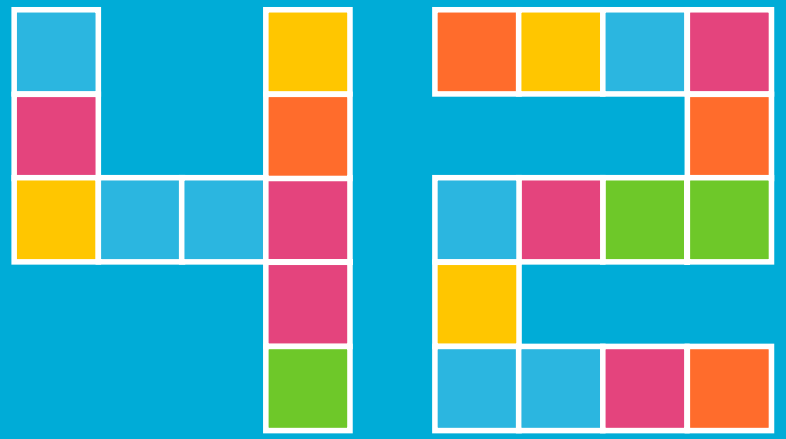
Lean Spring Boot

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.

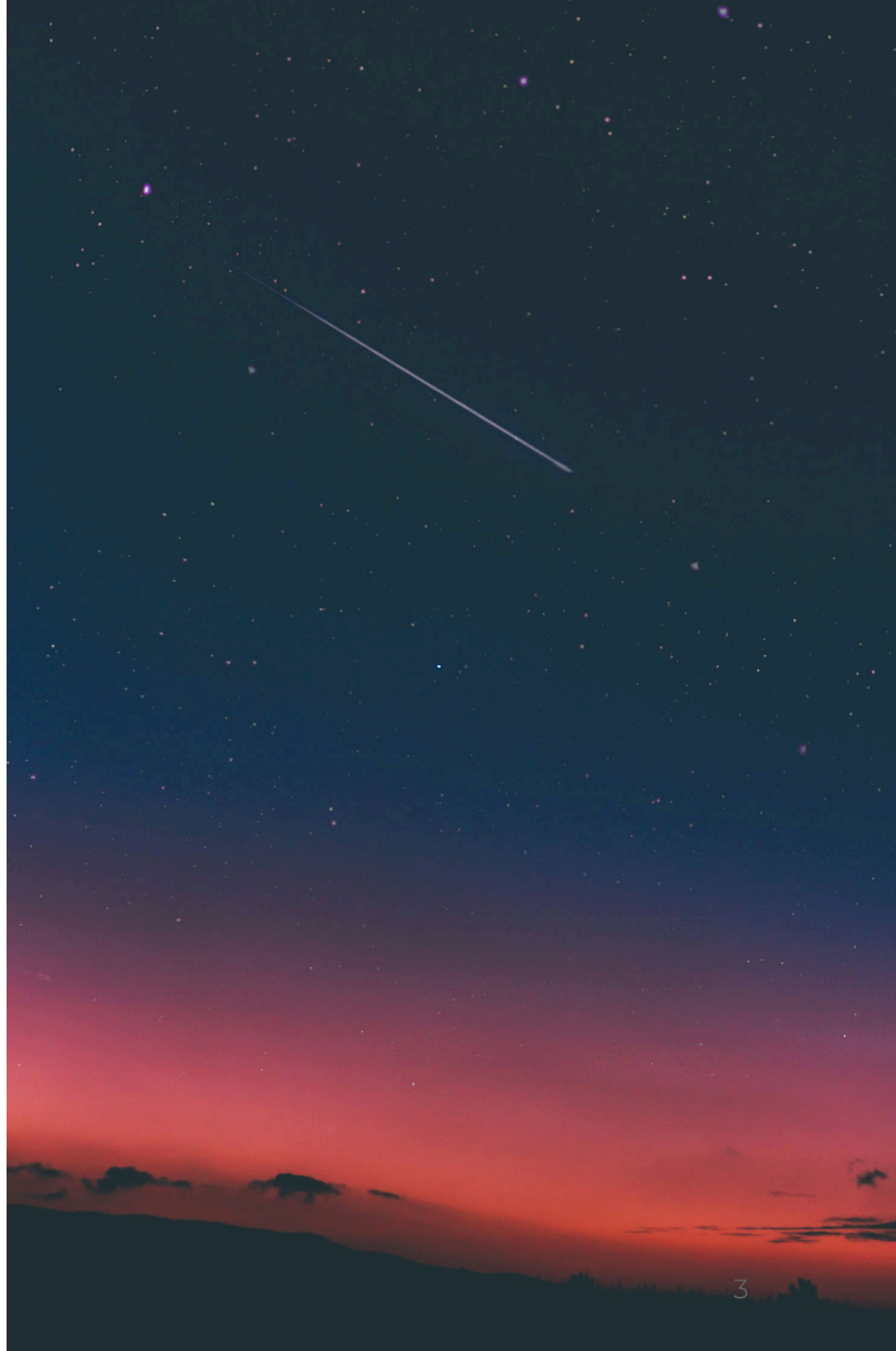
Let's make Spring Boot great again!

