JAVA

READY FOR MICROSERVICES?

"OPEN, EVOLVING, NIMBLE & SCALABLE"?

ANDREAS TELL

CADEC 2018.03.08 | CALLISTAENTERPRISE.SE

CALLISTA

— ENTERPRISE —

2017-08-31 : JAVA EE 8

- +4 years since last release…
  - Containers & Microservice Architecture
- New/Updated APIs:
  - Focus on Web, REST and JSON
- https://github.com/javaee/

SERVLET API 4.0
- HTTP/2
- HTTP TRAILER (RFC 7230, RFC 7540)

JSON-P 1.1

JSON-B 1.0
- "JAXB FOR JSON"

JAX-RS 1.1
- REACTIVE CLIENT API
- SERVER SENT EVENTS

JSF 2.3

SECURITY 1.0

BEAN VALIDATION 2.0

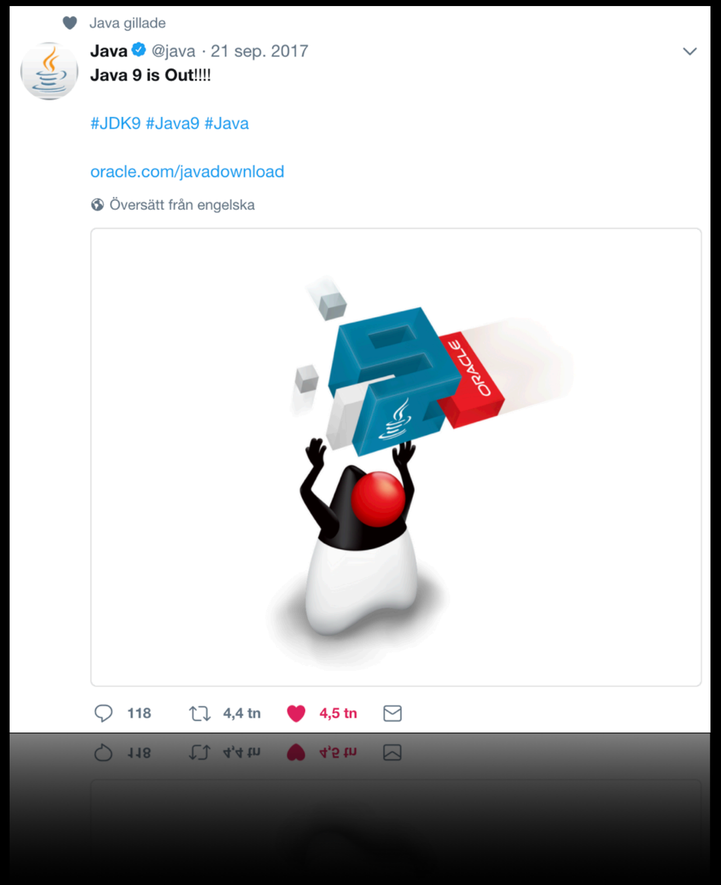
CDI 2.0

JPA 2.2

…

…

2017-09-21 : JAVA SE 9



102: Process API Updates
110: HTTP 2 Client
143: Improve Contended Locking
158: Unified JVM Logging
165: Compiler Control
193: Variable Handles
197: Segmented Code Cache
199: Smart Java Compilation, Phase Two
200: The Modular JDK
201: Modular Source Code
211: Elide Deprecation Warnings on Import Statements
212: Resolve Lint and Doclint Warnings
213: Milling Project Coin
214: Remove GC Combinations Deprecated in JDK 8
215: Tiered Attribution for javac
216: Process Import Statements Correctly
217: Annotations Pipeline 2.0
219: Datagram Transport Layer Security (DTLS)
220: Modular Run-Time Images
221: Simplified Doclet API
222: jshell: The Java Shell (Read-Eval-Print Loop)
223: New Version-String Scheme
224: HTML5 Javadoc
225: Javadoc Search
226: UTF-8 Property Files
227: Unicode 7.0
228: Add More Diagnostic Commands
229: Create PKCS12 Keystores by Default
231: Remove Launch-Time JRE Version Selection
232: Improve Secure Application Performance
233: Generate Run-Time Compiler Tests Automatically
235: Test Class-File Attributes Generated by javac
236: Parser API for Nashorn
237: Linux/AArch64 Port
238: Multi-Release JAR Files
240: Remove the JVM TI hprof Agent
241: Remove the jhat Tool
243: Java-Level JVM Compiler Interface
244: TLS Application-Layer Protocol Negotiation Extension
245: Validate JVM Command-Line Flag Arguments
246: Leverage CPU Instructions for GHASH and RSA
247: Compile for Older Platform Versions
248: Make G1 the Default Garbage Collector
249: OCSP Stapling for TLS
