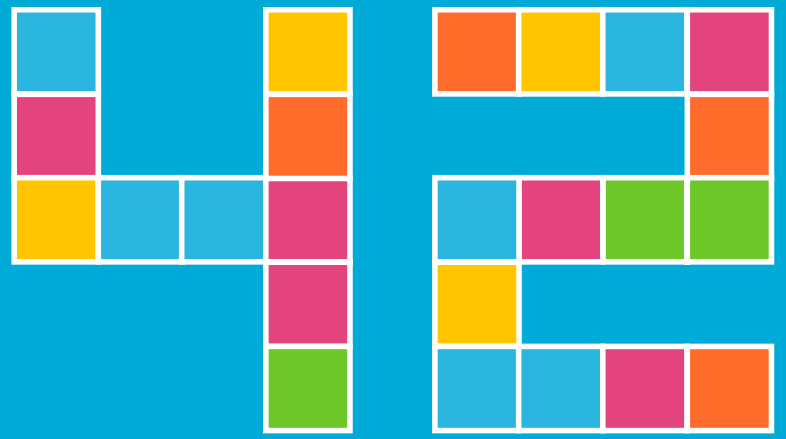
Lean Spring Boot

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?

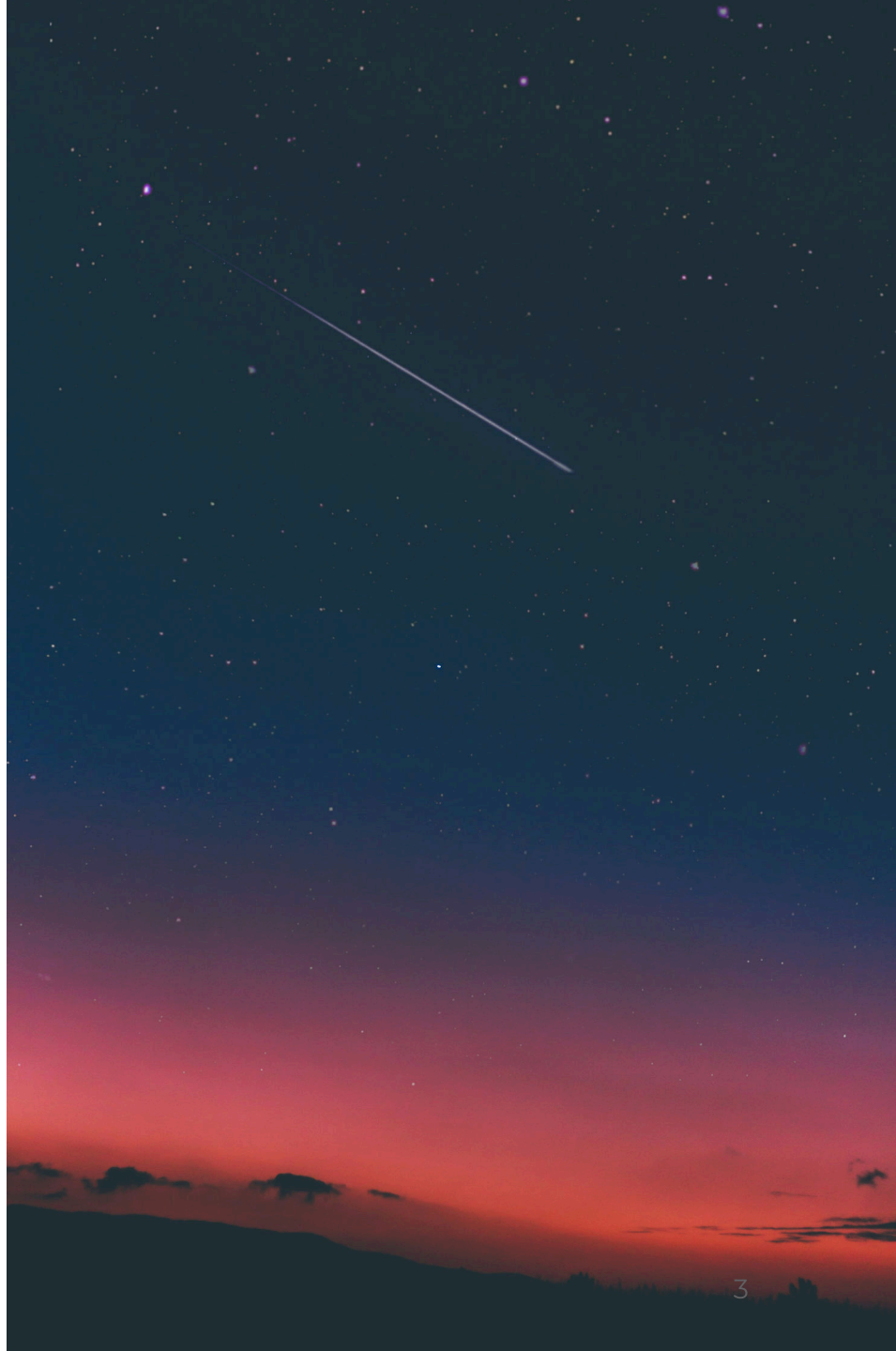
In this talk, we will go into Spring-Boot features like Spring AOT, classpath exclusions, lazy spring beans, actuator, and more. In addition, we're also looking at switching to a different JVM and other tools. All state-of-the-art technology, of course.

Let's make Spring Boot great again!

How To Optimise Your Spring Boot Apps for the Cloud?

Patrick Baumgartner
42talents GmbH, Zürich, Switzerland

@patbaumgartner
patrick.baumgartner@42talents.com



⚠ 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?

