Lean Spring Boot

Applications for The Cloud

Patrick Baumgartner

42talents GmbH, Zürich, Switzerland

@patbaumgartner

patrick.baumgartner@42talents.com

TALENTS

Abstract

Lean Spring Boot Applications for The Cloud

With the starters, Spring-Boot offers a functionality that allows you to set up a new software project with little effort and start programming right away. You don't have to worry about the dependencies, since the "right" ones are already preconfigured. But how can you, for example, optimize the start-up times and reduce the memory footprint and thus better prepare the application for the cloud?

In this talk, we will go into Spring-Boot features like Spring AOT, classpath exclusions, lazy spring beans, actuator, and more. In addition, we're also looking at switching to a different JVM and other tools. All state-of-the-art technology, of course.

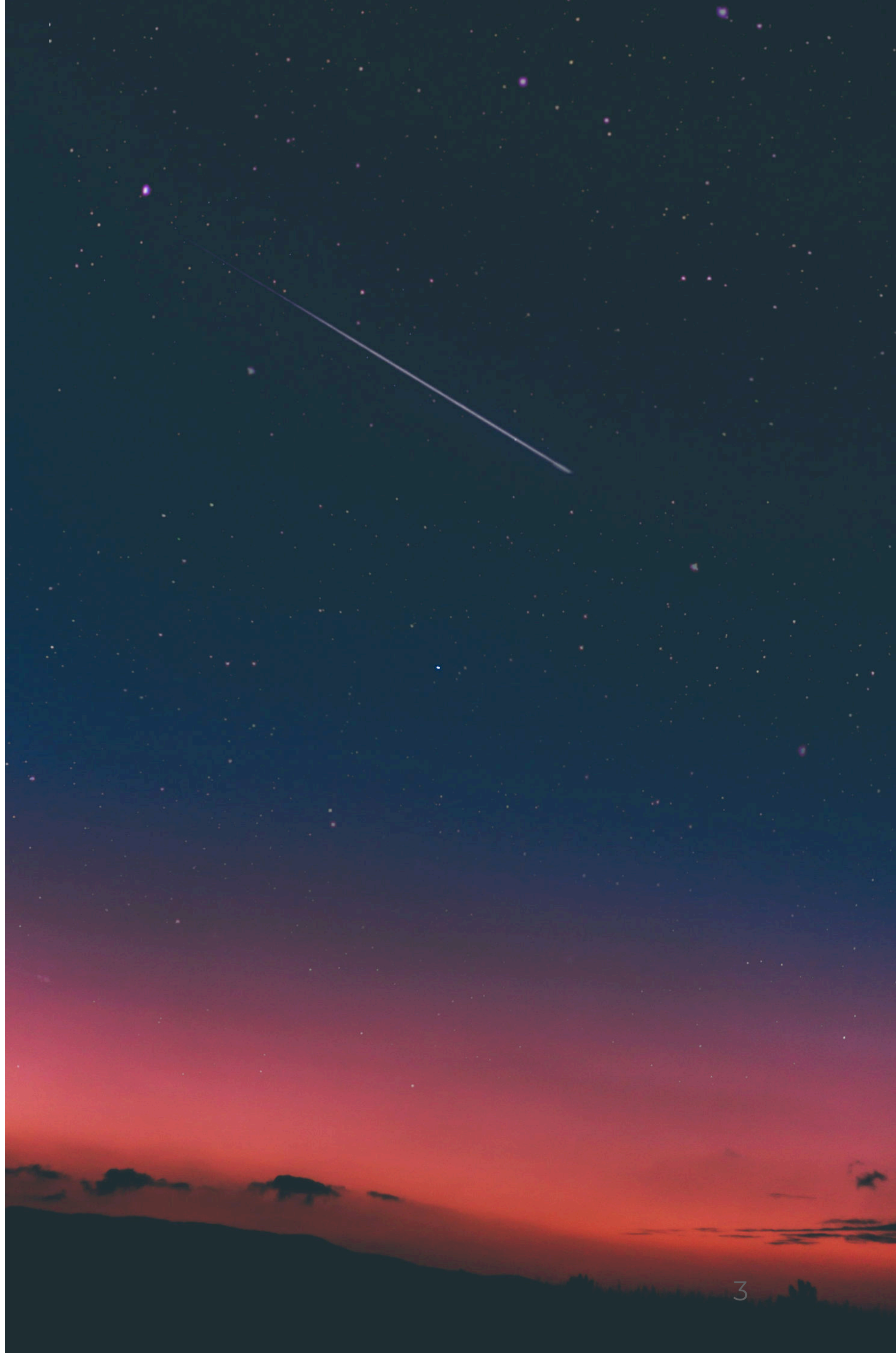
Let's make Spring Boot great again!

Lean Spring Boot

Applications for The Cloud

Patrick Baumgartner
42talents GmbH, Zürich, Switzerland

@patbaumgartner
patrick.baumgartner@42talents.com



SPRING IO

#springio24

Lean Spring Boot Applications for the Cloud

Patrick Baumgartner
@patbaumgartner

THE SPRING IO DEVELOPER CONFERENCE

MAY · 30-31 · 2024

SPRINGIO.NET

⚠️ WARNING:

Numbers shown in this talk are **not** based on **real data** but **only estimates** and **assumptions** made by the **author** for **educational purposes** only.

Introduction

Patrick Baumgartner

Technical Agile Coach @ 42talents

My focus is on the development of software solutions *with* humans.

Coaching, Architecture, Development, Reviews, and Training.

Lecturer @ Zurich University of Applied Sciences ZHAW

@patbaumgartner

What is the problem?

Why this talk?

Java 😉 & Spring Boot ❤️

Requirements

When Deploying to a Cloud

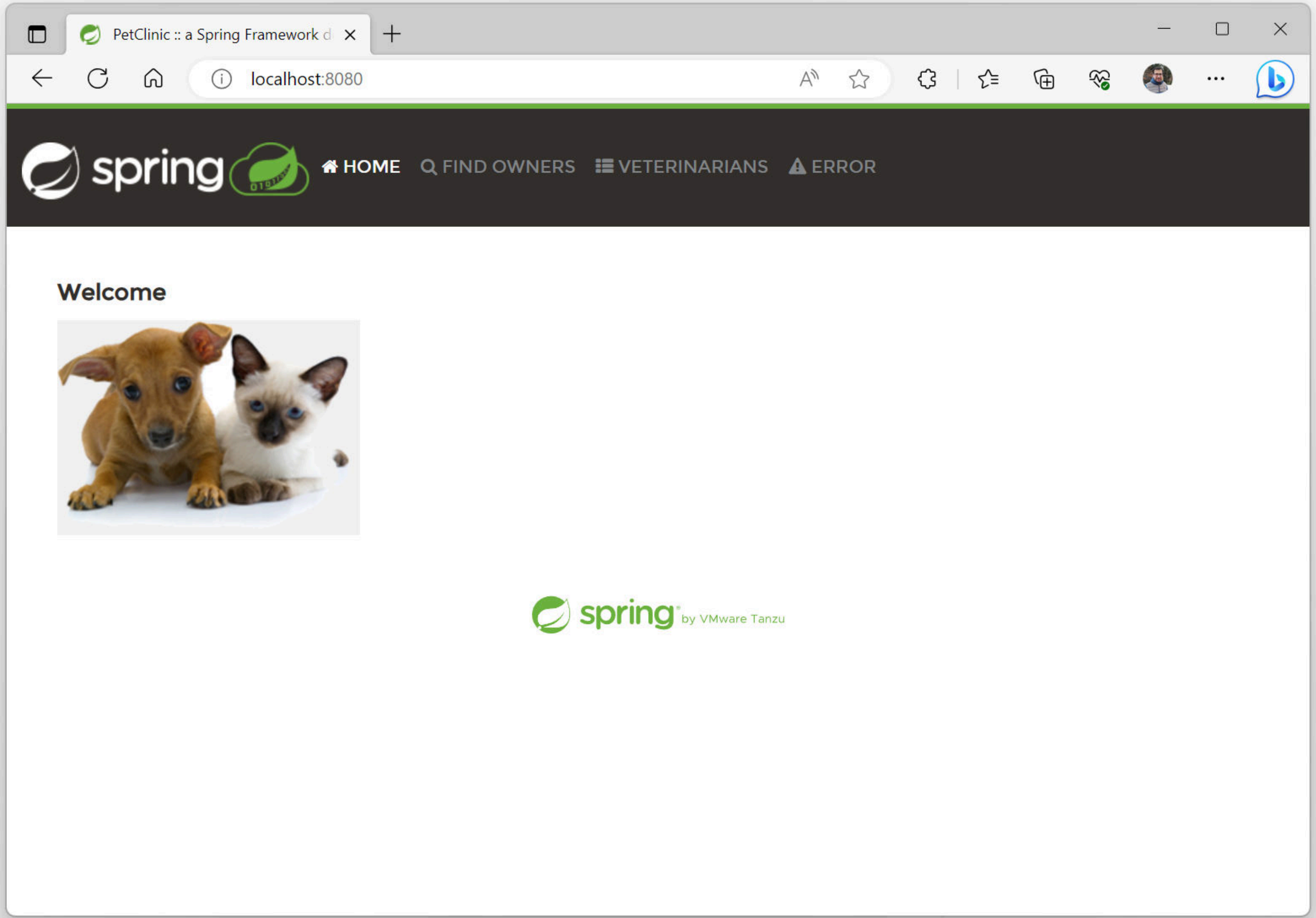
- How many vCPUs will my application need?
- How much RAM do I need?
- How much storage do I need?
- What technology stack should I use?
- What type of application do we build?

