  - 201 fields
  - 4,800 methods

A Real Example

🤘 **Spring Boot 3 application contents:**

- 38,741 classes
- 83,538 fields
- 276,952 methods
- **Reflection:**
  - 10,976 classes
  - 1,025 fields
  - 12,885 methods

🤘 **The Quarkus application has:**

- 15,035 (39%) less classes
- 35,396 (42%) less fields
- 87,389 (32%) less methods

🤘 **Quarkus 3 application contents:**

- 23,707 classes
- 48,142 fields
- 189,563 methods
- **Reflection:**
  - 6,656 classes
  - 201 fields
  - 4,800 methods

- 4,320 (39%) less classes using reflection
- 824 (80%) less fields using reflection
- 8,085 (63%) less methods using reflection

Don't Take Our Word For It!

Accelerating time-to-market while optimizing costs

Saved operational costs thanks to a smaller footprint

The Red Hat build of Quarkus allows developers to optimize container-native applications for peak performance. Its small computing resource footprint ensures low memory and CPU consumption, reducing overall costs.

Reduced cluster usage by 40%, making room for new workloads

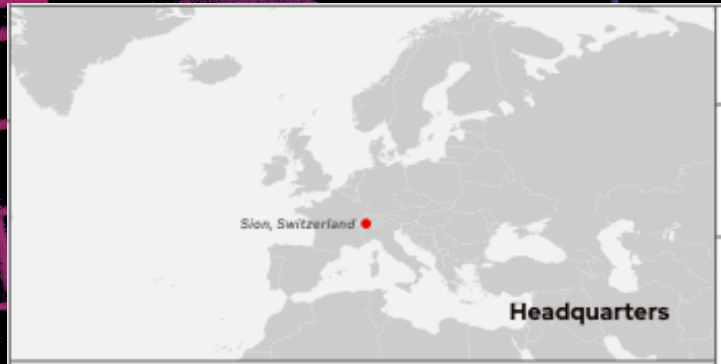
With those applications having a significantly smaller footprint using the framework of Red Hat's build of Quarkus, operations can now run additional meaningful workloads on their existing Red Hat OpenShift cluster. Spring Boot would require a cluster up to 40% larger than Red Hat's build of Quarkus.

Increased developer productivity threefold

The fast startup times of Red Hat's build of Quarkus' have helped Etat du Valais significantly reduce development time. "With Red Hat Quarkus, we've cut development time by three compared to using our old J2EE server eight years ago," said Favez. "Our developers can write their code faster, and they can test it faster too."

Ensured an easy transition from Spring and J2EE for Java developers

Switching from the Spring Boot and J2EE frameworks to Red Hat's build of Quarkus was really easy for Etat du Valais' developers. "Red Hat Quarkus is still Java," said Favez. "It uses a language that my team was already familiar with. So, it was no big deal to move from J2EE or Spring Boot to Red Hat Quarkus. It was really easy. There is nothing new for them."



"We have cut development time by 3 with the Red Hat build of Quarkus."

Steve Favez
Web Architect and Head of Internet Capability Center, Etat du Valais

Government

40 business units;
3,500 employees

Benefits

- Saved cost thanks to low memory and CPU consumption
- Reduced cluster usage by 40%, making room for new workloads
- Increased developer productivity threefold
- Ensured an easy transition from Spring and J2EE for Java developers

https://www.redhat.com/en/resources/etat-du-valais-customer-case-study

@edeandrea

Don't Take Our Word For It!

asiakastieto	"We went from 1 min startup times to 400 ms."
vodafone	"We became increasingly worried about resource consumption that Spring Boot was having while being deployed on the Kubernetes cluster... It became increasingly cumbersome to find ways to circumvent the methodology we were using just to squeeze every little bit of performance out of Spring Boot" - Christos Sotiriou DXL Backend Chapter Lead, Vodafone Greece
Lufthansa	"We could run 3x denser deployments without sacrificing availability and response times of service." "Quarkus is close to what our developers are already doing with Spring and it's familiar to them. This is a big benefit"
wipro	"Before we introduced Quarkus, many of our customers had started to look at alternative stacks like Go and node.js to improve performance and efficiency. These customers were weary of selecting a new language and having to hire new developers or retrain their existing Java developers." - Arijit Mazumdar "There was a low learning curve with Quarkus. It took one of our developers one week to get up to speed on Quarkus and another week to migrate a Spring application to Quarkus." - Arijit Mazumdar "Quarkus and the Spring API compatibility reduced the migration time and complexity which is critically important for our customers." - Arijit Mazumdar