Considerations

Resources

- CPU & RAM not linearly scalable
- Image size & network bandwidth

Scalability

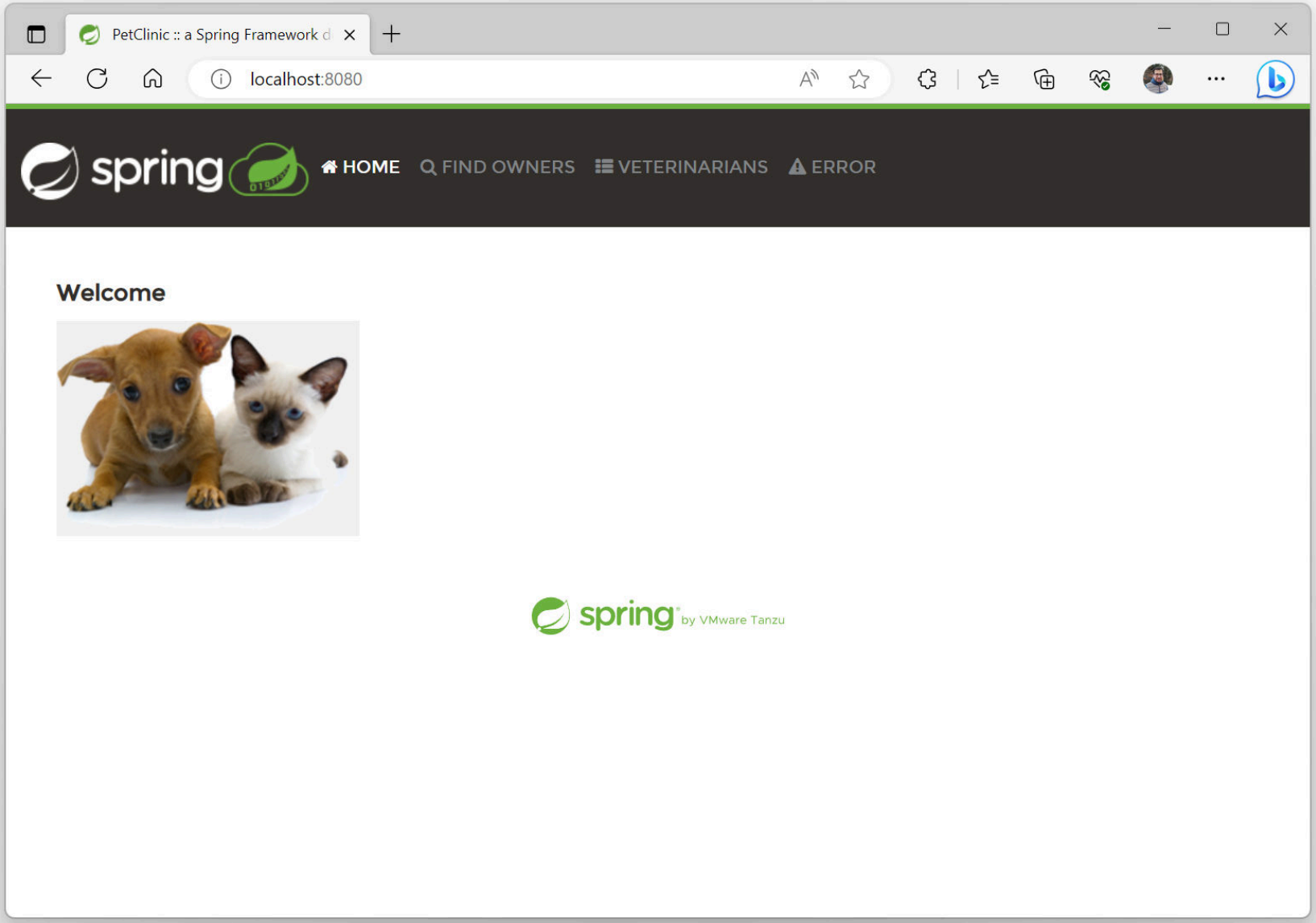
- Fast startup
- Graceful shutdown
- Data throughput
- Latency

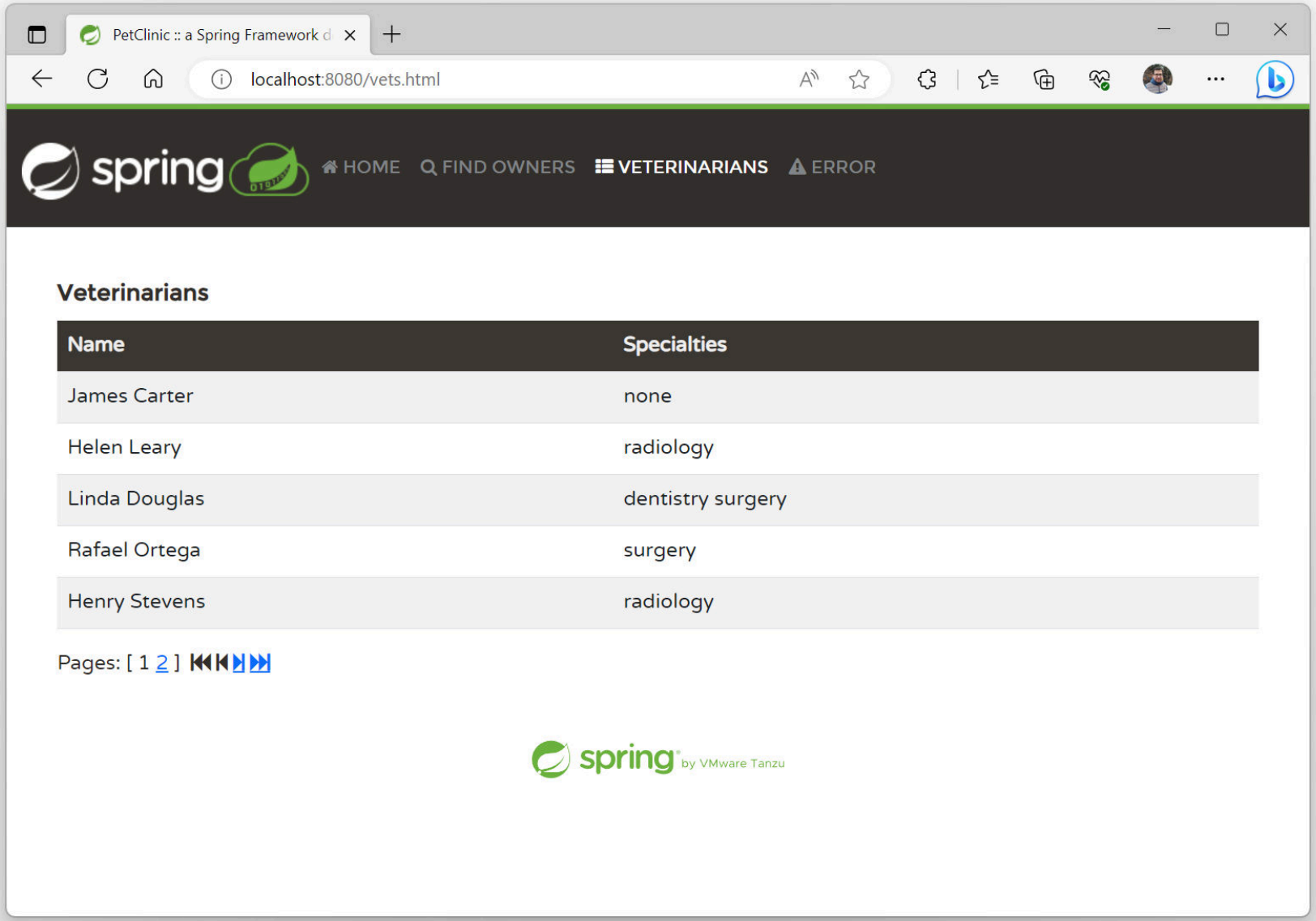
Agenda

Agenda

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

Spring PetClinic


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


PetClinic :: a Spring Framework d
localhost:8080/vets.html
spring
HOME
FIND OWNERS
VETERINARIANS
ERROR
Veterinarians
Name
Specialties
James Carter
none
Helen Leary
radiology
Linda Douglas
dentistry surgery
Rafael Ortega
surgery
Henry Stevens
radiology
Pages: [1 2]
spring by VMware Tanzu