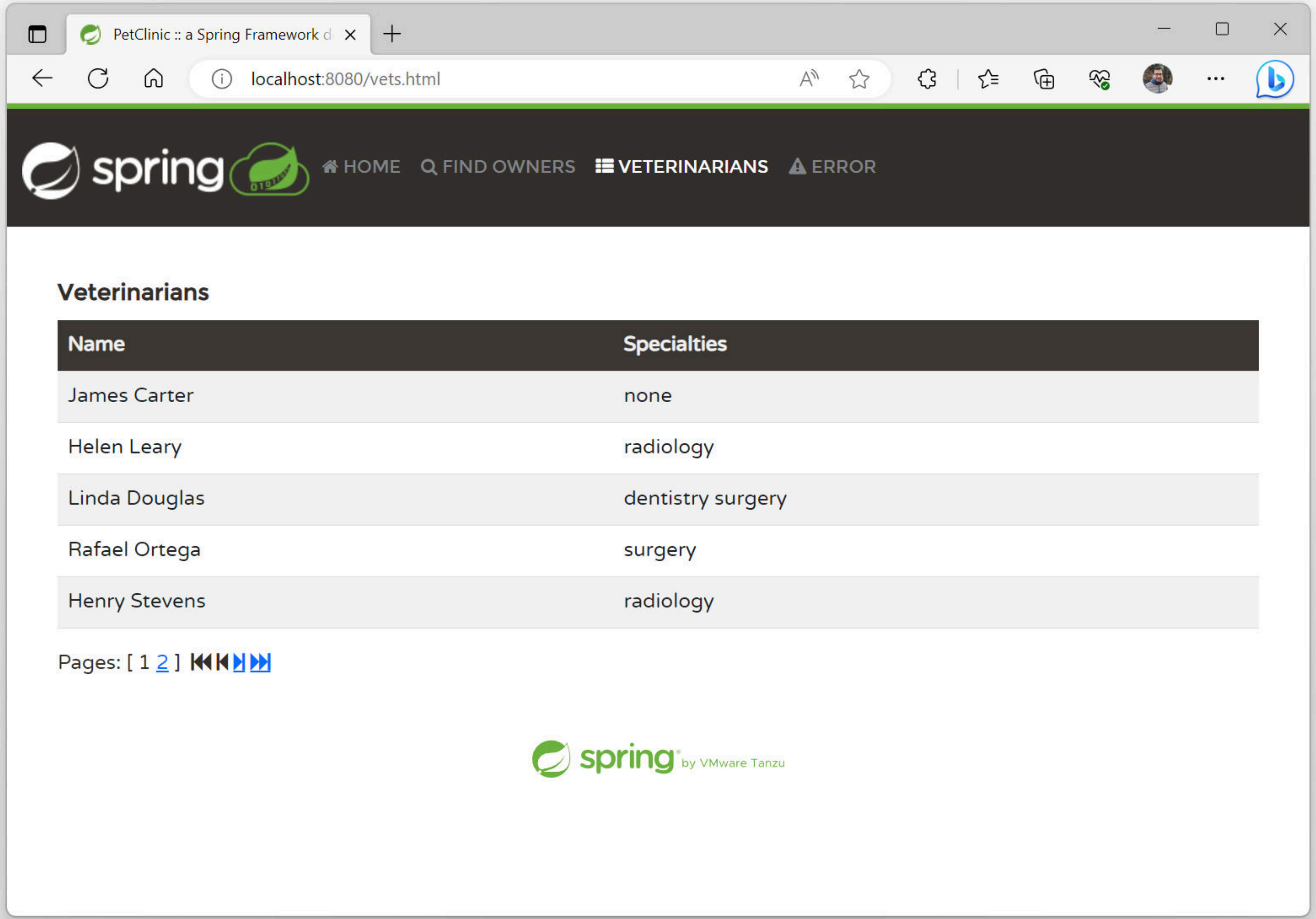
Agenda

Agenda

- Spring PetClinic & Baseline for comparison
- JVM Optimisations
- Spring Boot Optimisations
- Application Optimisations
- Other Runtimes
- Conclusions
- Some simple optimisations applied (OpenJDK examples)

Spring PetClinic


PetClinic :: a Spring Framework d
localhost:8080
HOME
FIND OWNERS
VETERINARIANS
ERROR
Welcome
spring by VMware Tanzu



Veterinarians

Name	Specialties
James Carter	none
Helen Leary	radiology
Linda Douglas	dentistry surgery
Rafael Ortega	surgery
Henry Stevens	radiology

Pages: [1 2]

Spring Petclinic Community

- spring-framework-petclinic
- spring-petclinic-angular(js)
- spring-petclinic-rest
- spring-petclinic-graphql
- spring-petclinic-microservices
- spring-petclinic-data-jdbc
- spring-petclinic-cloud
- spring-petclinic-mustache
- spring-petclinic-kotlin
- spring-petclinic-reactive
- spring-petclinic-hilla
- spring-petclinic-angularjs
- spring-petclinic-vaadin-flow
- spring-petclinic-reactjs
- spring-petclinic-htmx
- spring-petclinic-istio
- …

Projects on GitHub: https://github.com/spring-petclinic

NO!

The official Spring PetClinic! 🐾🏥

Based on Spring Boot, Caffeine, Thymeleaf, Spring Data JPA, H2 and Spring MVC ...

Project on GitHub: https://github.com/spring-projects/spring-petclinic

Optimisation Experiments

Baseline

Technology Stack

- OCI Container built with Buildpacks
- Java JDK 17 LTS
- Spring Boot 3.3.0
- Testcontainers
- DB migration using SQL scripts

Examination

- Build time
- Startup time
- Resource usage
- Container image size
- Throughput

Buildpacks.io

paketo

buildpacks

paketo BUILDPACKS

Your app,
in your favorite language,
ready to run in the cloud

Java GraalVM GO .NET Core NGINX node js php python

```
Calculating JVM memory based on 23635928K available memory
For more information on this calculation, see https://paketo.io/docs/reference/java-reference/#memory-calculator
Calculated JVM Memory Configuration: -XX:MaxDirectMemorySize=10M -Xmx23002593K -XX:MaxMetaspaceSize=121334K -XX:ReservedCodeCacheSize=240M -Xss1M
(Total Memory: 23635928K, Thread Count: 250, Loaded Class Count: 19008, Headroom: 0%)
Enabling Java Native Memory Tracking
Adding 137 container CA certificates to JVM truststore
Spring Cloud Bindings Enabled
Picked up JAVA_TOOL_OPTIONS: -Djava.security.properties=/layers/paketo-buildpacks_bellsoft-liberica/java-security-properties/java-security.properties
-XX:+ExitOnOutOfMemoryError -XX:MaxDirectMemorySize=10M -Xmx23002593K -XX:MaxMetaspaceSize=121334K -XX:ReservedCodeCacheSize=240M -Xss1M -XX:
+UnlockDiagnosticVMOptions -XX:NativeMemoryTracking=summary -XX:+PrintNMTStatistics -Dorg.springframework.cloud.bindings.boot.enable=true


              |\      _,,,--,,_
             /,`.-'`'   ._  \-;;,_
  _______ __|,4-  ) )_   .;.(__`'-'__     ___ __    _ ___ _______
 |       | '---''(_/._)-'(_\_)   |   |   |   |  |  | |   |       |
 |    _  |    ___|_     _|       |   |   |   |   |_| |   |       | __ _ _
 |   |_| |   |___  |   | |       |   |   |   |       |   |       | \ \ \ \
 |    ___|    ___| |   | |      _|   |___|   |  _    |   |      _|  \ \ \ \
 |   |   |   |___  |   | |     |_|       |   | | |   |   |     |_    ) ) ) )
 |___|   |_______| |___| |_______|_______|___|_|  |__|___|_______|  / / / /
 ==================================================================/_/_/_/

:: Built with Spring Boot :: 3.3.0


2024-05-26T12:00:51.738Z  INFO 1 --- [           main] o.s.s.petclinic.PetClinicApplication     : Starting PetClinicApplication v3.3.0-SNAPSHOT using Java 17.0.11 with PID 1
                                                 (/workspace/BOOT-INF/classes started by cnb in /workspace)
2024-05-26T12:00:51.741Z  INFO 1 --- [           main] o.s.s.petclinic.PetClinicApplication     : No active profile set, falling back to 1 default profile: "default"
2024-05-26T12:00:53.027Z  INFO 1 --- [           main] .s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data JPA repositories in DEFAULT mode.
2024-05-26T12:00:53.075Z  INFO 1 --- [           main] .s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository scanning in 42 ms. Found 2 JPA repository interfaces.
2024-05-26T12:00:53.961Z  INFO 1 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port 8080 (http)
2024-05-26T12:00:53.977Z  INFO 1 --- [           main] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
2024-05-26T12:00:53.977Z  INFO 1 --- [           main] o.apache.catalina.core.StandardEngine    : Starting Servlet engine: [Apache Tomcat/10.1.24]
2024-05-26T12:00:54.058Z  INFO 1 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
2024-05-26T12:00:54.060Z  INFO 1 --- [           main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 2211 ms
2024-05-26T12:00:54.599Z  INFO 1 --- [           main] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Starting...
2024-05-26T12:00:54.765Z  INFO 1 --- [           main] com.zaxxer.hikari.pool.HikariPool        : HikariPool-1 - Added connection conn0: url=jdbc:h2:mem:7a64d2ac-5851-43d8-bf48-ecd4dff96898 user=SA
2024-05-26T12:00:54.767Z  INFO 1 --- [           main] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Start completed.
2024-05-26T12:00:54.959Z  INFO 1 --- [           main] o.hibernate.jpa.internal.util.LogHelper  : HHH000204: Processing PersistenceUnitInfo [name: default]
2024-05-26T12:00:54.998Z  INFO 1 --- [           main] org.hibernate.Version                    : HHH000412: Hibernate ORM core version 6.5.2.Final
2024-05-26T12:00:55.025Z  INFO 1 --- [           main] o.h.c.internal.RegionFactoryInitiator    : HHH000026: Second-level cache disabled
2024-05-26T12:00:55.279Z  INFO 1 --- [           main] o.s.o.j.p.SpringPersistenceUnitInfo      : No LoadTimeWeaver setup: ignoring JPA class transformer
2024-05-26T12:00:56.661Z  INFO 1 --- [           main] o.h.e.t.j.p.i.JtaPlatformInitiator       : HHH000489: No JTA platform available (set 'hibernate.transaction.jta.platform' to enable JTA platform integration)
2024-05-26T12:00:56.665Z  INFO 1 --- [           main] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persistence unit 'default'
2024-05-26T12:00:57.074Z  INFO 1 --- [           main] o.s.d.j.r.query.QueryEnhancerFactory     : Hibernate is in classpath; If applicable, HQL parser will be used.
2024-05-26T12:00:58.316Z  INFO 1 --- [           main] o.s.b.a.e.web.EndpointLinksResolver      : Exposing 14 endpoints beneath base path '/actuator'
2024-05-26T12:00:58.385Z  INFO 1 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port 8080 (http) with context path '/'
2024-05-26T12:00:58.400Z  INFO 1 --- [           main] o.s.s.petclinic.PetClinicApplication     : Started PetClinicApplication in 7.086 seconds (process running for 7.68)
```