https://quarkus.io/blog/tag/user-story

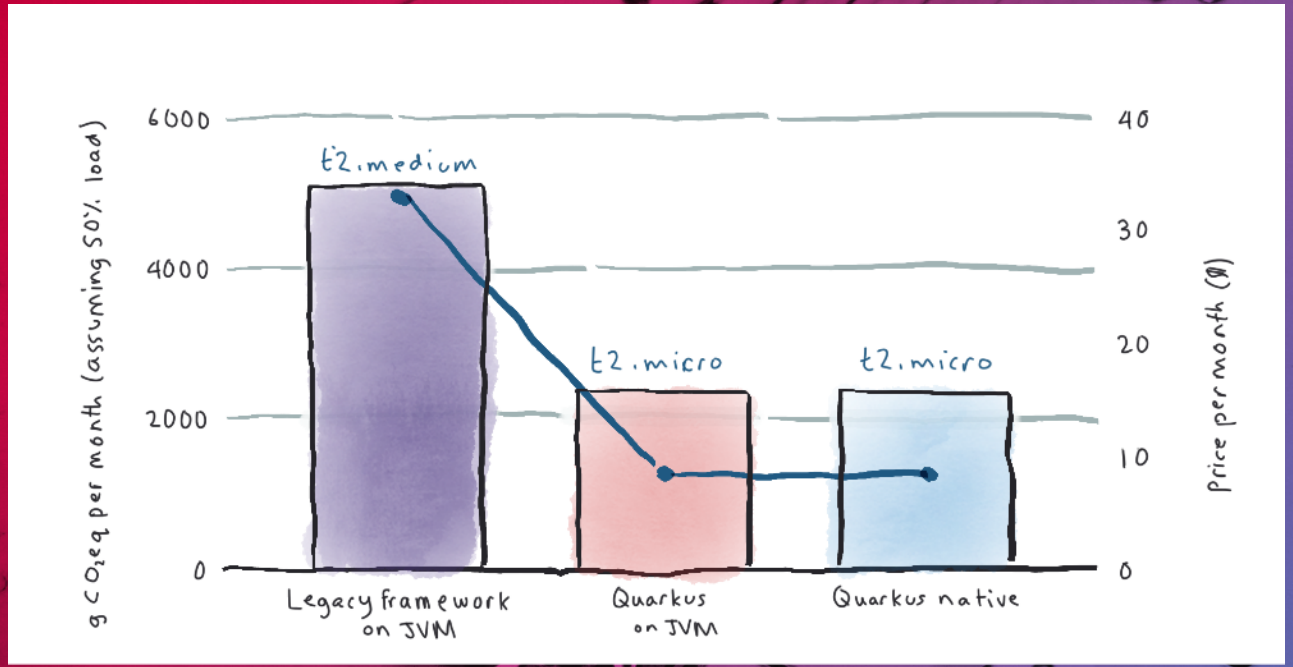
Don't Take Our Word For It!

ENNOVATIVE SOLUTIONS	"using Spring with AWS Lambda would have been prohibitive because the startup time of Spring in AWS Lambda is too big from my research" - Dennis Baerten "As costs increase, this is when the benefit of using Quarkus will be experienced due to its more efficient use of cloud resources and fast startup time compared to plain Java and Spring Boot" - Dennis Baerten "It took me about 3 days to get familiar with the Quarkus stack" - Dennis Berten, Spring Developer
Payair	"Some of Payair's developers had mainly Spring experience, we were concerned that it would be difficult for them to "switch sides". It turned out that our fear of the unknown was completely unfounded. Quarkus leverages some good old Jakarta EE standards that all Java developers are familiar with. We did not have to learn a bunch of new APIs." - Hubert Lewandowski "As a long term Spring developer I realized that Spring is slowly becoming the very thing it swore to destroy. The initial premises of Spring (which basically can be summed up as a lightweight alternative to Jakarta EE) are way past the expiry date now. Spring is the undisputed heavyweight champion that can handle everything you imagine but is not your best option for fast and light services. And that applies to Spring Boot as well. - Hubert Lewandowski
talkdesk	"When you adopt Quarkus, you will be productive from day one since you don't really need to learn new technologies." -TalkDesk
Logic Monitor	"After deploying, we found that Quarkus used about 15% of the CPU, 12% of the memory compared to Spring Boot. So far, we are sticking with Quarkus!" - Sam Dacanay, Lead Software Engineer

https://quarkus.io/blog/tag/user-story

@edeandrea

The Cost / Carbon Impact



Setup

- 🤘 AWS (us-east-1)
- 🤘 SLA > 99%
- 🤘 800 req/sec over 20 days
- 🤘 50% load

https://www.redhat.com/en/resources/greener-java-applications-detail

@edeandrea


ARE YOU
What is he doing?
A BELIEVER NOW?
He's beginning to believe.
imgflip.com

@edeandrea

But I Already Know Spring...

But I already know Spring...