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

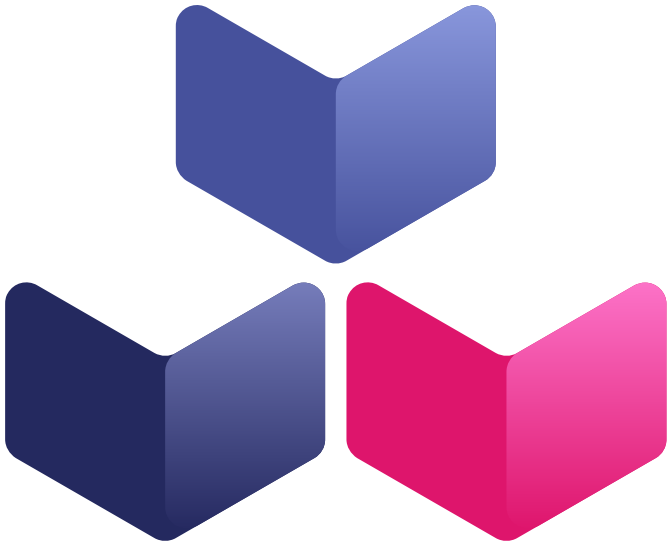
Baseline

Technology Stack

- OCI Container built with Buildpacks
- Java JDK 17 LTS
- Spring Boot 3.2.5
- Testcontainers
- DB migration using SQL scripts

Examination

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



Buildpacks.io

paketo

buildpacks

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

```
Setting Active Processor Count to 4
Calculating JVM memory based on 23848520K 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 -Xmx23216233K -XX:MaxMetaspaceSize=120286K -XX:ReservedCodeCacheSize=240M -Xss1M
(Total Memory: 23848520K, Thread Count: 250, Loaded Class Count: 18823, 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:ActiveProcessorCount=4 -XX:MaxDirectMemorySize=10M -Xmx23216233K -XX:MaxMetaspaceSize=120286K -XX:ReservedCodeCacheSize=240M -Xss1M -XX:
+UnlockDiagnosticVMOptions -XX:NativeMemoryTracking=summary -XX:+PrintNMTStatistics -Dorg.springframework.cloud.bindings.boot.enable=true


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

:: Built with Spring Boot :: 3.2.5


2024-04-20T07:24:25.064Z  INFO 1 --- [           main] o.s.s.petclinic.PetClinicApplication     : Starting PetClinicApplication v3.2.0-SNAPSHOT using Java 17.0.11 with PID 1
                                                         (/workspace/BOOT-INF/classes started by cnb in /workspace)
2024-04-20T07:24:25.068Z  INFO 1 --- [           main] o.s.s.petclinic.PetClinicApplication     : No active profile set, falling back to 1 default profile: "default"
2024-04-20T07:24:25.971Z  INFO 1 --- [           main] .s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data JPA repositories in DEFAULT mode.
2024-04-20T07:24:26.013Z  INFO 1 --- [           main] .s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository scanning in 36 ms. Found 2 JPA repository interfaces.
2024-04-20T07:24:26.601Z  INFO 1 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port 8080 (http)
2024-04-20T07:24:26.610Z  INFO 1 --- [           main] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
2024-04-20T07:24:26.610Z  INFO 1 --- [           main] o.apache.catalina.core.StandardEngine    : Starting Servlet engine: [Apache Tomcat/10.1.20]
2024-04-20T07:24:26.648Z  INFO 1 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
2024-04-20T07:24:26.649Z  INFO 1 --- [           main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 1507 ms
2024-04-20T07:24:26.848Z  INFO 1 --- [           main] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Starting...
2024-04-20T07:24:27.014Z  INFO 1 --- [           main] com.zaxxer.hikari.pool.HikariPool        : HikariPool-1 - Added connection conn0: url=jdbc:h2:mem:6c81973b-1084-481e-8eb7-ccd212a65f4c user=SA
2024-04-20T07:24:27.015Z  INFO 1 --- [           main] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Start completed.
2024-04-20T07:24:27.127Z  INFO 1 --- [           main] o.hibernate.jpa.internal.util.LogHelper  : HHH000204: Processing PersistenceUnitInfo [name: default]
2024-04-20T07:24:27.166Z  INFO 1 --- [           main] org.hibernate.Version                    : HHH000412: Hibernate ORM core version 6.4.4.Final
2024-04-20T07:24:27.190Z  INFO 1 --- [           main] o.h.c.internal.RegionFactoryInitiator    : HHH000026: Second-level cache disabled
2024-04-20T07:24:27.345Z  INFO 1 --- [           main] o.s.o.j.p.SpringPersistenceUnitInfo      : No LoadTimeWeaver setup: ignoring JPA class transformer
2024-04-20T07:24:28.323Z  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)
2024-04-20T07:24:28.325Z  INFO 1 --- [           main] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persistence unit 'default'
2024-04-20T07:24:28.539Z  INFO 1 --- [           main] o.s.d.j.r.query.QueryEnhancerFactory     : Hibernate is in classpath; If applicable, HQL parser will be used.
2024-04-20T07:24:29.562Z  INFO 1 --- [           main] o.s.b.a.e.web.EndpointLinksResolver      : Exposing 13 endpoint(s) beneath base path '/actuator'
2024-04-20T07:24:29.659Z  INFO 1 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port 8080 (http) with context path ''
2024-04-20T07:24:29.676Z  INFO 1 --- [           main] o.s.s.petclinic.PetClinicApplication     : Started PetClinicApplication in 4.946 seconds (process running for 5.253)
```