1000x better than your regular **Dockerfile** 📶 …

… more **secure** 🔒 and maintained by the **Buildpacks community.**

See also: https://buildpacks.io/ and https://www.cncf.io/projects/buildpacks/

Startup Reporting

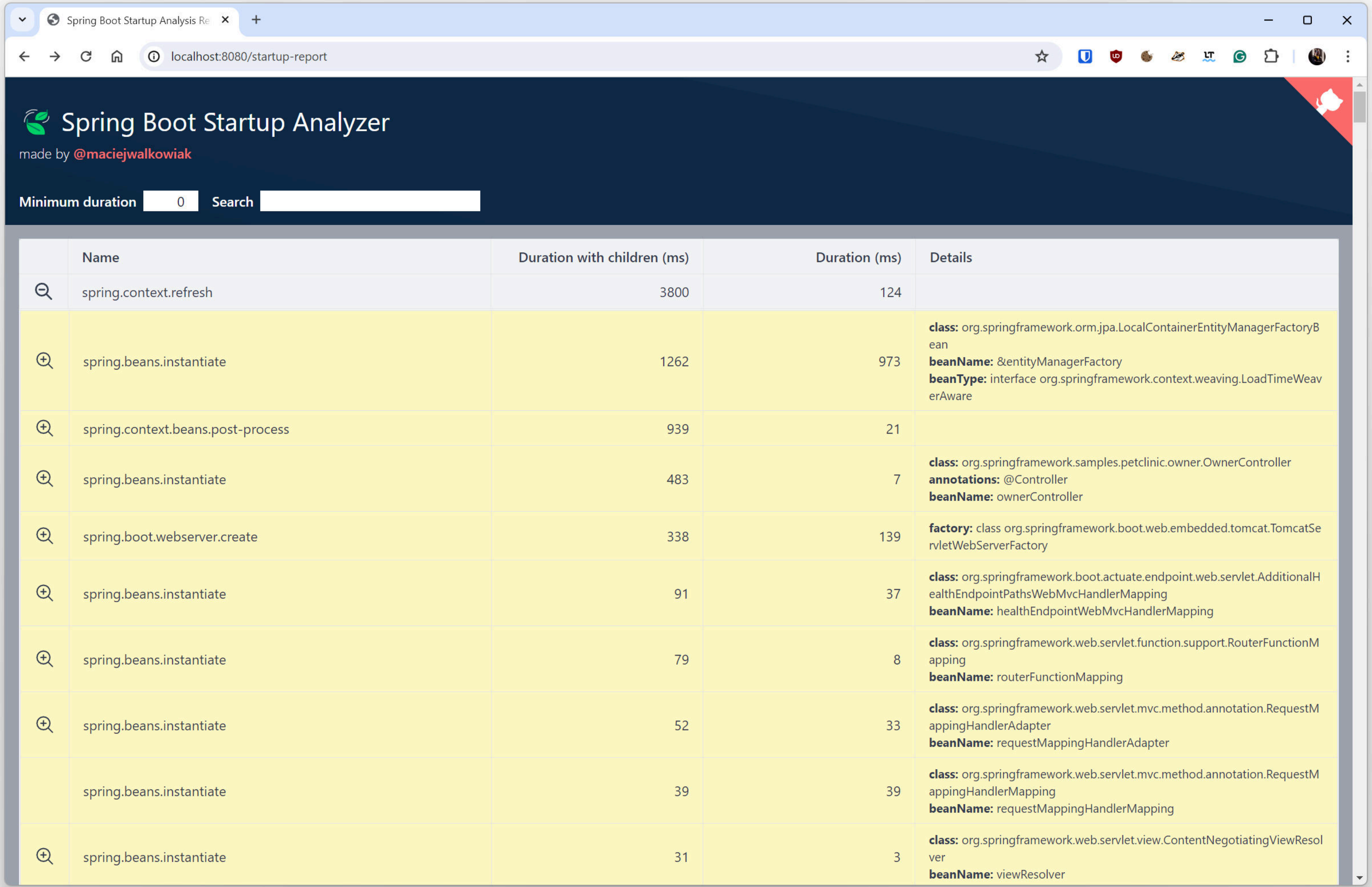
Spring Boot Startup Report

By Maciej Walkowiak

- Startup report available in runtime as an interactive HTML page
- Generation of startup reports in integration tests
- Flame chart for timings
- Search by class or annotation

```
<dependency>
    <groupId>com.maciejwalkowiak.spring</groupId>
    <artifactId>spring-boot-startup-report</artifactId>
    <version>0.2.0</version>
    <optional>true</optional>
</dependency>
```