250: Store Interned Strings in CDS Archives
251: Multi-Resolution Images
252: Use CLDR Locale Data by Default
253: Prepare JavaFX UI Controls & CSS APIs for Modularization
254: Compact Strings
255: Merge Selected Xerces 2.11.0 Updates into JAXP
256: BeanInfo Annotations
257: Update JavaFX/Media to Newer Version of GStreamer
258: HarfBuzz Font-Layout Engine
259: Stack-Walking API
260: Encapsulate Most Internal APIs
261: Module System
262: TIFF Image I/O
263: HiDPI Graphics on Windows and Linux
264: Platform Logging API and Service
265: Marlin Graphics Renderer
266: More Concurrency Updates
267: Unicode 8.0
268: XML Catalogs
269: Convenience Factory Methods for Collections
270: Reserved Stack Areas for Critical Sections
271: Unified GC Logging
272: Platform-Specific Desktop Features
273: DRBG-Based SecureRandom Implementations
274: Enhanced Method Handles
275: Modular Java Application Packaging
276: Dynamic Linking of Language-Defined Object Models
277: Enhanced Deprecation
278: Additional Tests for Humongous Objects in G1
279: Improve Test-Failure Troubleshooting
280: Indify String Concatenation
281: HotSpot C++ Unit-Test Framework
282: jlink: The Java Linker
283: Enable GTK 3 on Linux
284: New HotSpot Build System
285: Spin-Wait Hints
287: SHA-3 Hash Algorithms
288: ~~Disable SHA-1 Certificates~~
289: Deprecate the Applet API 😎
290: ~~Filter Incoming Serialization Data~~
291: Deprecate the Concurrent Mark Sweep (CMS) Garbage Collector
292: Implement Selected ECMAScript 6 Features in Nashorn
294: Linux/s390x Port
295: Ahead-of-Time Compilation
297: Unified arm32/arm64 Port
298: Remove Demos and Samples
299: Reorganize Documentation

2017-10-01 : JAVA ONE

JAVA PLATFORM, STATE OF AFFAIRS:

Huge and vibrant ecosystem + 12M Developers worldwide

Consistently rated #1 or #2 as most popular in programming language for over a decade (Redmonk, Tiobe)

#1 runtime in the cloud (AWS, MS Azure, GCP)

Slow pace of innovation
3-4 years of release cadence

Closed ecosystem

The JVM is not optimized for short-lived ephemeral workloads;
Startup time, Memory footprint, JIT

2017-10-01 : JAVA ONE

All features in Oracle JDK will be open sourced (GPL)