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

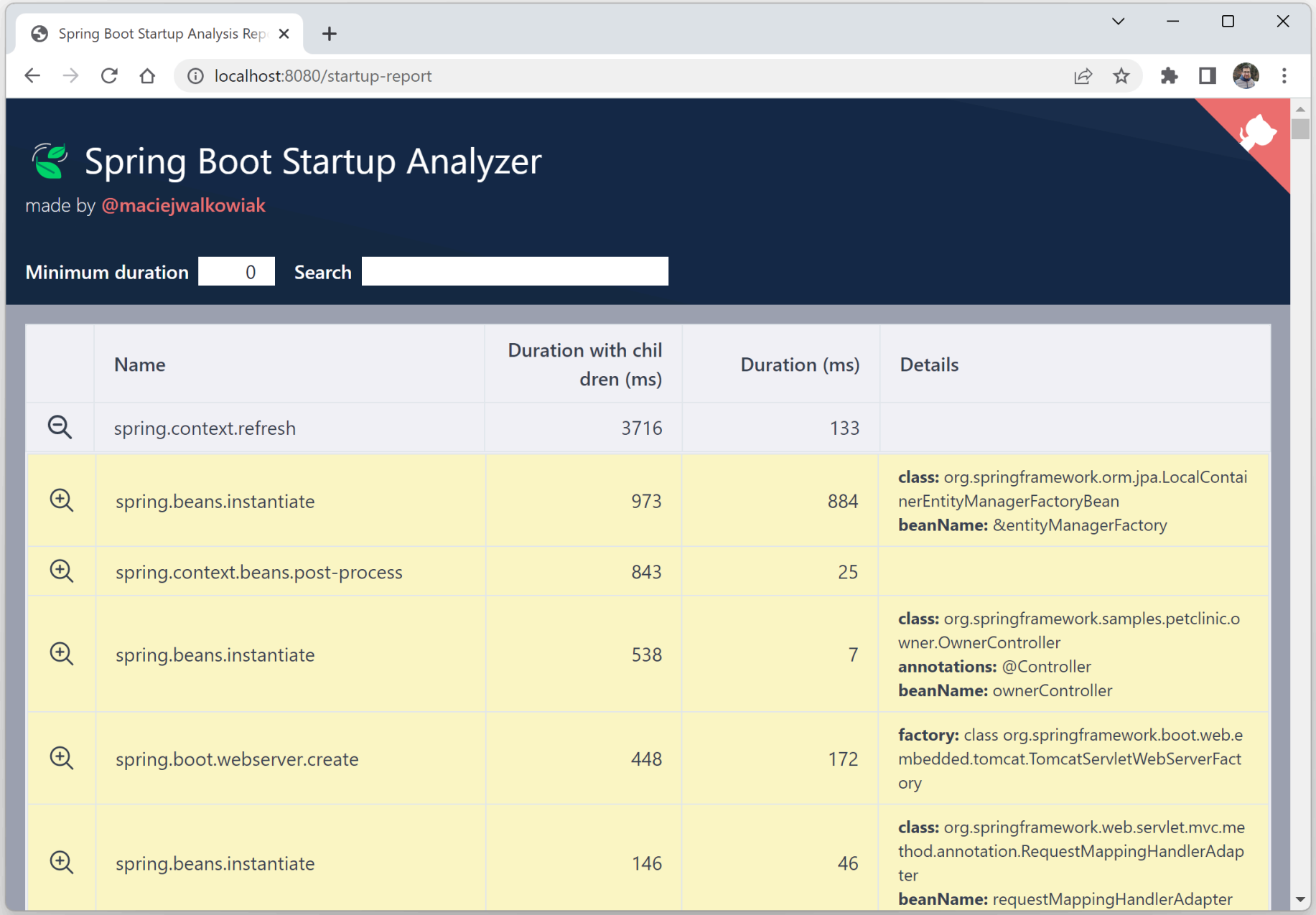
Spring Boot Startup Report

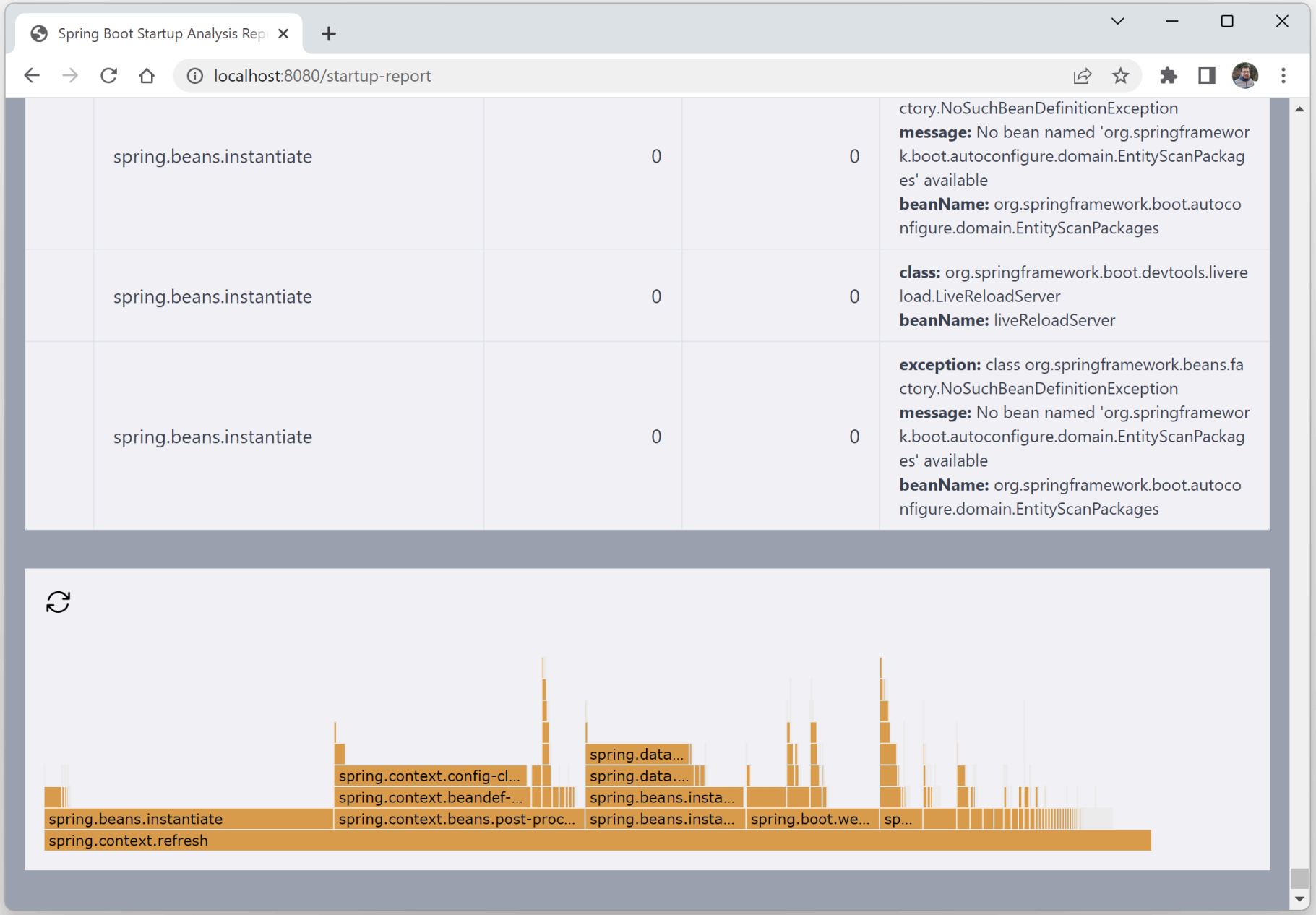
By Maciej Walkowiak

- 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 Rep
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 3716 133
spring.beans.instantiate 973 884 class: org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean beanName: &entityManagerFactory
spring.context.beans.post-process 843 25
spring.beans.instantiate 538 7 class: org.springframework.samples.petclinic.owner.OwnerController annotations: @Controller beanName: ownerController
spring.boot.webserver.create 448 172 factory: class org.springframework.boot.web.embedded.tomcat.TomcatServletWebServerFactory
spring.beans.instantiate 146 46 class: org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter beanName: requestMappingHandlerAdapter


Spring Boot Startup Analysis Rep
localhost:8080/startup-report
spring.beans.instantiate
0
0
ctory.NoSuchBeanDefinitionException
message: No bean named 'org.springframework.boot.autoconfigure.domain.EntityScanPackages' available
beanName: org.springframework.boot.autoconfigure.domain.EntityScanPackages
spring.beans.instantiate
0
0
class: org.springframework.boot.devtools.livereload.LiveReloadServer
beanName: liveReloadServer
spring.beans.instantiate
0
0
exception: class org.springframework.beans.factory.NoSuchBeanDefinitionException
message: No bean named 'org.springframework.boot.autoconfigure.domain.EntityScanPackages' available
beanName: org.springframework.boot.autoconfigure.domain.EntityScanPackages
spring.data...
spring.context.config-cl...
spring.data....
spring.context.beandef-...
spring.beans.insta...
spring.beans.instantiate
spring.context.beans.post-proc...
spring.beans.insta...
spring.boot.we...
sp...
spring.context.refresh