https://github.com/maciejwalkowiak/spring-boot-startup-report

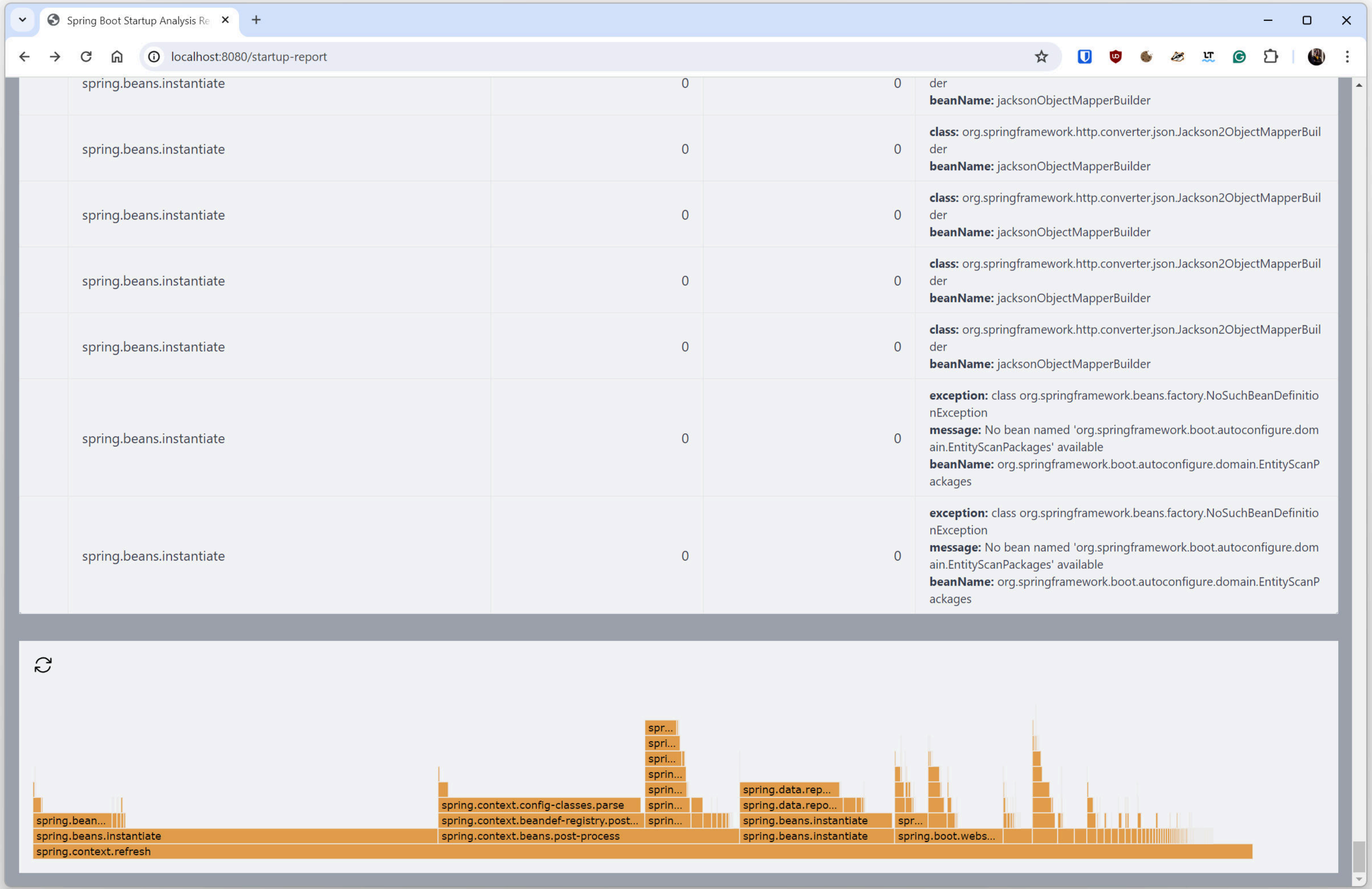

Spring Boot Startup Analysis Re
localhost:8080/startup-report

Spring Boot Startup Analyzer

made by @maciejwalkowiak

Minimum duration 0 Search

	Name	Duration with children (ms)	Duration (ms)	Details
🔍	spring.context.refresh	3800	124	
🔍	spring.beans.instantiate	1262	973	**class:** org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean **beanName:** &entityManagerFactory **beanType:** interface org.springframework.context.weaving.LoadTimeWeaverAware
🔍	spring.context.beans.post-process	939	21	
🔍	spring.beans.instantiate	483	7	**class:** org.springframework.samples.petclinic.owner.OwnerController **annotations:** @Controller **beanName:** ownerController
🔍	spring.boot.webserver.create	338	139	**factory:** class org.springframework.boot.web.embedded.tomcat.TomcatServletWebServerFactory
🔍	spring.beans.instantiate	91	37	**class:** org.springframework.boot.actuate.endpoint.web.servlet.AdditionalHealthEndpointPathsWebMvcHandlerMapping **beanName:** healthEndpointWebMvcHandlerMapping
🔍	spring.beans.instantiate	79	8	**class:** org.springframework.web.servlet.function.support.RouterFunctionMapping **beanName:** routerFunctionMapping
🔍	spring.beans.instantiate	52	33	**class:** org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter **beanName:** requestMappingHandlerAdapter
	spring.beans.instantiate	39	39	**class:** org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerMapping **beanName:** requestMappingHandlerMapping
🔍	spring.beans.instantiate	31	3	**class:** org.springframework.web.servlet.view.ContentNegotiatingViewResolver **beanName:** viewResolver


Spring Boot Startup Analysis Re
localhost:8080/startup-report

	spring.beans.instantiate	0	0	der **beanName:** jacksonObjectMapperBuilder
	spring.beans.instantiate	0	0	**class:** org.springframework.http.converter.json.Jackson2ObjectMapperBuil der **beanName:** jacksonObjectMapperBuilder
	spring.beans.instantiate	0	0	**class:** org.springframework.http.converter.json.Jackson2ObjectMapperBuil der **beanName:** jacksonObjectMapperBuilder
	spring.beans.instantiate	0	0	**class:** org.springframework.http.converter.json.Jackson2ObjectMapperBuil der **beanName:** jacksonObjectMapperBuilder
	spring.beans.instantiate	0	0	**class:** org.springframework.http.converter.json.Jackson2ObjectMapperBuil der **beanName:** jacksonObjectMapperBuilder
	spring.beans.instantiate	0	0	**exception:** class org.springframework.beans.factory.NoSuchBeanDefinitio nException **message:** No bean named 'org.springframework.boot.autoconfigure.dom ain.EntityScanPackages' available **beanName:** org.springframework.boot.autoconfigure.domain.EntityScanP ackages
	spring.beans.instantiate	0	0	**exception:** class org.springframework.beans.factory.NoSuchBeanDefinitio nException **message:** No bean named 'org.springframework.boot.autoconfigure.dom ain.EntityScanPackages' available **beanName:** org.springframework.boot.autoconfigure.domain.EntityScanP ackages

Benchmarks

Benchmarks

- Build
  - Maven build time
  - Artifact / Container Image size
- Startup
  - Startup time
  - Memory usage
- Throughput & Latency
  - `wrk2 -t4 -c200 -d60s -R2000 --latency <HOST>`
  - 1 min warmup, 1min measurement
  - Docker container with 4 vCPU and 1 GB RAM

No Optimizing - Baseline

No Optimizing - Baseline JDK 17

- Spring PetClinic (no adjustments)
- Bellsoft Liberica JDK 17.0.11
- Java Memory Calculator

```
sdk use java 17.0.11-librca

mvn spring-boot:build-image

docker run -p 8080:8080 -t spring-petclinic:3.3.0-SNAPSHOT
```

	Build	**Image**	**Startup**	**Initial RAM**	**Throughput**	**RAM**	**50%**	**75%**	**90%**	**99%**	**99.9%**
⏱ Java 17	68s	359MB	4.317s	302MB	1997/s	478MB	6ms	10ms	15ms	31ms	99ms

No Optimizing - Baseline JDK 21

- Spring PetClinic (JDK 21 adjustments)
- Bellsoft Liberica JDK 21.0.3
- Java Memory Calculator

```
sdk use java 21.0.3-librca

mvn -Djava.version=21 spring-boot:build-image \
    -Dspring-boot.build-image.imageName=spring-petclinic:3.3.0-SNAPSHOT-jdk21

docker run -p 8080:8080 -t spring-petclinic:3.3.0-SNAPSHOT-jdk21
```

	Build	Image	Startup	Initial RAM	Throughput	RAM	50%	75%	90%	99%	99.9%
⏱ Java 17	68s	359MB	4.317s	302MB	1997/s	478MB	6ms	10ms	15ms	31ms	99ms
Java 21	68s	387MB	4.517s	302MB	1996/s	496MB	5ms	9ms	13ms	30ms	98ms

No Optimizing - Baseline JDK 22

- Spring PetClinic (JDK 22 adjustments)
- Bellsoft Liberica JDK 22.0.1
- Java Memory Calculator

```
sdk use java 22.0.1-librca

mvn -Djava.version=22 spring-boot:build-image \
    -Dspring-boot.build-image.imageName=spring-petclinic:3.3.0-SNAPSHOT-jdk22

docker run -p 8080:8080 -t spring-petclinic:3.3.0-SNAPSHOT-jdk22
```

	Build	Image	Startup	Initial RAM	Throughput	RAM	50%	75%	90%	99%	99.9%
⏱ Java 17	68s	359MB	4.317s	302MB	1997/s	478MB	6ms	10ms	15ms	31ms	99ms
Java 22	71s	383MB	4.453s	306MB	1994/s	480MB	5ms	10ms	16ms	30ms	89ms

JVM Optimisations

-noverify

-noverify

The verifier is turned off because some of the bytecode rewriting stretches the meaning of some bytecodes - in a way that doesn't bother the JVM, but does bother the verifier.

Warning: The `-Xverify:none` and `-noverify` options are deprecated in JDK 13 and are likely to be removed in a future release.

```
docker run -p 8080:8080 -e "JAVA_TOOL_OPTIONS=-noverify" \
  -t spring-petclinic:3.3.0-SNAPSHOT
```

	Build	Image	Startup	Initial RAM	Throughput	RAM	50%	75%	90%	99%	99.9%
⏱ Java 17	68s	359MB	4.317s	302MB	1997/s	478MB	6ms	10ms	15ms	31ms	99ms
Java 17	-	-	3.92s	298MB	1997/s	462MB	4ms	8ms	13ms	28ms	87ms

-XX:TieredStopAtLevel=1

-XX:TieredStopAtLevel=1

Tiered compilation is enabled by default since Java 8. Unless explicitly specified, the JVM decides which JIT compiler to use based on our CPU. For multi-core processors or 64-bit VMs, the JVM will select C2.

To disable C2 and use only the C1 compiler with no profiling overhead, we can use the `-XX:TieredStopAtLevel=1` parameter.

```
docker run -p 8080:8080 -e "JAVA_TOOL_OPTIONS=-XX:TieredStopAtLevel=1" \
  -t spring-petclinic:3.3.0-SNAPSHOT
```

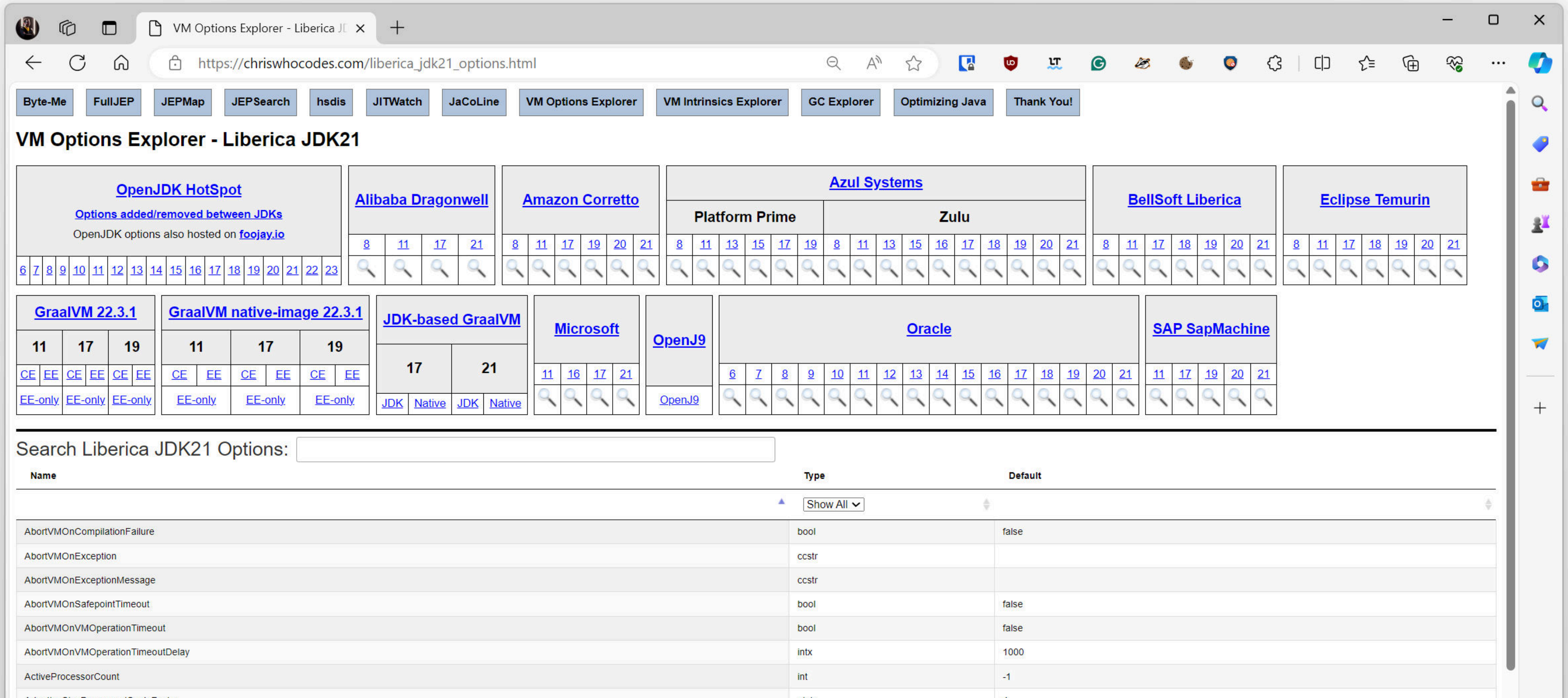
It will slow down the JIT later at the expense of the saved startup time!