🤘 Can I do reactive/imperative/blocking?

@edeandrea

But I Already Know Spring...

- 🤘 Can I do reactive/imperative/blocking?
- 🤘 What if there isn't an extension for my library?

@edeandrea

But I Already Know Spring...

- 🤘 Can I do reactive/imperative/blocking?
- 🤘 What if there isn't an extension for my library?
- 🤘 Can I reuse my existing Spring code?

@edeandrea

But I Already Know Spring...

- 🤘 Can I do reactive/imperative/blocking?
- 🤘 What if there isn't an extension for my library?
- 🤘 Can I reuse my existing Spring code?
- 🤘 How do I migrate?

@edeandrea

But I Already Know Spring...

- 🤘 Can I do reactive/imperative/blocking?
- 🤘 What if there isn't an extension for my library?
- 🤘 Can I reuse my existing Spring code?
- 🤘 How do I migrate?
- 🤘 How stable is it?

@edeandrea

But I Already Know Spring...

- Can I do reactive/imperative/blocking?
- What if there isn't an extension for my library?
- Can I reuse my existing Spring code?
- How do I migrate?
- How stable is it?
- GraalVM compilation takes time & memory...

@edeandrea

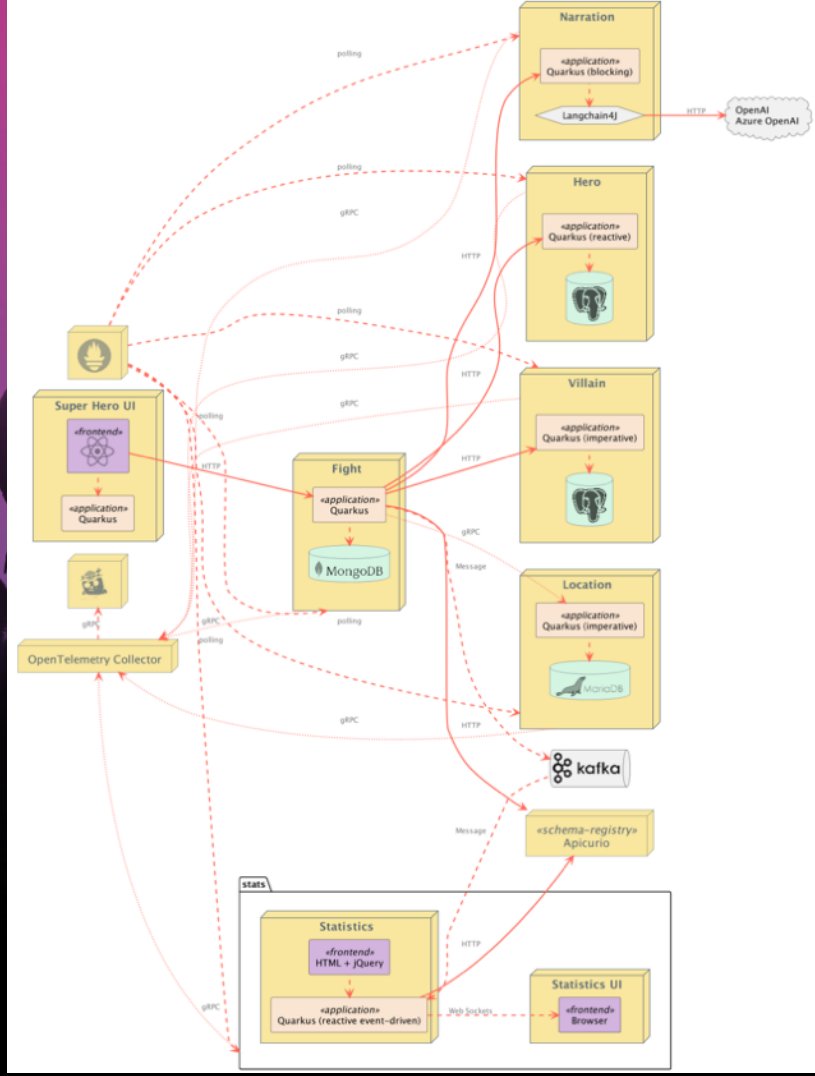
But I Already Know Spring...

- 🤘 Can I do reactive/imperative/blocking?
- 🤘 What if there isn't an extension for my library?
- 🤘 Can I reuse my existing Spring code?
- 🤘 How do I migrate?
- 🤘 How stable is it?
- 🤘 GraalVM compilation takes time & memory...
- 🤘 What if I have my own "meta" framework on "top" of Spring?

@edeandrea

Quarkus Superheroes

https://github.com/quarkusio/quarkus-super-heroes
https://quarkus.io/quarkus-workshops/super-heroes




Thank You!
@edeandrea