Lean Spring Boot

Applications for The Cloud

Patrick Baumgartner
42talents GmbH, Zürich, Switzerland

@patbaumgartner
patrick.baumgartner@42talents.com



⚠ 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**

Co-Organizer of **Voxxed Days Zurich**, **JUG Switzerland**, ...

@patbaumgartner

What's the challenge?

Why does this matter?

I ❤️ Java 😉 & Spring Boot 🍃

Requirements

When Deploying to a Cloud

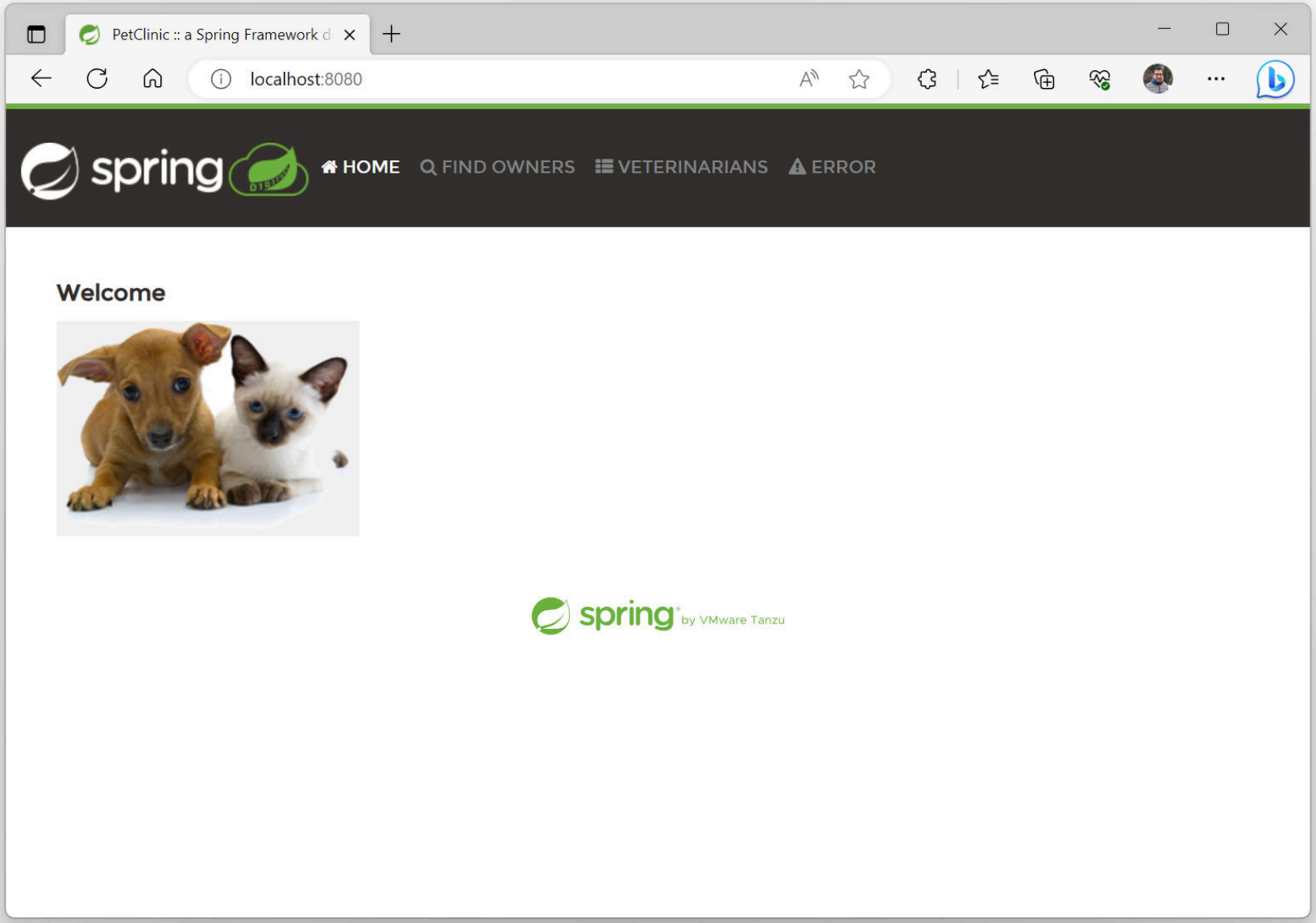
- How many vCPUs will my application need?
- How much RAM do I need?
- How much storage do I need?
- What technology stack should I use?
- What type of application do we build?