	Build	Image	Startup	Initial RAM	T...	RAM	50%	75%	90%	99%	99.9%
⏱ Java 17	68s	359MB	4.317s	302MB	1997/s	478MB	6ms	10ms	15ms	31ms	99ms
Java 17	-	-	3.878s	216MB	1631/s	337MB	6173ms	8570ms	10470ms	13423ms	15157ms

VM Options Explorer

VM Options Explorer

https://chriswhocodes.com



VM Options Explorer - Liberica JDK21

https://chriswhocodes.com/liberica_jdk21_options.html

Byte-Me | FullJEP | JEPMap | JEPSearch | hsdis | JITWatch | JaCoLine | VM Options Explorer | VM Intrinsics Explorer | GC Explorer | Optimizing Java | Thank You!

VM Options Explorer - Liberica JDK21

OpenJDK HotSpot
Options added/removed between JDKs
OpenJDK options also hosted on foojay.io
6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23

Alibaba Dragonwell: 8 | 11 | 17 | 21

Amazon Corretto: 8 | 11 | 17 | 19 | 20 | 21

Azul Systems
Platform Prime: 8 | 11 | 13 | 15 | 17 | 19
Zulu: 8 | 11 | 13 | 15 | 16 | 17 | 18 | 19 | 20 | 21

BellSoft Liberica: 8 | 11 | 17 | 18 | 19 | 20 | 21

Eclipse Temurin: 8 | 11 | 17 | 18 | 19 | 20 | 21

GraalVM 22.3.1
11: CE | EE | EE-only
17: CE | EE | EE-only
19: CE | EE | EE-only

GraalVM native-image 22.3.1
11: CE | EE | EE-only
17: CE | EE | EE-only
19: CE | EE | EE-only

JDK-based GraalVM
17: JDK | Native
21: JDK | Native

Microsoft: 11 | 16 | 17 | 21

OpenJ9: OpenJ9

Oracle: 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21

SAP SapMachine: 11 | 17 | 19 | 20 | 21

Search Liberica JDK21 Options:

Name	Type	Default
	Show All	
AbortVMOnCompilationFailure	bool	false
AbortVMOnException	ccstr	
AbortVMOnExceptionMessage	ccstr	
AbortVMOnSafepointTimeout	bool	false
AbortVMOnVMOperationTimeout	bool	false
AbortVMOnVMOperationTimeoutDelay	intx	1000
ActiveProcessorCount	int	-1

Spring Boot Optimisations

Lazy Spring Beans

Lazy Spring Beans (1)

Configure lazy initialisation throughout your application. A Spring Boot property makes all beans lazy by default, initialising them only when needed. You can use `@Lazy` to override this behaviour, e.g. `@Lazy(false)`.

```
docker run -p 8080:8080  \
  -e spring.main.lazy-initialization=true  \
  -e spring.data.jpa.repositories.bootstrap-mode=lazy \
  -t spring-petclinic:3.3.0-SNAPSHOT
```

	Build	Image	Startup	Initial RAM	Throughput	RAM	50%	75%	90%	99%	99.9%
⏱ Java 17	68s	359MB	4.317s	302MB	1997/s	478MB	6ms	10ms	15ms	31ms	99ms
Java 17	-	-	3.347s	258MB	1992/s	468MB	5ms	10ms	15ms	31ms	97ms

Lazy Spring Beans (2)

Pros

- Faster startup useful in cloud environments
- Application startup is a CPU-intensive task. Spreads load over time

Cons

- Initial requests may take longer
- Class loading problems and misconfigurations not detected at startup
- Beans creation errors only be found when the bean is loaded

Fixing Spring Boot Config Location

Fixing Spring Boot Config Location

Determine the location of the Spring Boot configuration file(s).
Considered in the following order (`application.properties` and `YAML` variants)

- Application properties packaged in your jar
- Profile-specific application properties packaged within your jar
- Application properties outside your packaged jar
- Profile-specific application properties outside your packaged jar

```
docker run -p 8080:8080 -e spring.config.location=classpath:application.properties \
  -t spring-petclinic:3.3.0-SNAPSHOT
```

	Build	Image	Startup	Initial RAM	Throughput	RAM	50%	75%	90%	99%	99.9%
⏱ Java 17	68s	359MB	4.317s	302MB	1997/s	478MB	6ms	10ms	15ms	31ms	99ms
Java 17	-	-	4.308s	307MB	1989/s	486MB	6ms	11ms	16ms	32ms	100ms

No Spring Boot Actuators

No Spring Boot Actuators

Don't use actuators if you can afford not to 😊.

- Number of Spring Beans
  - Spring Pet Clinic with actuators: 452
  - Spring Pet Clinic without actuators: 276 🔥

```
sdk use java 17.0.11-librca

mvn spring-boot:build-image

docker run -p 8080:8080 -t spring-petclinic-no-actuator:3.3.0-SNAPSHOT
```

	Build	Image	Startup	Initial RAM	Throughput	RAM	50%	75%	90%	99%	99.9%
⏱ Java 17	68s	359MB	4.317s	302MB	1997/s	478MB	6ms	10ms	15ms	31ms	99ms
Java 17	66s	356MB	3.669s	266MB	1997/s	448MB	4ms	8ms	12ms	33ms	98ms

Ahead-of-Time Processing (AOT)

Ahead-of-Time Processing (AOT) (1)

Spring AOT is a process that analyses your application at build time and generates an optimised version of it.

As the `BeanFactory` is fully prepared at build time, conditions are also evaluated. E.g. in Configurations, Component- & Entity-Scan, `@Profile`, `@Conditional`, etc.

Spring AOT serves as a replacement for `spring-context-indexer` since Spring Framework 6.1 and Spring Boot 3.2.

Ahead-of-Time Processing (AOT) (2)

```
<plugin>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-maven-plugin</artifactId>
    <configuration>
      <image>
        <env>
          <BP_SPRING_AOT_ENABLED>true</BP_SPRING_AOT_ENABLED>
        </env>
      </image>
    </configuration>
    <executions>
      <execution>
          <id>process-aot</id>
          <goals>
              <goal>process-aot</goal>
          </goals>
      </execution>
    </executions>
</plugin>
```

Ahead-of-Time Processing (AOT) (3)

We create a new container image with the AOT-processed application. The Buildpack enables the property `spring.aot.enabled=true`

```
sdk use java 17.0.11-librca

mvn spring-boot:build-image

docker run -p 8080:8080 -t spring-petclinic-aot:3.3.0-SNAPSHOT
```

	Build	Image	Startup	Initial RAM	Throughput	RAM	50%	75%	90%	99%	99.9%
⏱ Java 17	68s	359MB	4.317s	302MB	1997/s	478MB	6ms	10ms	15ms	31ms	99ms
Java 17	74s	360MB	4.198s	285MB	1996/s	472MB	6ms	11ms	16ms	35ms	102ms

Disabling JMX

Disabling JMX

JMX is `spring.jmx.enabled=false` by default in Spring Boot since 2.2.0 and later. Setting `BPL_JMX_ENABLED=true` and `BPL_JMX_PORT=9999` on the container will add the following arguments to the `java` command.

```
-Djava.rmi.server.hostname=127.0.0.1
-Dcom.sun.management.jmxremote.authenticate=false
-Dcom.sun.management.jmxremote.ssl=false
-Dcom.sun.management.jmxremote.port=9999
-Dcom.sun.management.jmxremote.rmi.port=9999
```

```
docker run -p 8080:8080 -p 9999:9999 \
  -e BPL_JMX_ENABLED=false \
  -e BPL_JMX_PORT=9999 \
  -e spring.jmx.enabled=false \
  -t spring-petclinic:3.3.0-SNAPSHOT
```

I ❤️

Spring Boot 🍃 & Buildpacks

Application Optimisations

Dependency Cleanup

Dependency Cleanup (2)