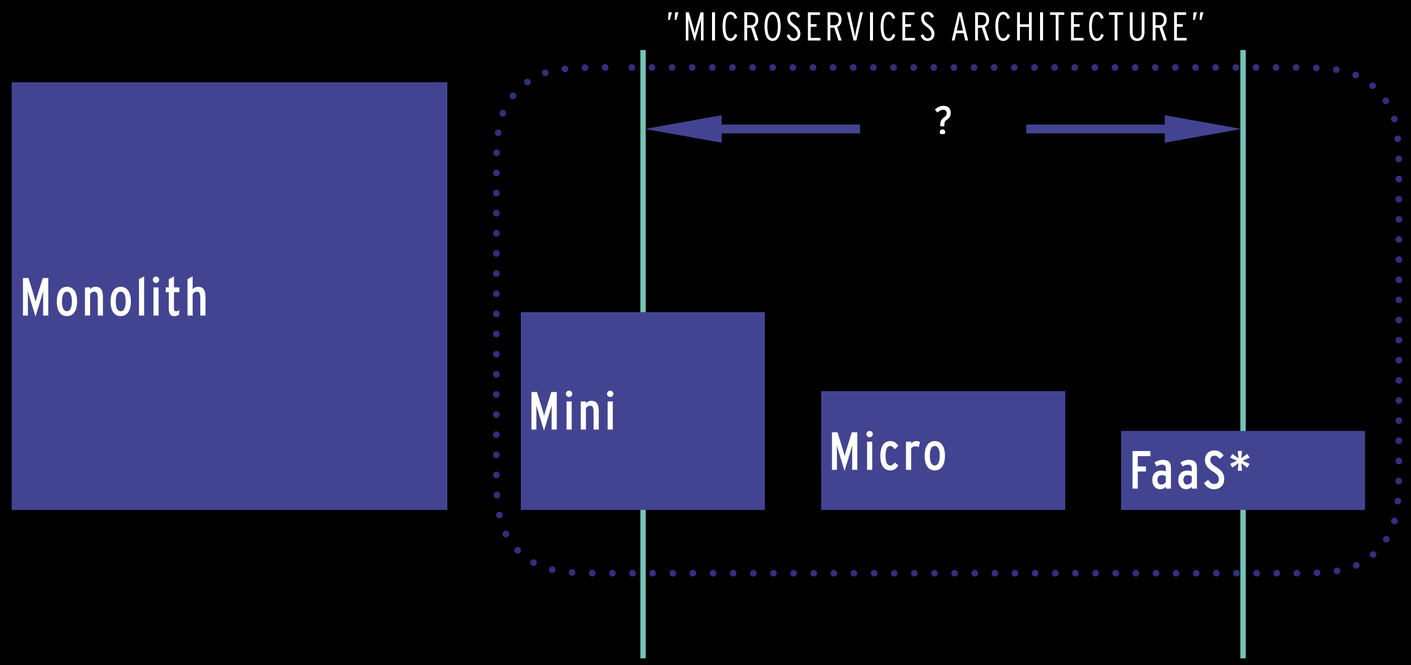
Java SE - six month release cadence 😎

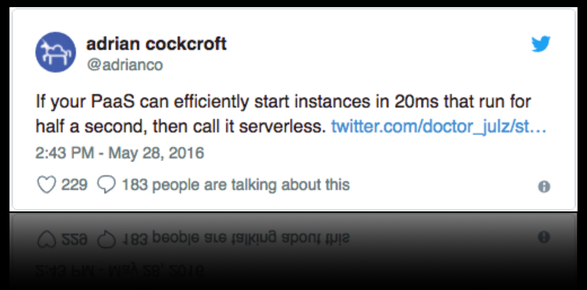
Jakarta EE moved to Eclipse Foundation, project "EE4J"

https://github.com/eclipse-ee4j

READY FOR MICROSERVICES?

- Libraries/Tools
- Competence
- Container aware
  - cgroups, name spaces
- Runtime Footprint
  - Memory
  - CPU
- Startup time





* FAAS = FUNCTION AS A SERVICE:
AWS LAMBDA
AZURE FUNCTIONS ET. AL.

HAS RUNTIME MEMORY FOOTPRINT IMPROVED WITH JAVA 9? LET'S FIND OUT!