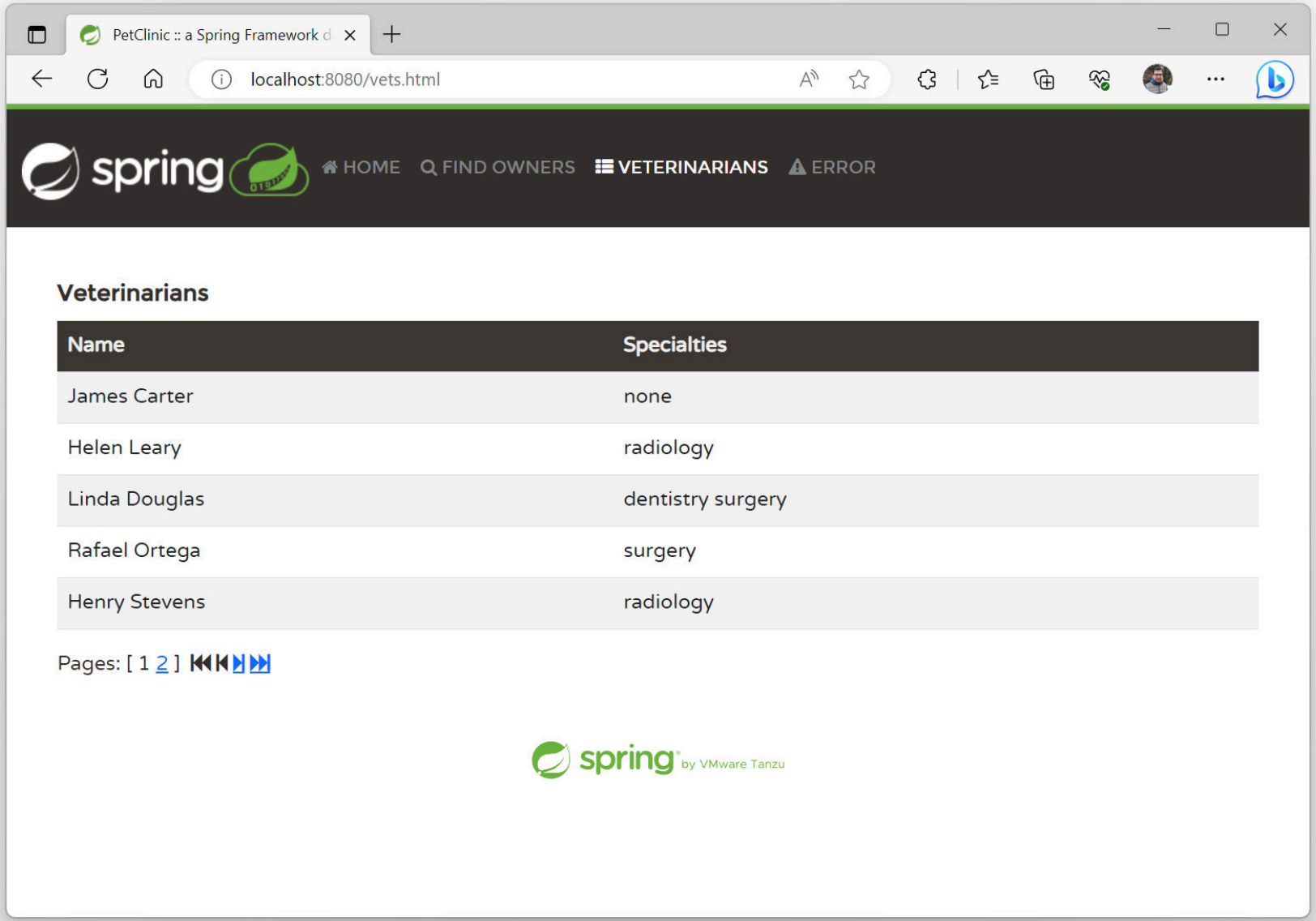
Agenda

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
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