DepClean detects all unused dependencies declared in the `pom.xml` file of a project and creates a `pom-debloated.xml`. The generated report shows possible unused dependencies.

```xml
<plugin>
  <groupId>se.kth.castor</groupId>
  <artifactId>depclean-maven-plugin</artifactId>
  <version>2.0.6</version>
  <executions>
    <execution>
      <goals>
        <goal>depclean</goal>
      </goals>
    </execution>
  </executions>
</plugin>
```

```
mvn se.kth.castor:depclean-maven-plugin:2.0.6:depclean -DfailIfUnusedDirect=true -DignoreScopes=provided,test,runtime,system,import
```

```
...
-------------------------------------------------------
 D E P C L E A N   A N A L Y S I S   R E S U L T S
-------------------------------------------------------
USED DIRECT DEPENDENCIES [9]:
        com.h2database:h2:2.2.224:runtime (2 MB)
        com.mysql:mysql-connector-j:8.3.0:runtime (2 MB)
        com.github.ben-manes.caffeine:caffeine:3.1.8:compile (868 KB)
        jakarta.xml.bind:jakarta.xml.bind-api:4.0.2:compile (127 KB)
        ...
USED TRANSITIVE DEPENDENCIES [93]:
        org.testcontainers:testcontainers:1.19.8:test (16 MB)
        org.hibernate.orm:hibernate-core:6.5.2.Final:compile (11 MB)
        net.bytebuddy:byte-buddy:1.14.16:runtime (4 MB)
        org.apache.tomcat.embed:tomcat-embed-core:10.1.24:compile (3 MB)
        org.aspectj:aspectjweaver:1.9.22:compile (2 MB)
        org.springframework.boot:spring-boot-autoconfigure:3.3.0:compile (1 MB)
        org.springframework:spring-web:6.1.8:compile (1 MB)
        ...
USED INHERITED DIRECT DEPENDENCIES [0]:
USED INHERITED TRANSITIVE DEPENDENCIES [0]:
POTENTIALLY UNUSED DIRECT DEPENDENCIES [11]:
        org.webjars.npm:bootstrap:5.3.3:compile (1 MB)
        org.webjars.npm:font-awesome:4.7.0:compile (665 KB)
        org.springframework.boot:spring-boot-devtools:3.3.0:test (198 KB)
        org.springframework.boot:spring-boot-docker-compose:3.3.0:test (191 KB)
        org.springframework.boot:spring-boot-starter-actuator:3.3.0:compile (4 KB)
        ...
POTENTIALLY UNUSED TRANSITIVE DEPENDENCIES [18]:
        org.springframework.boot:spring-boot-actuator-autoconfigure:3.3.0:compile (758 KB)
        org.springframework.boot:spring-boot-actuator:3.3.0:compile (658 KB)
        org.yaml:snakeyaml:2.2:compile (326 KB)
        org.attoparser:attoparser:2.0.7.RELEASE:compile (240 KB)
        org.thymeleaf:thymeleaf-spring6:3.1.2.RELEASE:compile (184 KB)
        org.hdrhistogram:HdrHistogram:2.2.1:runtime (171 KB)
        org.unbescape:unbescape:1.1.6.RELEASE:compile (169 KB)
        org.awaitility:awaitility:4.2.1:test (94 KB)
        io.micrometer:micrometer-jakarta9:1.13.0:compile (31 KB)
        ...
POTENTIALLY UNUSED INHERITED DIRECT DEPENDENCIES [0]:
POTENTIALLY UNUSED INHERITED TRANSITIVE DEPENDENCIES [0]:
[INFO] Analysis done in 0min 12s
```

Dependency Cleanup (2)

But there are some challenges:

- Component & Entity Scanning through Classpath Scanning
- Spring Boot uses `META-INF/spring-boot/org.springframework.boot.autoconfigure.AutoConfiguration.imports`
- Spring XML Configuration and `web.xml`

```
sdk use java 17.0.11-librca

mvn spring-boot:build-image

docker run -p 8080:8080 -t spring-petclinic-depclean:3.3.0-SNAPSHOT
```

	Build	Image	Startup	Initial RAM	Throughput	RAM	50%	75%	90%	99%	99.9%
⏱ Java 17	68s	359MB	4.317s	302MB	1997/s	478MB	6ms	10ms	15ms	31ms	99ms
Java 17	77s	354MB	3.514s	255MB	1996/s	440MB	5ms	9ms	13ms	35ms	70ms

Other Runtimes

JLink

JLink (1)

Jlink assembles and optimises a set of modules and their dependencies into a custom runtime image for your application.

```
$ jlink \
  --add-modules java.base, ... \
  --strip-debug \
  --no-man-pages \
  --no-header-files \
  --compress=2 \
  --output /javaruntime
```

```
$ /javaruntime/bin/java HelloWorld
Hello, World!
```

JLink (2)

```
<plugin>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-maven-plugin</artifactId>
  <configuration>
      <image>
          <env>
              <BP_JVM_JLINK_ENABLED>true</BP_JVM_JLINK_ENABLED>
          </env>
      </image>
  </configuration>
</plugin>
```

```
sdk use java 17.0.11-librca

mvn

doc
```

	Build	Image	Startup	Initial RAM	Throughput	RAM	50%	75%	90%	99%	99.9%
⏱ Java 17	68s	359MB	4.317s	302MB	1997/s	478MB	6ms	10ms	15ms	31ms	99ms
Java 17	86s	270MB	4.75s	309MB	1997/s	491MB	5ms	9ms	14ms	30ms	94ms
Java 21	88s	278MB	4.744s	316MB	1997/s	498MB	4ms	9ms	13ms	30ms	86ms

JLink (3)

```xml
<plugin>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-maven-plugin</artifactId>
    <configuration>
        <image>
            <env>
                <BP_JVM_JLINK_ENABLED>true</BP_JVM_JLINK_ENABLED>
                <BP_JVM_JLINK_ARGS>--add-modules jdk.management.agent,java.base,java.logging,
                java.xml,jdk.unsupported,java.sql,java.naming,java.desktop,java.management,
                java.security.jgss,java.instrument --compress=2 --no-header-files --no-man-pages
                --strip-debug</BP_JVM_JLINK_ARGS>
            </env>
        </image>
    </configuration>
</plugin>
```

	Build	Image	Startup	Initial RAM	Throughput	RAM	50%	75%	90%	99%	99.9%
⏱ Java 17	68s	359MB	4.317s	302MB	1997/s	478MB	6ms	10ms	15ms	31ms	99ms
Java 17	83s	258MB	4.748s	303MB	1994/s	475MB	5ms	9ms	14ms	32ms	101ms
Java 21	85s	265MB	4.793s	314MB	1996/s	489MB	6ms	10ms	15ms	34ms	105ms

Application Class Data Sharing (App CDS)

App CDS (1)

Class Data Sharing (CDS) is a JVM feature that can help reduce the startup time and memory footprint of Java applications. It allows classes to be pre-processed into a shared archive file that can be memory-mapped at runtime.

```xml
<plugin>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-maven-plugin</artifactId>
  <configuration>
    <image>
      <env>
        <BP_JVM_CDS_ENABLED>true</BP_JVM_CDS_ENABLED>
      </env>
    </image>
  </configuration>
</plugin>
```

App CDS (2)

To create the archive, two additional JVM flags must be specified:

- `-XX:ArchiveClassesAtExit=application.jsa` : creates the CDS archive on exit
- `-Dspring.context.exit=onRefresh` : starts and then immediately exits

Once the archive is available, the JVM can be started with the additional flag:

- `-XX:SharedArchiveFile=application.jsa` : to use it

```
sdk use java 17.0.11-sdk use java 17.0.11-librca

mvn

doc
```

	Build	Image	Startup	Initial RAM	Throughput	RAM	50%	75%	90%	99%	99.9%
⏱ Java 17	68s	359MB	4.317s	302MB	1997/s	478MB	6ms	10ms	15ms	31ms	99ms
Java 17	84s	510MB	2.868s	268MB	1997/s	444MB	5ms	9ms	13ms	34ms	100ms
Java 21	84s	537MB	2.92s	265MB	1997/s	456MB	5ms	10ms	15ms	36ms	116ms

I ❤️

Spring Boot 🍃 & Buildpacks

Eclipse OpenJ9

Docs Blog Slack Join Slack OpenJ9 Issues

Unleash the power of Java

Optimized to run Java™ applications cost-effectively in the cloud, Eclipse OpenJ9™ is a fast and efficient JVM that delivers power and performance when you need it most.

Optimized for the Cloud, for microservices and monoliths too!