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:3.2.0-SNAPSHOT
```

Build	Image	Startup	Initial RAM	Throughput	RAM	50%	75%	90%	99%	99.90%	99.99%
⏱68s	354MB	4.692s	297MB	1997/s	477MB	5ms	9ms	14ms	34ms	103ms	149ms

No Optimizing - Baseline JDK 21

- Spring PetClinic (JDK 21 adjustments)
- Bellsoft Liberica JDK 21.0.2
- Java Memory Calculator

```
sdk use java 21.0.2-librca

mvn -Djava.version=21 spring-boot:build-image \
    -Dspring-boot.build-image.imageName=spring-petclinic:3.2.0-SNAPSHOT-jdk21

docker run -p 8080:8080 -t spring-petclinic:3.2.0-SNAPSHOT-jdk21
```

Build	Image	Startup	Initial RAM	Throughput	RAM	50%	75%	90%	99%	99.90%	99.99%
⏱68s	354MB	4.692s	297MB	1997/s	477MB	5ms	9ms	14ms	34ms	103ms	149ms
69s	406MB	4.684s	292MB	1989/s	462MB	7ms	12ms	18ms	32ms	99ms	144ms

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:3.2.0-SNAPSHOT-jdk21

docker run -p 8080:8080 -t spring-petclinic:3.2.0-SNAPSHOT-jdk21
```

Build	Image	Startup	Initial RAM	Throughput	RAM	50%	75%	90%	99%	99.90%	99.99%
⏱68s	354MB	4.692s	297MB	1997/s	477MB	5ms	9ms	14ms	34ms	103ms	149ms
69s	402MB	4.674s	298MB	1994/s	484MB	6ms	11ms	16ms	34ms	101ms	162ms

-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:3.2.0-SNAPSHOT
```

Build	Image	Startup	Initial RAM	Throughput	RAM	50%	75%	90%	99%	99.90%	99.99%
⏱68s	354MB	4.692s	297MB	1997/s	477MB	5ms	9ms	14ms	34ms	103ms	149ms
-	-	4.156s	284MB	1994/s	459MB	7ms	12ms	18ms	38ms	101ms	145ms

-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:3.2.0-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%	99.90%	99.99%
⏱ 68s	354MB	4.692s	297MB	1997/s	477MB	5ms	9ms	14ms	34ms	103ms	149ms
-	-	4.1s	214MB	1539/s	327MB	7826ms	10760ms	12980ms	16026ms	17820ms	18962ms

-XX:+UseZGC

-XX:+UseZGC

The Z Garbage Collector (ZGC) is a scalable, low-latency garbage collector. ZGC performs all expensive work concurrently without stopping application threads for more than 10ms, making it suitable for applications that require low latency and/or use a very large heap (multi-terabyte).

```
docker run -p 8080:8080 -e "JAVA_TOOL_OPTIONS=-XX:+UseZGC" \
  -t spring-petclinic:3.2.0-SNAPSHOT
```

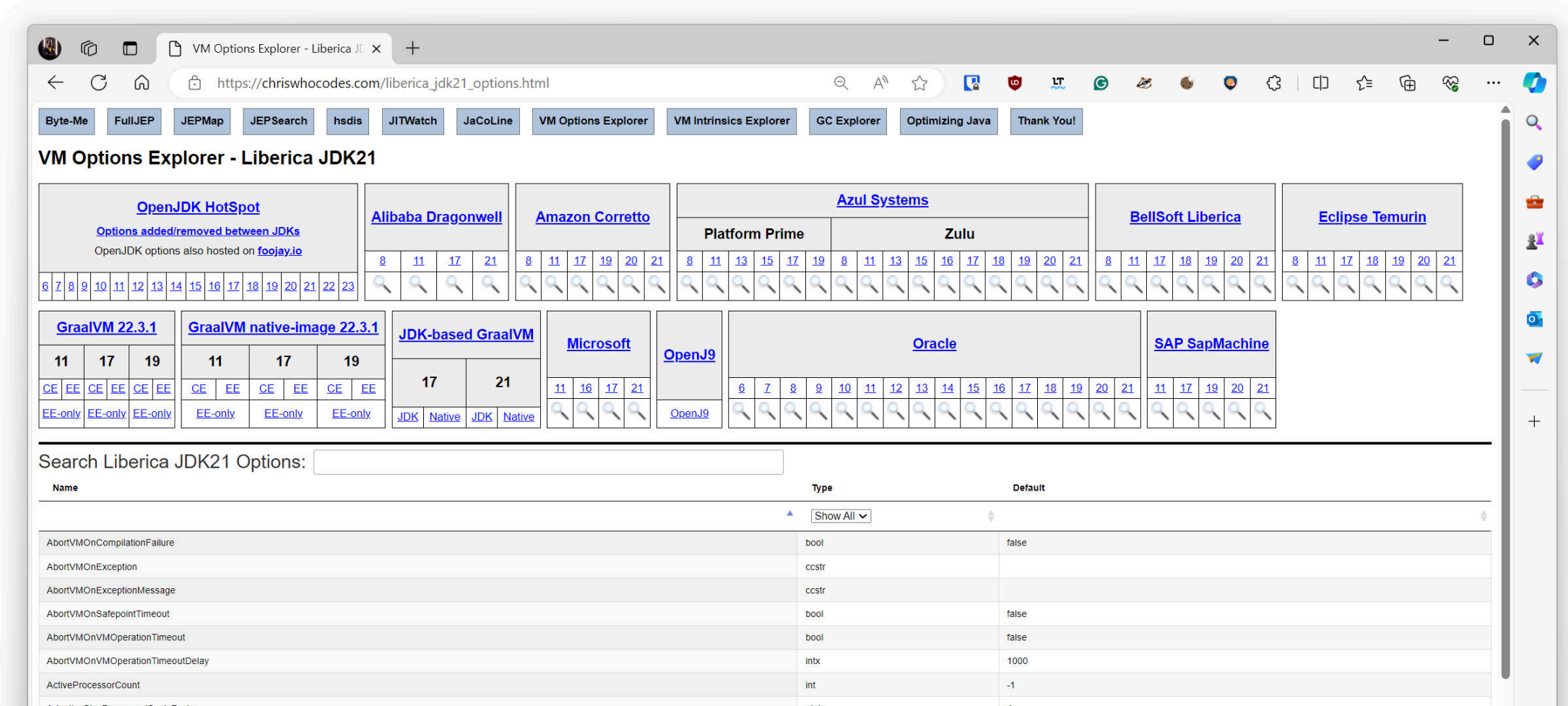
See also: https://wiki.openjdk.org/display/zgc/Main

Build	Image	Startup	Initial RAM	Throughput	RAM	50%	75%	90%	99%	99.90%	99.99%
⏱68s	354MB	4.692s	297MB	1997/s	477MB	5ms	9ms	14ms	34ms	103ms	149ms
-	-	4.721s	989MB	1996/s	1555MB	14ms	27ms	47ms	128ms	223ms	344ms

VM Options Explorer

VM Options Explorer

https://chriswhocodes.com



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:3.2.0-SNAPSHOT
```