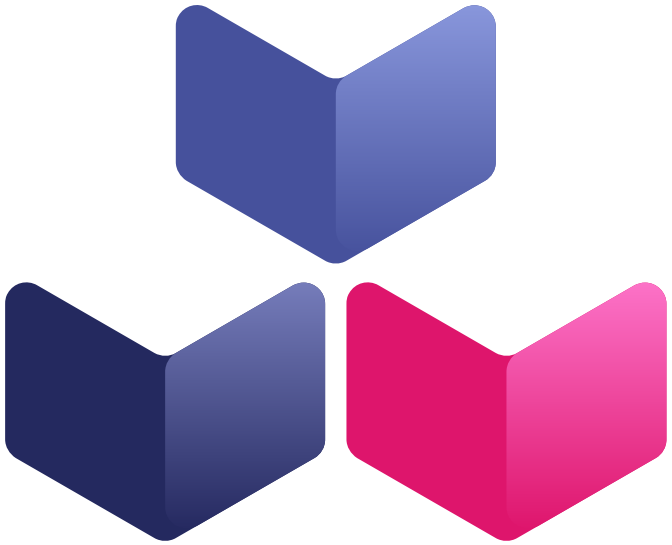
Baseline

Technology Stack

- OCI Container built with Buildpacks
- Java JDK 17 LTS
- Spring Boot 3.3.4
- 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

```
Calculating JVM memory based on 22787416K 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 -Xmx22154489K -XX:MaxMetaspaceSize=120926K -XX:ReservedCodeCacheSize=240M -Xss1M
(Total Memory: 22787416K, Thread Count: 250, Loaded Class Count: 18936, Headroom: 0%)
Enabling Java Native Memory Tracking
Adding 146 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 -Xmx22154489K -XX:MaxMetaspaceSize=120926K  -XX:ReservedCodeCacheSize=240M -Xss1M
  -XX:+UnlockDiagnosticVMOptions -XX:NativeMemoryTracking=summary -XX:+PrintNMTStatistics -Dorg.springframework.cloud.bindings.boot.enable=true


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

:: Built with Spring Boot :: 3.3.4


2024-10-01T17:47:04.691Z  INFO 1 --- [           main] o.s.s.petclinic.PetClinicApplication     : Starting PetClinicApplication v3.3.0-SNAPSHOT using Java 17.0.12
                                                         with PID 1 (/workspace/BOOT-INF/classes started by cnb in /workspace)
2024-10-01T17:47:04.694Z  INFO 1 --- [           main] o.s.s.petclinic.PetClinicApplication     : No active profile set, falling back to 1 default profile: "default"
2024-10-01T17:47:05.421Z  INFO 1 --- [           main] .s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data JPA repositories in DEFAULT mode.
2024-10-01T17:47:05.461Z  INFO 1 --- [           main] .s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository scanning in 36 ms. Found 2 JPA repository interfaces.
2024-10-01T17:47:06.877Z  INFO 1 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port 8080 (http)
2024-10-01T17:47:06.886Z  INFO 1 --- [           main] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
2024-10-01T17:47:06.887Z  INFO 1 --- [           main] o.apache.catalina.core.StandardEngine    : Starting Servlet engine: [Apache Tomcat/10.1.30]
2024-10-01T17:47:06.930Z  INFO 1 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
2024-10-01T17:47:06.931Z  INFO 1 --- [           main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 2195 ms
2024-10-01T17:47:07.205Z  INFO 1 --- [           main] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Starting...
2024-10-01T17:47:07.381Z  INFO 1 --- [           main] com.zaxxer.hikari.pool.HikariPool        : HikariPool-1 - Added connection conn0: url=jdbc:h2:mem:dc0e8fe7-6898-4bcc-a3b6-8a801504d412 user=SA
2024-10-01T17:47:07.383Z  INFO 1 --- [           main] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Start completed.
2024-10-01T17:47:07.541Z  INFO 1 --- [           main] o.hibernate.jpa.internal.util.LogHelper  : HHH000204: Processing PersistenceUnitInfo [name: default]
2024-10-01T17:47:07.598Z  INFO 1 --- [           main] org.hibernate.Version                    : HHH000412: Hibernate ORM core version 6.5.3.Final
2024-10-01T17:47:07.621Z  INFO 1 --- [           main] o.h.c.internal.RegionFactoryInitiator    : HHH000026: Second-level cache disabled
2024-10-01T17:47:07.868Z  INFO 1 --- [           main] o.s.o.j.p.SpringPersistenceUnitInfo      : No LoadTimeWeaver setup: ignoring JPA class transformer
2024-10-01T17:47:09.045Z  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)
2024-10-01T17:47:09.048Z  INFO 1 --- [           main] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persistence unit 'default'
2024-10-01T17:47:09.289Z  INFO 1 --- [           main] o.s.d.j.r.query.QueryEnhancerFactory     : Hibernate is in classpath; If applicable, HQL parser will be used.
2024-10-01T17:47:10.689Z  INFO 1 --- [           main] o.s.b.a.e.web.EndpointLinksResolver      : Exposing 14 endpoints beneath base path '/actuator'
2024-10-01T17:47:10.781Z  INFO 1 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port 8080 (http) with context path '/'
2024-10-01T17:47:10.795Z  INFO 1 --- [           main] o.s.s.petclinic.PetClinicApplication     : Started PetClinicApplication in 6.403 seconds (process running for 6.786)
```