Faster Startup

Faster Ramp-up, when deployed to cloud

Smaller

Our Story

Eclipse OpenJ9

```xml
<plugin>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-maven-plugin</artifactId>
  <configuration>
    <image>
        <buildpacks>
            <buildpack>gcr.io/paketo-buildpacks/eclipse-openj9:latest</buildpack>
            <!-- Used to inherit all the other Java buildpacks -->
            <buildpack>gcr.io/paketo-buildpacks/java:latest</buildpack>
        </buildpacks>
    </image>
  </configuration>
</plugin>
```

```
sdk

mvn

doc
```

	Build	Image	Startup	Initial RAM	Throughput	RAM	50%	75%	90%	99%	99.9%
⏱ Java 17	68s	359MB	4.317s	302MB	1997/s	478MB	6ms	10ms	15ms	31ms	99ms
Java 17	78s	334MB	7.708s	180MB	1990/s	362MB	7ms	12ms	17ms	33ms	58ms
Java 21	78s	353MB	7.795s	181MB	1997/s	366MB	9ms	14ms	20ms	43ms	86ms

GraalVM

GraalVM CE

```
<plugin>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-maven-plugin</artifactId>
  <configuration>
    <image>
        <buildpacks>
            <buildpack>gcr.io/paketo-buildpacks/graalvm:latest</buildpack>
            <!-- Used to inherit all the other Java buildpacks -->
            <buildpack>gcr.io/paketo-buildpacks/java:latest</buildpack>
        </buildpacks>
    </image>
  </configuration>
</plugin>
```

```
sdk

mvn

doc
```

	Build	Image	Startup	Initial RAM	Throughput	RAM	50%	75%	90%	99%	99.9%
⏱ Java 17	68s	359MB	4.317s	302MB	1997/s	478MB	6ms	10ms	15ms	31ms	99ms
Java 17	90s	756MB	4.658s	256MB	1990/s	446MB	6ms	10ms	15ms	30ms	105ms
Java 21	86s	742MB	4.713s	270MB	1997/s	453MB	5ms	10ms	14ms	32ms	105ms

GraalVM Oracle

```xml
<plugin>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-maven-plugin</artifactId>
  <configuration>
    <image>
        <buildpacks>
            <buildpack>gcr.io/paketo-buildpacks/oracle:latest</buildpack>
            <!-- Used to inherit all the other Java buildpacks -->
            <buildpack>gcr.io/paketo-buildpacks/java:latest</buildpack>
        </buildpacks>
    </image>
  </configuration>
</plugin>
```

```
sdk

mvn

doc
```

	Build	Image	Startup	Initial RAM	Throughput	RAM	50%	75%	90%	99%	99.9%
⏱ Java 17	68s	359MB	4.317s	302MB	1997/s	478MB	6ms	10ms	15ms	31ms	99ms
Java 17	77s	498MB	4.684s	301MB	1994/s	482MB	7ms	11ms	17ms	39ms	96ms
Java 21	77s	527MB	4.686s	296MB	1996/s	486MB	6ms	10ms	15ms	34ms	115ms

Other Buildpack Builders

Bellsoft Buildpack Builder (Java 17 only)

Bellsoft provides an optimised builder for Spring Boot applications. It uses the Bellsoft Alpaquita, Liberica JDK and the musl C library. A glibc version is also available.

```xml
<plugin>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-maven-plugin</artifactId>
  <configuration>
    <image>
      <builder>bellsoft/buildpacks.builder:musl</builder>
    </image>
  </configuration>
</plugin>
```

sdk
mvn
doc

	Build	Image	Startup	Initial RAM	Throughput	RAM	50%	75%	90%	99%	99.9%
⏱ Java 17	68s	359MB	4.317s	302MB	1997/s	478MB	6ms	10ms	15ms	31ms	99ms
musl	64s	178MB	5.141s	265MB	1996/s	413MB	6ms	10ms	15ms	38ms	96ms
glibc	65s	192MB	4.671s	299MB	1996/s	486MB	5ms	10ms	14ms	33ms	89ms

Buildpack Builder Tiny

It's based on Ubuntu Jammy, and works with current JAVA versions. While the base image (default) is bigger the tiny is intended to be used with GraalVM native images.

```
<plugin>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-maven-plugin</artifactId>
  <configuration>
    <image>
      <builder>paketobuildpacks/builder-jammy-tiny</builder>
    </image>
  </configuration>
</plugin>
```

```
sdk

mvn

doc
```

	Build	Image	Startup	Initial RAM	Throughput	RAM	50%	75%	90%	99%	99.9%
⏱ Java 17	68s	359MB	4.317s	302MB	1997/s	478MB	6ms	10ms	15ms	31ms	99ms
Java 17	72s	278MB	4.798s	294MB	1997/s	479MB	6ms	11ms	16ms	35ms	99ms
Java 21	69s	306MB	4.758s	297MB	1992/s	494MB	5ms	10ms	14ms	26ms	61ms

GraalVM Native Image

GraalVM Native Image (CE & Oracle)

A native image is a technology for building Java code into a standalone executable. This executable contains the application classes, classes from its dependencies, runtime library classes and statically linked native code from the JDK. The JVM is packaged in the native image, so there's no need for a Java Runtime Environment on the target system, but the build artifact is platform dependent.

```
mvn -Pnative spring-boot:build-image

docker run -p 8080:8080 -t spring-petclinic-native:3.3.0-SNAPSHOT
```

	Build	Image	Startup	Initial RAM	Throughput	RAM	50%	75%	90%	99%	99.9%
⏱ Java 17	68s	359MB	4.317s	302MB	1997/s	478MB	6ms	10ms	15ms	31ms	99ms
17 CE	367s	225MB	0.49s	255MB	1989/s	468MB	27ms	49ms	88ms	231ms	376ms
17 Oracle	516s	249MB	0.335s	237MB	1994/s	403MB	13ms	23ms	37ms	92ms	171ms

CRaC - OpenJDK

CRaC - OpenJDK (1)

CRaC (Coordinated Restore at Checkpoint) is a feature that allows you to take a snapshot of the state of a Java application and restart it from that state.

Currently only available from:

- Azul Zulu
- Bellsoft Liberica

The application starts within milliseconds!

CRaC - OpenJDK (2)

```
export JAVA_HOME=/opt/openjdk-17-crac+6_linux-x64/
export PATH=$JAVA_HOME/bin:$PATH
```

```
mvn clean verify
```

```
java -XX:CRaCCheckpointTo=crac-files -jar target/spring-petclinic-crac-3.3.0-SNAPSHOT.jar
```

```
jcmd target/spring-petclinic-crac-3.3.0-SNAPSHOT.jar JDK.checkpoint
```

```
java -XX:CRaCRestoreFrom=crac-files
```

CRaC - OpenJDK (3)

CRaC is currently in an experimental state and has the following limitations.

- Since Spring Boot 3.2 CraC support finalised
  - Spring Framework 6.1.0
- Only available for Linux x86 / ARM 64 Bit
- Azul Zulu Build of OpenJDK with CRaC support for development purposes
  - Available for Windows and macOS
  - Simulated checkpoint/restore mechanism for development and testing

Other JVM vendors have similar features, e.g. OpenJ9 with CRIU support.

I 😭

No CRaC Buildpacks for Spring Boot 🍃

Virtual Threads (Java 21)

Virtual Threads

A thread is the smallest unit of processing that can be scheduled. It runs concurrently with - and largely independently of - other such units. It is an instance of `java.lang.Thread`. There are two types of threads, platform threads and virtual threads.

```
sdk use java 21.0.3-librca

mvn spring-boot:build-image

docker run -e spring.threads.virtual.enabled=true \
  -p 8080:8080 -t spring-petclinic-virtual-threads:3.3.0-SNAPSHOT
```

	Build	Image	Startup	Initial RAM	Throughput	RAM	50%	75%	90%	99%	99.9%
⏱ Java 17	68s	359MB	4.317s	302MB	1997/s	478MB	6ms	10ms	15ms	31ms	99ms
Java 21	70s	387MB	4.9s	303MB	1992/s	485MB	4ms	8ms	11ms	31ms	97ms

Summary

Summary

- No Optimisations with JDK 17 & JDK 21
- JVM Tuning
- Lazy Spring Beans
- No Spring Boot Actuators
- Fix Spring Boot Config Location
- Disabling JMX
- Dependency Cleanup
- Ahead-of-Time Processing (AOT)
- JLink
- Other JVMs (Eclipse OpenJ9, GraalVM, OpenJDK with CRaC)
- GraalVM Native Image

Conclusions

Conclusions (1)

CPUs

- Your application may not need a full CPU at runtime.
- It will need several CPUs to start as fast as possible (at least 2, 4 is better).
- If you don't mind a slower startup, you can throttle the CPUs below 4.

See: https://spring.io/blog/2018/11/08/spring-boot-in-a-container

Conclusions (2)

Throughput

- Every application is different and has different requirements.
- Proper load testing can help find the optimal configuration for your application.

Conclusions (3)

Other Runtimes

- CRIU Support for OpenJDK and OpenJ9 is promising.
  - Supported by Spring since Spring Boot 3.2 / Spring Framework 6.1
- GraalVM Native Image is a great option for Java applications
  - But build times are long
  - The result is different from what you run in your IDE
- Eclipse OpenJ9 is an excellent option for running applications with less memory
  - But startup times are longer than with HotSpot.
- Depending on the distribution, you may get other exciting features.
  - Oracle GraalVM Enterprise Edition, Azul Platform Prime, IBM Semeru Runtime, ...

Conclusions (4)

Other Ideas

- CRaC (Coordinated Restore at Checkpoint)*
- Using an Obfuscator such as ProGuard*
- More JVM tuning (GC, Memory, etc.)
- Project Leyden

A Few Simple Optimisations Applied

A Few Simple Optimisations Applied

- Dependency Cleanup
  - DB Drivers, Spring Boot Actuator, Jackson, Tomcat Websocket, ...
- Bellsoft Builder (musl) / Base Builder Tiny
- JLink
- JVM Parameters (java-memory-calculator)
- Application Class Data Sharing (AppCDS)
- Spring AOT
- Lazy Spring Beans
- Fixing Spring Boot Config Location
- Virtual Threads

Java 17 & Bellsoft Builder (musl)

```
sdk use java 17.0.11-librca

mvn spring-boot:build-image

docker run -p 8080:8080 \
  -e spring.main.lazy-initialization=true \
  -e spring.data.jpa.repositories.bootstrap-mode=lazy \
  -e spring.config.location=classpath:application.properties \
  -t spring-petclinic-optimized-builder-bellsoft-musl:3.3.0-SNAPSHOT
```

	Build	Image	Startup	Initial RAM	Throughput	RAM	50%	75%	90%	99%	99.9%
⏱ Java 17	68s	359MB	4.317s	302MB	1997/s	478MB	6ms	10ms	15ms	31ms	99ms
Java 17	74s	143MB	2.341s	200MB	1994/s	365MB	6ms	12ms	18ms	57ms	112ms