Build	Image	Startup	Initial RAM	Throughput	RAM	50%	75%	90%	99%	99.90%	99.99%
⏱68s	354MB	4.692s	297MB	1997/s	477MB	5ms	9ms	14ms	34ms	103ms	149ms
-	-	3.451s	262MB	1991/s	470MB	5ms	10ms	14ms	31ms	78ms	108ms

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 until 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:3.2.0-SNAPSHOT
```

Build	Image	Startup	Initial RAM	Throughput	RAM	50%	75%	90%	99%	99.90%	99.99%
⏱68s	354MB	4.692s	297MB	1997/s	477MB	5ms	9ms	14ms	34ms	103ms	149ms
-	-	4.697s	290MB	1991/s	466MB	6ms	11ms	15ms	31ms	95ms	154ms

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: 447
  - Spring Pet Clinic without actuators: 275 🔥

```
sdk use java 17.0.11-librca

mvn spring-boot:build-image

docker run -p 8080:8080 -t spring-petclinic-no-actuator:3.2.0-SNAPSHOT
```

Build	Image	Startup	Initial RAM	Throughput	RAM	50%	75%	90%	99%	99.90%	99.99%
⏱️68s	354MB	4.692s	297MB	1997/s	477MB	5ms	9ms	14ms	34ms	103ms	149ms
65s	351MB	3.791s	262MB	1994/s	445MB	5ms	9ms	13ms	32ms	100ms	147ms

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:3.2.0-SNAPSHOT
```

I 

Spring Boot  & Buildpacks

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.

```
<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)
        org.postgresql:postgresql:42.6.2:runtime (1 MB)
        com.github.ben-manes.caffeine:caffeine:3.1.8:compile (868 KB)
        ...
USED TRANSITIVE DEPENDENCIES [89]:
        org.testcontainers:testcontainers:1.19.7:test (16 MB)
        org.hibernate.orm:hibernate-core:6.4.4.Final:compile (11 MB)
        net.bytebuddy:byte-buddy:1.14.13:runtime (4 MB)
        org.apache.tomcat.embed:tomcat-embed-core:10.1.20:compile (3 MB)
        com.github.docker-java:docker-java-transport-zerodep:3.3.6:test (2 MB)
        org.aspectj:aspectjweaver:1.9.22:compile (2 MB)
        org.springframework.boot:spring-boot-autoconfigure:3.2.5:compile (1 MB)
        org.springframework:spring-web:6.1.6:compile (1 MB)
        org.springframework:spring-core:6.1.6: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:3.2.5:test (198 KB)
        org.springframework.boot:spring-boot-docker-compose:3.2.5:test (177 KB)
        org.springframework.boot:spring-boot-starter-web:3.2.5:compile (4 KB)
        org.springframework.boot:spring-boot-starter-test:3.2.5:test (4 KB)
        org.springframework.boot:spring-boot-starter-thymeleaf:3.2.5:compile (4 KB)
        org.springframework.boot:spring-boot-starter:3.2.5:compile (4 KB)
        ...
POTENTIALLY UNUSED TRANSITIVE DEPENDENCIES [11]:
        org.attoparser:attoparser:2.0.7.RELEASE:compile (240 KB)
        org.thymeleaf:thymeleaf-spring6:3.1.2.RELEASE:compile (184 KB)
        org.unbescape:unbescape:1.1.6.RELEASE:compile (169 KB)
        org.awaitility:awaitility:4.2.1:test (94 KB)
        com.google.errorprone:error_prone_annotations:2.21.1:compile (16 KB)
        org.slf4j:jul-to-slf4j:2.0.13:compile (6 KB)
        org.springframework.boot:spring-boot-starter-tomcat:3.2.5:compile (4 KB)
        ...
POTENTIALLY UNUSED INHERITED DIRECT DEPENDENCIES [0]:
POTENTIALLY UNUSED INHERITED TRANSITIVE DEPENDENCIES [0]:
...
[INFO] Analysis done in 0min 10s
```

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:3.2.0-SNAPSHOT
```

Build	Image	Startup	Initial RAM	Throughput	RAM	50%	75%	90%	99%	99.90%	99.99%
⏱68s	354MB	4.692s	297MB	1997/s	477MB	5ms	9ms	14ms	34ms	103ms	149ms
74s	348MB	3.653s	262MB	1994/s	434MB	5ms	10ms	15ms	32ms	95ms	133ms

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.

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

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

We create a new container image with the AOT-processed application.

```
sdk use java 17.0.11-librca

mvn spring-boot:build-image

docker run -e spring.aot.enabled=true -p 8080:8080 -t spring-petclinic-aot:3.2.0-SNAPSHOT
```

Build	Image	Startup	Initial RAM	Throughput	RAM	50%	75%	90%	99%	99.90%	99.99%
⏱68s	354MB	4.692s	297MB	1997/s	477MB	5ms	9ms	14ms	34ms	103ms	149ms
72s	355MB	4.798s	297MB	1992/s	469MB	6ms	10ms	15ms	36ms	101ms	146ms

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 spring-boot:build-image

dock
```

Build	Image	Startup	Initial RAM	Throughput	RAM	50%	75%	90%	99%	99.90%	99.99%
⏱68s	354MB	4.692s	297MB	1997/s	477MB	5ms	9ms	14ms	34ms	103ms	149ms
83s	287MB	4.823s	322MB	1997/s	481MB	7ms	12ms	18ms	38ms	90ms	131ms

JLink (3)

```
<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%	99.90%	99.99%
⏱ 68s	354MB	4.692s	297MB	1997/s	477MB	5ms	9ms	14ms	34ms	103ms	149ms
83s	274MB	4.899s	309MB	1992/s	479MB	7ms	12ms	17ms	36ms	106ms	151ms

I 

Spring Boot 🍃 & Buildpacks

Eclipse OpenJ9

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

```
<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 buildpacks -->
            <buildpack>gcr.io/paketo-buildpacks/java:latest</buildpack>
        </buildpacks>
    </image>
  </configuration>
</plugin>
```

```
sdk use java 17.0.10-sem

mvn

dock
```

Build	Image	Startup	Initial RAM	Throughput	RAM	50%	75%	90%	99%	99.90%	99.99%
⏱68s	354MB	4.692s	297MB	1997/s	477MB	5ms	9ms	14ms	34ms	103ms	149ms
76s	351MB	8.071s	176MB	1991/s	361MB	8ms	15ms	23ms	45ms	86ms	149ms

Eclipse OpenJ9 Optimized

Eclipse OpenJ9 Optimized

When `-Xtune:virtualised` is used in conjunction with the `-Xshareclasses` option, the JIT compiler is more aggressive in its use of AOT-compiled code than when only `-Xshareclasses` is set. This provides additional CPU savings during application startup and ramp-up, but at the cost of a small additional loss in throughput.