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/

Friends don't let friends write Dockerfiles!

@starbuxman

Startup Reporting

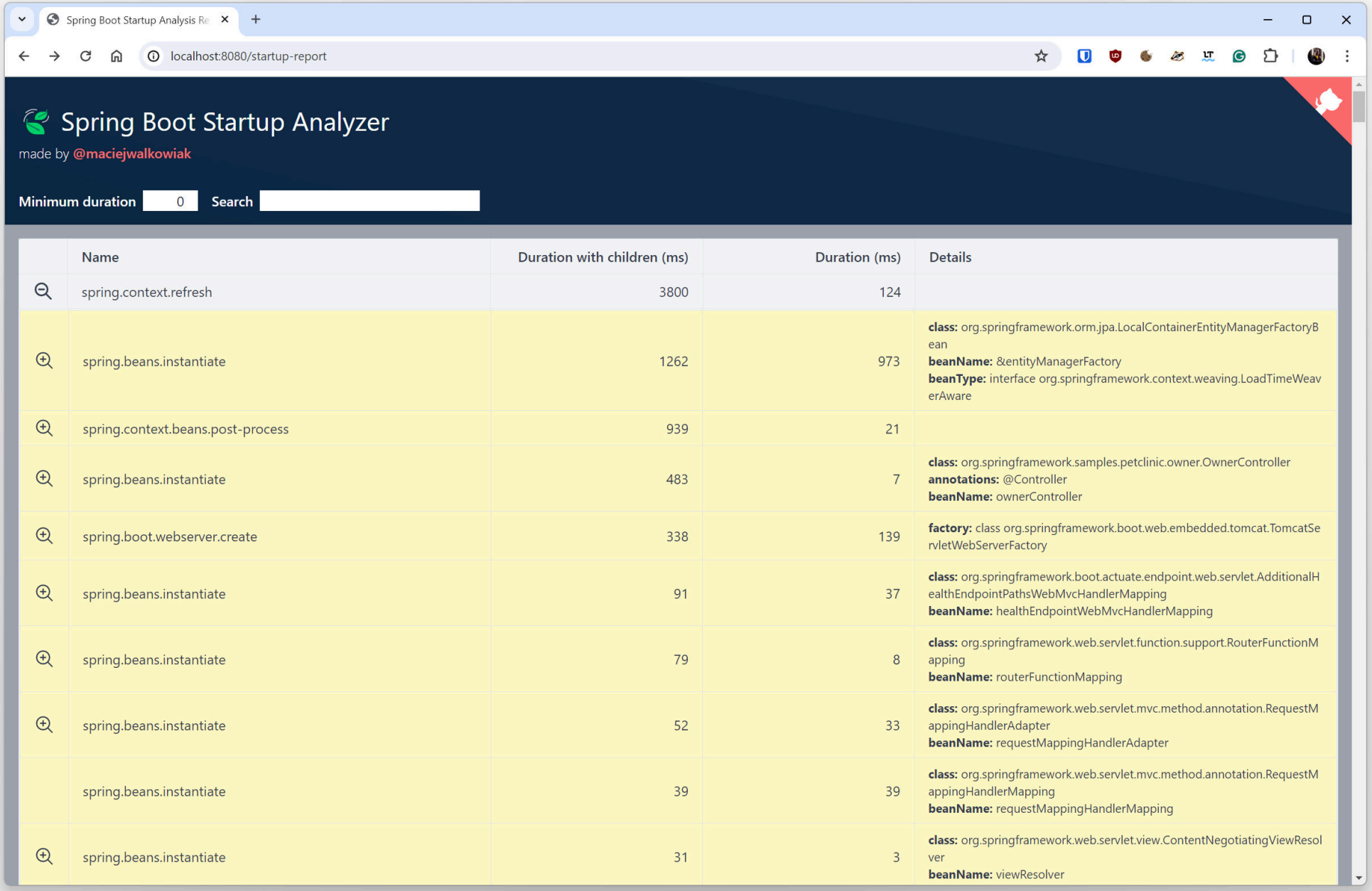
Spring Boot Startup Report

By Maciej Walkowiak

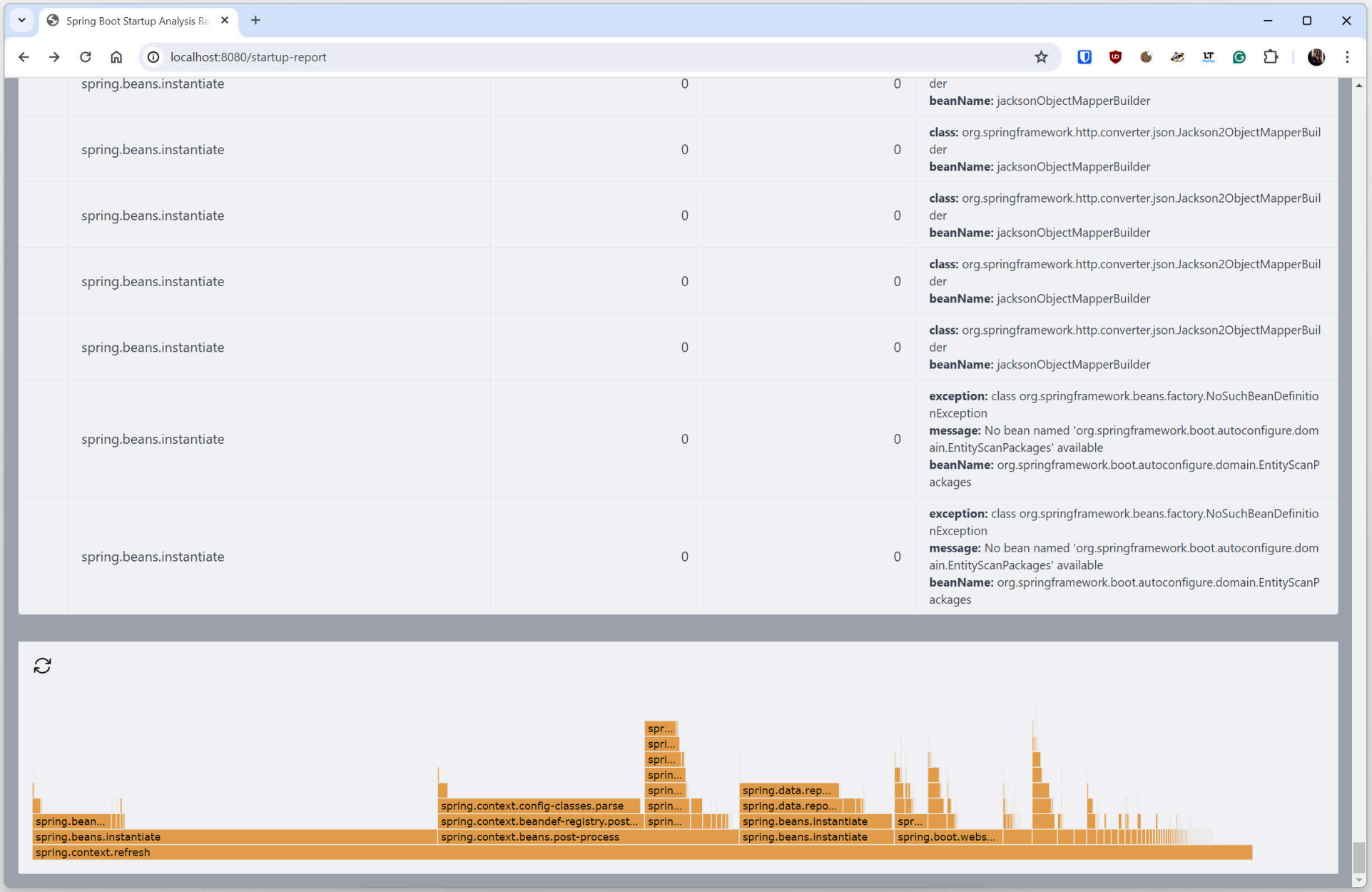
- Startup report available in runtime as an interactive HTML page
- Generation of startup reports in integration tests
- Flame chart for timings
- Search by class or annotation

```
<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



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.Jackson2ObjectMapperBuilder beanName: jacksonObjectMapperBuilder
spring.beans.instantiate 0 0 class: org.springframework.http.converter.json.Jackson2ObjectMapperBuilder beanName: jacksonObjectMapperBuilder
spring.beans.instantiate 0 0 class: org.springframework.http.converter.json.Jackson2ObjectMapperBuilder beanName: jacksonObjectMapperBuilder
spring.beans.instantiate 0 0 class: org.springframework.http.converter.json.Jackson2ObjectMapperBuilder beanName: jacksonObjectMapperBuilder
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.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.bean...
spring.beans.instantiate
spring.context.refresh
spring.context.config-classes.parse
spring.context.beandef-registry.post...
spring.context.beans.post-process
spr...
spri...
spri...
sprin...
sprin...
sprin...
sprin...
spring.data.rep...
spring.data.repo...
spring.beans.instantiate
spring.beans.instantiate
spr...
spring.boot.webs...

Benchmarks

Benchmarks

- Build
  - Maven build time
  - Artifact / Container Image size
- Startup
  - Startup time
  - Memory usage
- Throughput & Latency
  - `oha --no-tui -z60s --disable-keepalive <HOST>`
  - 30s warmup, 60s measurement
  - Docker container with 2 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 17.0.12-librca

mvn spring-boot:build-image

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

	Build	Image	Startup	Initial RAM	Requests/s	RAM	50%	75%	90%	99%	99.9%
⏱️ Java 17	68s	360MB	5.219s	255MB	845/s	369MB	59ms	81ms	98ms	169ms	247ms

No Optimizing - Baseline JDK 21

- Spring PetClinic (JDK 21 adjustments)
- Bellsoft Liberica JDK 21.0.4
- Java Memory Calculator