Java 21 & Base Builder Tiny

```
sdk use java 21.0.3-librca

mvn spring-boot:build-image

docker run -p 8080:8080 \
  -e spring.threads.virtual.enabled=true \
  -e spring.main.lazy-initialization=true \
  -e spring.data.jpa.repositories.bootstrap-mode=lazy \
  -e spring.config.location=classpath:application.properties \
  -t spring-petclinic-optimized-builder-tiny-virtual-threads:3.3.0-SNAPSHOT
```

	Build	Image	Startup	Initial RAM	Throughput	RAM	50%	75%	90%	99%	99.9%
⏱ Java 17	68s	359MB	4.317s	302MB	1997/s	478MB	6ms	10ms	15ms	31ms	99ms
Java 21	95s	311MB	3.051s	241MB	1996/s	446MB	6ms	10ms	17ms	53ms	119ms

Did I miss something? 🧐

Let me/us know! 🙋

... or not! 🙊

Lean Spring Boot

Applications for The Cloud

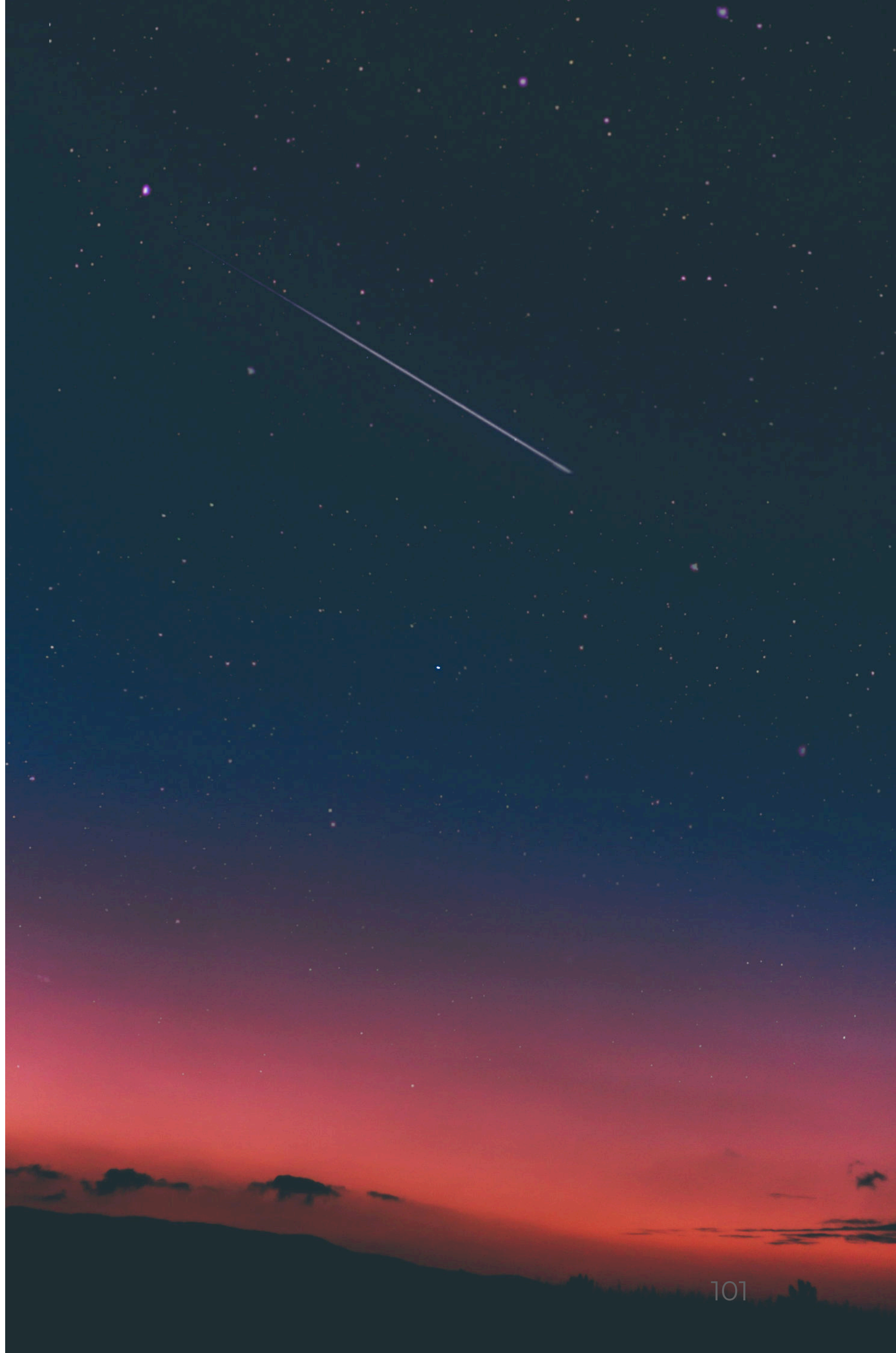
Patrick Baumgartner

42talents GmbH, Zürich, Switzerland

@patbaumgartner

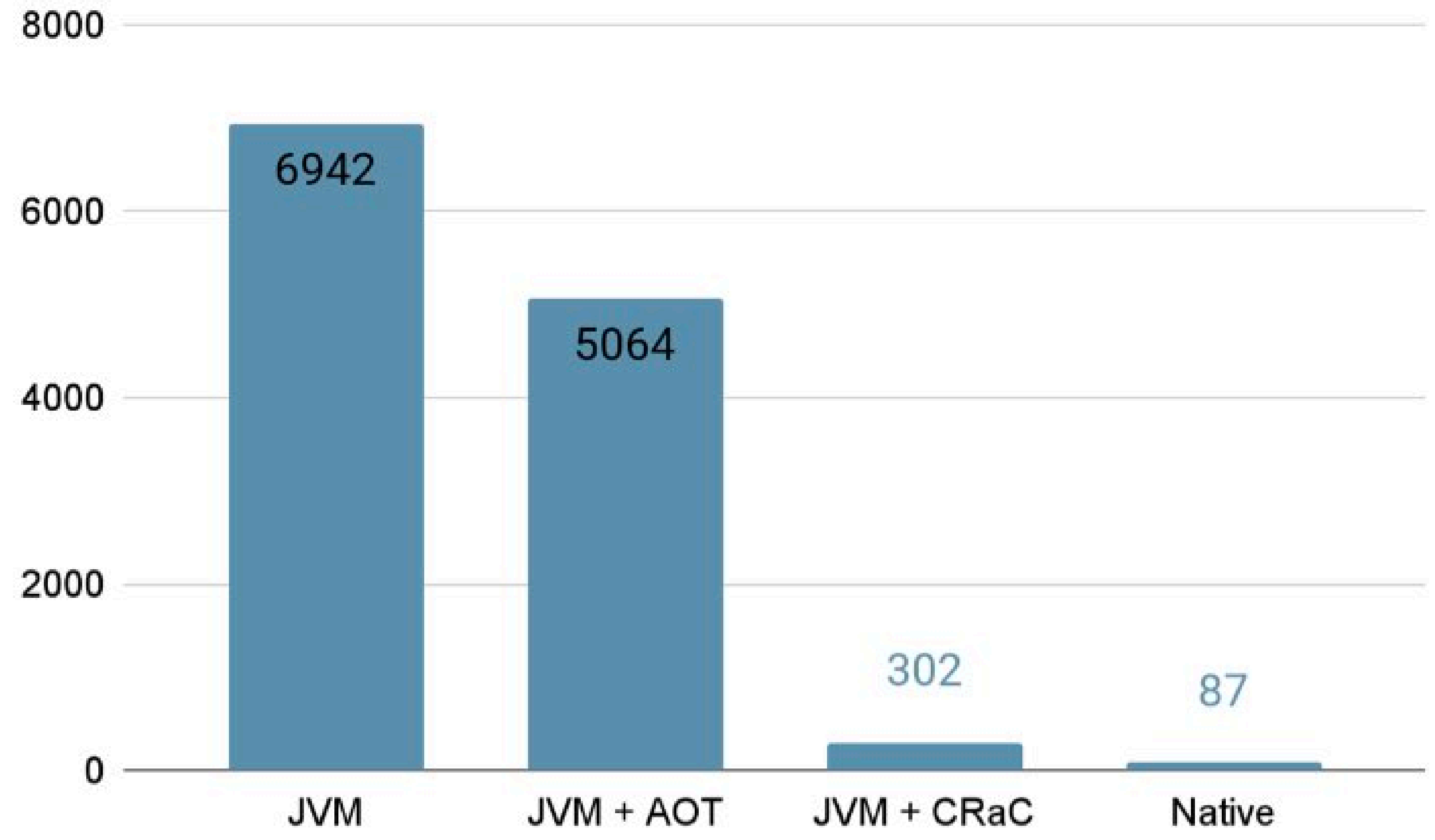
patrick.baumgartner@42talents.com

https://github.com/patbaumgartner/lean-spring-boot-applications-for-the-cloud



Container start to application ready (milliseconds)

Webapp on Azure Container Apps with 1 CPU 2G memory



https://twitter.com/sdeleuze/status/1659129795056115712

Different tradeoffs

	Instant startup with peak performance	Require upfront deployment and checkpoint storage	Compatibility	Run on low resource devices	Compilation time	Compact packaging	Performance
GraalVM native image	Yes	No	Reachability Metadata	Yes	Slow	Yes	EE / CE
CRaC JVM image	Yes	Yes for now[1]	Regular JVM[2]	No	Fast	JVM + checkpoint image	Regular JVM

[1] Build-time checkpoint could lift this requirement

[2] Can require custom checkpoint handling for specific use cases

https://twitter.com/sdeleuze/status/1659461015719890944