Simple REST API with two endpoints returning JSON using;

1. Spring Boot 2 (M7) App with embedded thread per request server (Undertow)

2. Spring Boot 2 (M7) App with Reactive thread per core server (Netty)

3. JVM JDK HttpServer App

4. Vert.x App with Reactive thread per core server (Netty)

Environment:
Docker w Docker-Compose
Unaltered thread pools
mem_limit=320m
-Xmx32m -Xss256k
Apache Benchmark Test
5 * 2000 requests, 4 concurrent reqs

RUNTIME FOOTPRINT : MEMORY

SPRING TRADITIONAL BLOCKING - THREAD PER REQUEST - WEB SERVER (UNDERTOW)

```
ctop - 16:22:03 CET      5 containers

  NAME                CID            CPU        MEM                NET RX/TX     IO R/W         PIDS
● orajre8-spring      100656c7b4a6   14%        181M / 320M        8M / 10M      16K / 4K       58
● orajre8-spring-cds  bb9faa318b87   14%        209M / 320M        8M / 10M      164K / 4K      58
● orajre9-spring      bcd56bdafe61   14%        216M / 320M        8M / 10M      4K / 4K        56
● orajre9-spring-appc… f0605edb763f  15%        197M / 320M        8M / 10M      0B / 4K        56
● orajre9-spring-cds  af0534cd1968   15%        185M / 320M        8M / 10M      0B / 4K        56
```

SPRING REACTIVE NON-BLOCKING – THREAD PER CORE - WEB SERVER (NETTY)

```
ctop - 21:07:32 CET      2 containers

  NAME                CID            CPU        MEM                NET RX/TX     IO R/W         PIDS
● openjdk10ea-spring-… 9177de4705cd   0%         164M / 320M        5M / 6M       0B / 0B        19
● orajre9-spring-react 5ff69c1dba8c   0%         157M / 320M        5M / 6M       0B / 4K        19
```

RUNTIME FOOTPRINT : MEMORY CONT´D

JDK HTTPSERVER

ctop - 13:20:04 CET 5 containers

	NAME	CID	CPU		MEM		NET RX/TX	IO R/W	PIDS
●	openjdk10ea-jvm	c295dd36af1f		0%		53M / 320M	2M / 3M	0B / 0B	15
●	orajdk9-jvm-aot	4b816387d2e6		0%		59M / 320M	2M / 3M	0B / 4K	15
●	orajre8-jvm	57f1c0ab78b5		0%		48M / 320M	2M / 3M	0B / 0B	13
●	orajre9-jvm	9df7d9e3eac9		0%		48M / 320M	2M / 3M	0B / 0B	15
●	orajre9-jvm-jlink	88281e8882f1		0%		35M / 320M	2M / 3M	0B / 116K	15

VERTX THREAD PER CORE - WEB SERVER (NETTY)

ctop - 00:03:18 CET 2 containers

	NAME	CID	CPU		MEM		NET RX/TX	IO R/W	PIDS
●	openjdk10ea-vertx	4a364d01d912		0%		79M / 320M	1M / 2M	0B / 0B	16
●	orajre9-vertx	ddb9ff9836b7		0%		79M / 320M	1M / 2M	0B / 4K	16

FINDINGS

No major diffrence Java 8 / 9 / 10 (in current test env).

To showcase "Compact Strings" a more realistic solution would be needed

CDS / AppCDS no major effect on overall memory, some effect on startup time
BUT: As the class cache can be shared; total gain will multiply by the number of nodes in cluster

Effects of AOT was hard to prove in test scenario

JLink has a significant effect on image size and positive effect on memory

Java 8/9 has limited cgroups awareness, better support coming in Java 10

Frameworks and servers make a HUGE difference on memory footprint

If resources (mem / CPU) are scarce - pick the right framework or look for alternatives;

http://callistaenterprise.se/blogg/teknik/2017/02/17/go-blog-series-part1/ 

... and - take micro-tests/benchmarks with a grain of salt!!

SUM UP

OPEN*

EVOLVING

NIMBLE

SCALABLE

THANKS!