```
docker run -p 8080:8080 -e "JAVA_TOOL_OPTIONS=-XX:+IgnoreUnrecognizedVMOptions -XX:+UseContainerSupport
-XX:+IdleTuningCompactOnIdle -XX:+IdleTuningGcOnIdle -Xscmx50M -Xshareclasses -Xtune:virtualized" \
-t spring-petclinic-custom-jvm:3.2.0-SNAPSHOT
```

Build	Image	Startup	Initial RAM	Throughput	RAM	50%	75%	90%	99%	99.90%	99.99%
⏱68s	354MB	4.692s	297MB	1997/s	477MB	5ms	9ms	14ms	34ms	103ms	149ms
(q) -	-	6.215s	175MB	1284/s	301MB	12172ms	16708ms	19554ms	22838ms	24788ms	25740ms
(qv) -	-	9.84s	186MB	1997/s	344MB	12ms	19ms	31ms	85ms	180ms	274ms

GraalVM

GraalVM

```
<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 buildpacks -->
            <buildpack>gcr.io/paketo-buildpacks/java:latest</buildpack>
        </buildpacks>
    </image>
  </configuration>
</plugin>
```

```
sdk use java 21.0.2-graalce
```

```
mvn
```

```
dock
```

Build	Image	Startup	Initial RAM	Throughput	RAM	50%	75%	90%	99%	99.90%	99.99%
⏱68s	354MB	4.692s	297MB	1997/s	477MB	5ms	9ms	14ms	34ms	103ms	149ms
95s	758MB	4.922s	265MB	1992/s	450MB	6ms	11ms	16ms	32ms	76ms	110ms

GraalVM Oracle

```
<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 buildpacks -->
            <buildpack>gcr.io/paketo-buildpacks/java:latest</buildpack>
        </buildpacks>
    </image>
  </configuration>
</plugin>
```

```
sdk use java 21.0.2-graal

mvn

dock
```

Build	Image	Startup	Initial RAM	Throughput	RAM	50%	75%	90%	99%	99.90%	99.99%
⏱68s	354MB	4.692s	297MB	1997/s	477MB	5ms	9ms	14ms	34ms	103ms	149ms
81s	543MB	4.872s	301MB	1995/s	477MB	6ms	10ms	15ms	37ms	106ms	164ms

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.

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

Build	Image	Startup	Initial RAM	Throughput	RAM	50%	75%	90%	99%	99.90%	99.99%
⏱68s	354MB	4.692s	297MB	1997/s	477MB	5ms	9ms	14ms	34ms	103ms	149ms
(m) 62s	177MB	5.252s	265MB	1994/s	407MB	5ms	10ms	15ms	36ms	106ms	156ms
(g) 62s	191MB	4.763s	306MB	1994/s	490MB	5ms	9ms	13ms	32ms	98ms	140ms

Buildpack Builder Tiny

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

```
sdk use java 17.0.11-librca

mvn spring-boot:build-image

docker run -p 8080:8080 -t spring-petclinic-bellsoft-builder:3.2.0-SNAPSHOT
```

Build	Image	Startup	Initial RAM	Throughput	RAM	50%	75%	90%	99%	99.90%	99.99%
⏱ 68s	354MB	4.692s	297MB	1997/s	477MB	5ms	9ms	14ms	34ms	103ms	149ms
68s	255MB	4.854s	295MB	1995/s	473MB	6ms	11ms	16ms	41ms	109ms	163ms

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:3.2.0-SNAPSHOT
```

Build	Image	Startup	Initial RAM	Throughput	RAM	50%	75%	90%	99%	99.90%	99.99%
⏱ 68s	354MB	4.692s	297MB	1997/s	477MB	5ms	9ms	14ms	34ms	103ms	149ms
(ce) 370s	223MB	0.497s	249MB	1995/s	468MB	41ms	73ms	129ms	318ms	520ms	688ms
(o) 569s	248MB	0.358s	238MB	1992/s	398MB	11ms	18ms	31ms	89ms	170ms	254ms

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+7_linux-x64/
export PATH=$JAVA_HOME/bin:$PATH
```

```
mvn clean verify
```

```
java -XX:CRaCCheckpointTo=crac-files -jar target/spring-petclinic-crac-3.2.0-SNAPSHOT.jar
```

```
jcmd target/spring-petclinic-crac-3.2.0-SNAPSHOT.jar JDK.checkpoint
```

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

CRaC - OpenJDK (3)

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

- Spring Boot 3.2 support finalised
  - Spring Framework 6.1.0
- Does not work on Windows or MacOS, but does work on Linux
  - Does not work in Docker containers over WSL
  - But works in VM with Ubuntu 22.04 LTS

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

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 21.0.2-librca

mvn spring-boot:build-image

docker run -e spring.threads.virtual.enabled=true \
  -p 8080:8080 -t spring-petclinic-virtual-threads:3.2.0-SNAPSHOT
```

Build	Image	Startup	Initial RAM	Throughput	RAM	50%	75%	90%	99%	99.90%	99.99%
⏱68s	354MB	4.692s	297MB	1997/s	477MB	5ms	9ms	14ms	34ms	103ms	149ms
69s	406MB	4.753s	289MB	1995/s	454MB	4ms	7ms	10ms	27ms	86ms	122ms

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)*
- App CDS (Class Data Sharing)*
- Using an Obfuscator such as ProGuard*
- Importing `AutoConfiguration` classes individually
- Use of functional bean definitions
- 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)
- 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.aot.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-bellsoft-builder:3.2.0-SNAPSHOT
```

Build	Image	Startup	Initial RAM	Throughput	RAM	50%	75%	90%	99%	99.90%	99.99%
⏱️68s	354MB	4.692s	297MB	1997/s	477MB	5ms	9ms	14ms	34ms	103ms	149ms
69s	142MB	2.278s	193MB	1994/s	354MB	5ms	9ms	14ms	45ms	98ms	151ms

Java 21 & Base Builder Tiny

```
sdk use java 21.0.2-librca

mvn spring-boot:build-image

docker run -p 8080:8080 \
  -e spring.threads.virtual.enabled=true \
  -e spring.aot.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:3.2.0-SNAPSHOT
```

Build	Image	Startup	Initial RAM	Throughput	RAM	50%	75%	90%	99%	99.90%	99.99%
⏱68s	354MB	4.692s	297MB	1997/s	477MB	5ms	9ms	14ms	34ms	103ms	149ms
83s	178MB	2.065s	244MB	1995/s	414MB	4ms	7ms	10ms	28ms	92ms	126ms

Did I miss something? 🧐

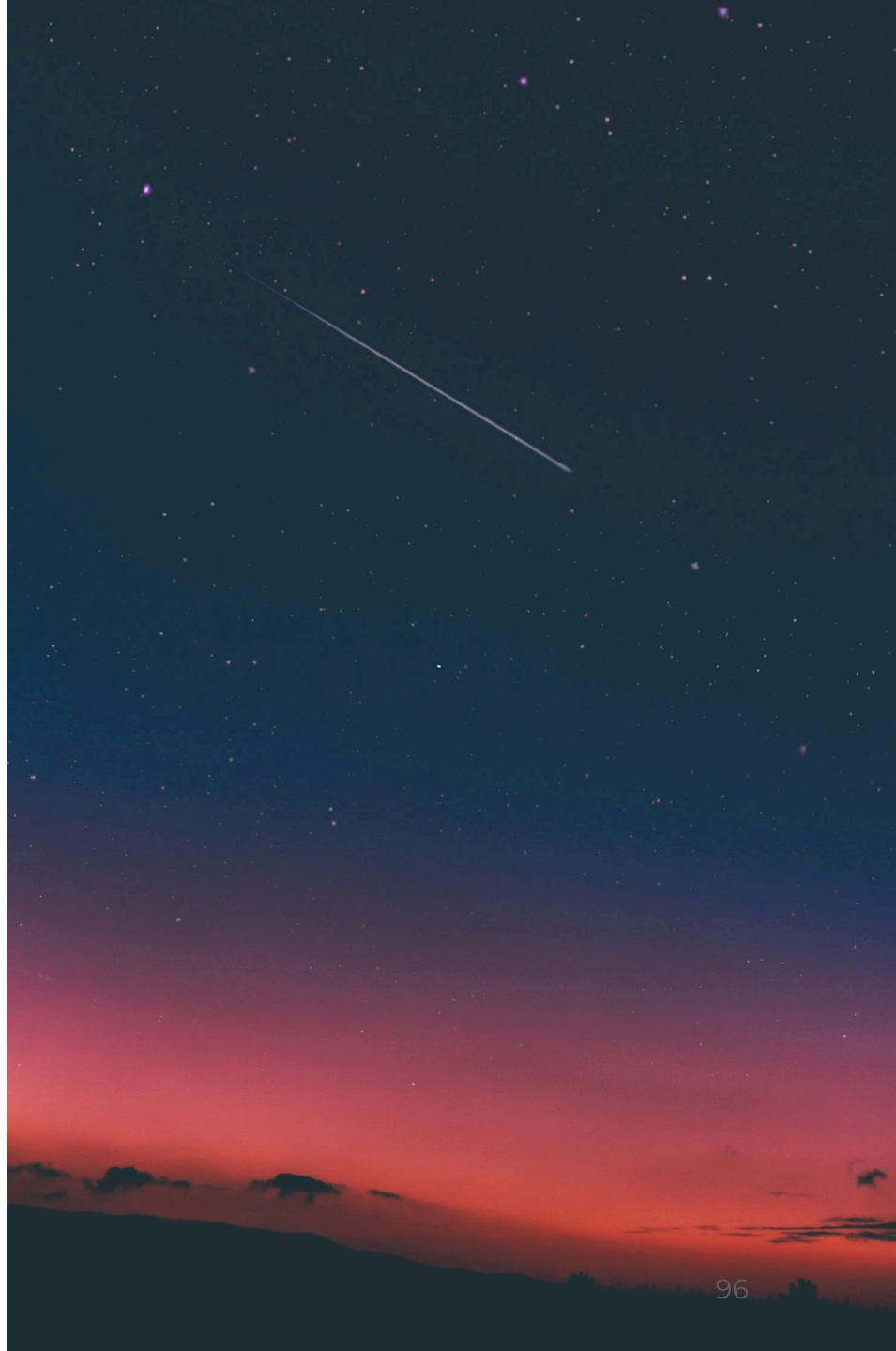
Let me/us know! 🙋

… or not! 🙊

How To Optimise Your Spring Boot Apps for the Cloud?

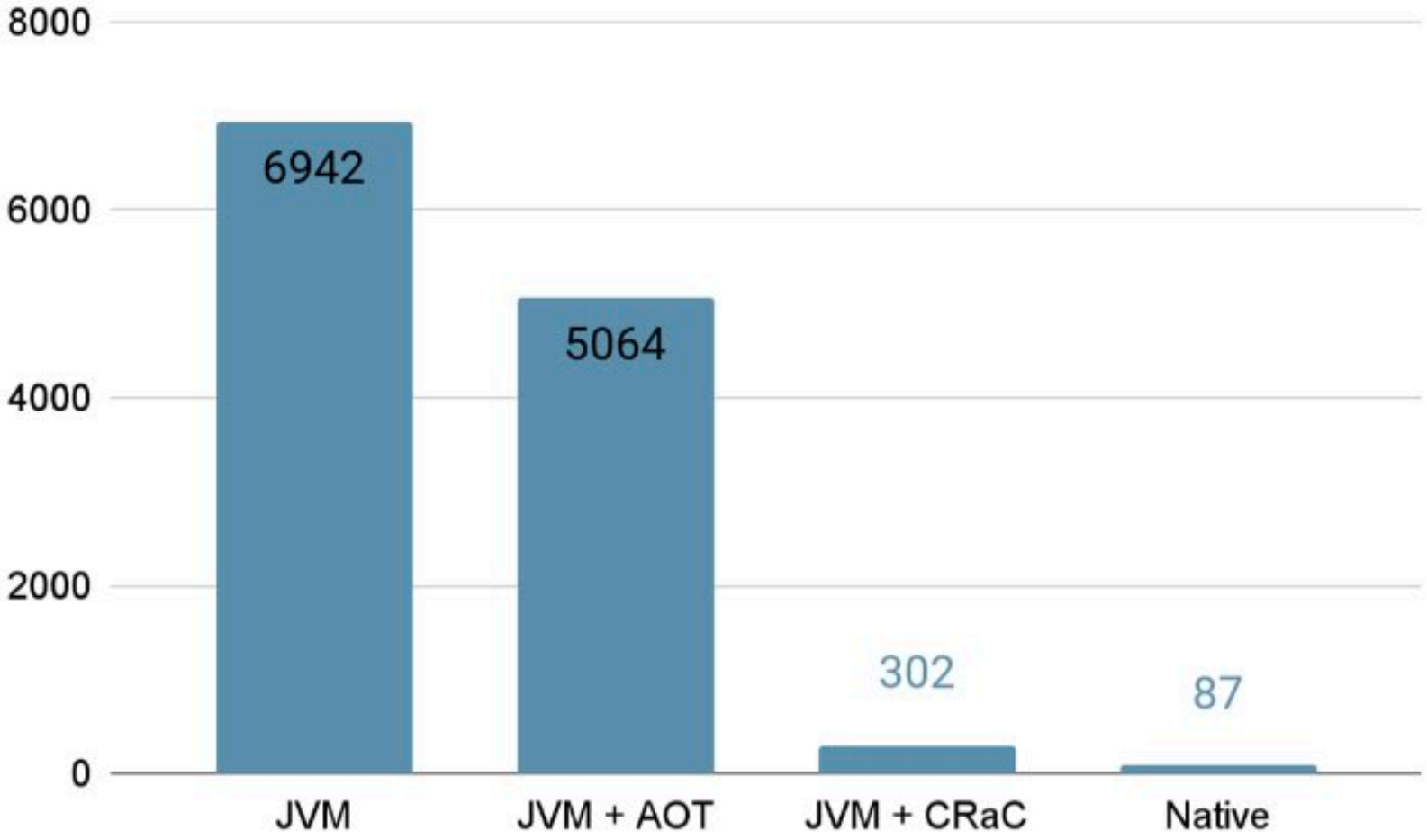
Patrick Baumgartner
42talents GmbH, Zürich, Switzerland

@patbaumgartner
patrick.baumgartner@42talents.com



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