```
sdk use java 21.0.4-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	Requests/s	RAM	50%	75%	90%	99%	99.9%
⏱️ Java 17	68s	360MB	5.219s	255MB	845/s	369MB	59ms	81ms	98ms	169ms	247ms
Java 21	68s	388MB	5.19s	275MB	968/s	379MB	48ms	73ms	91ms	156ms	222ms

No Optimizing - Baseline JDK 22

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

```
sdk use java 22.0.2-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	Requests/s	RAM	50%	75%	90%	99%	99.9%
⏱ Java 17	68s	360MB	5.219s	255MB	845/s	369MB	59ms	81ms	98ms	169ms	247ms
Java 22	69s	384MB	5.107s	274MB	1016/s	394MB	44ms	69ms	88ms	152ms	214ms

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	Requests/s	RAM	50%	75%	90%	99%	99.9%
⏱️ Java 17	68s	360MB	5.219s	255MB	845/s	369MB	59ms	81ms	98ms	169ms	247ms
Java 17	-	-	4.749s	246MB	858/s	361MB	58ms	79ms	97ms	173ms	258ms
Java 21	-	-	4.734s	265MB	954/s	381MB	49ms	73ms	91ms	159ms	227ms

-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
```

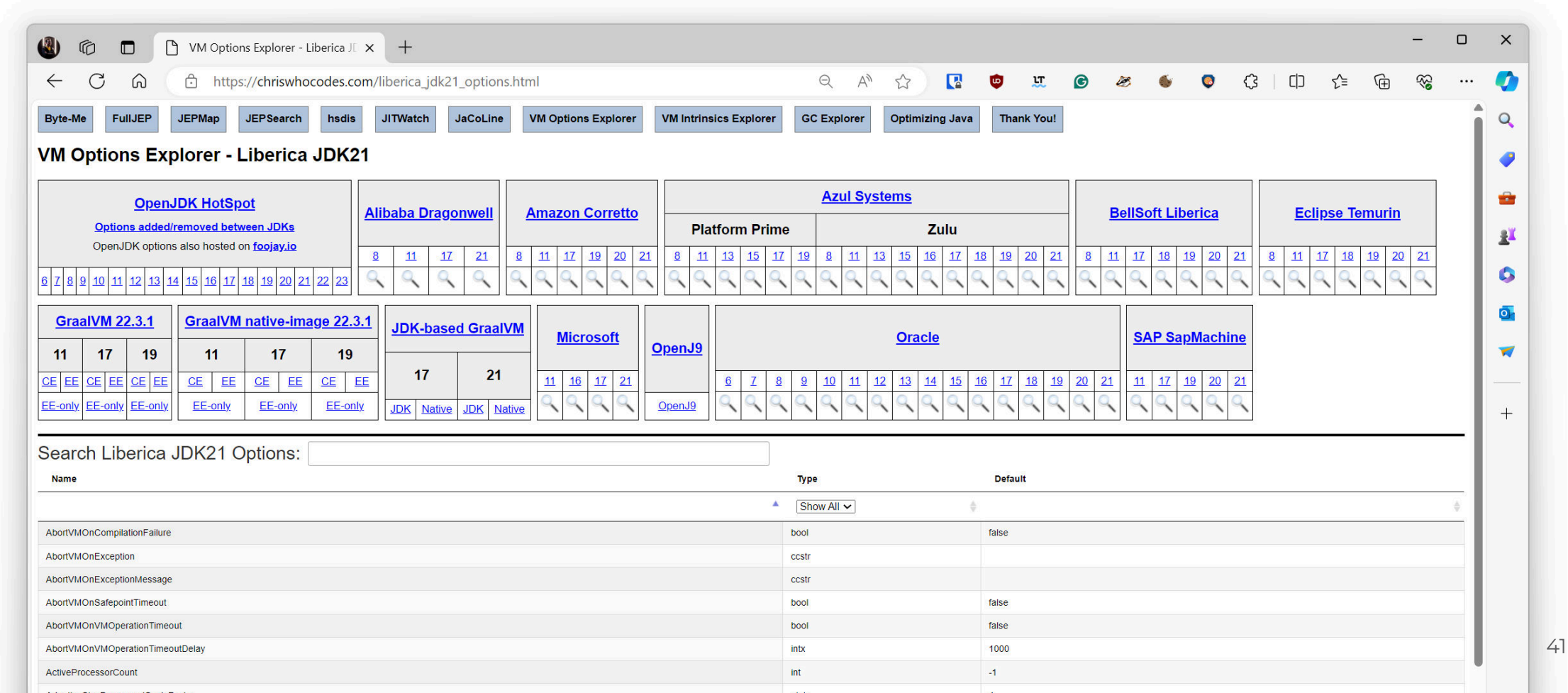
It will slow down the JIT later at the expense of the saved startup time!

	Build	Image	Startup	Initial RAM	T...	RAM	50%	75%	90%	99%	99.9%
⏱ Java 17	68s	360MB	5.219s	255MB	845/s	369MB	59ms	81ms	98ms	169ms	247ms
Java 17	-	-	3.9s	216MB	885/s	287MB	58ms	77ms	92ms	156ms	216ms
Java 21	-	-	4.21s	229MB	811/s	300MB	64ms	82ms	97ms	169ms	246ms

VM Options Explorer

VM Options Explorer

https://chriswhocodes.com



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	Requests/s	RAM	50%	75%	90%	99%	99.9%
⏱️ Java 17	68s	360MB	5.219s	255MB	845/s	369MB	59ms	81ms	98ms	169ms	247ms
Java 17	-	-	3.93s	228MB	825/s	370MB	62ms	82ms	98ms	170ms	247ms

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	Requests/s	RAM	50%	75%	90%	99%	99.9%
⏱ Java 17	68s	360MB	5.219s	255MB	845/s	369MB	59ms	81ms	98ms	169ms	247ms
Java 17	-	-	5.203s	265MB	834/s	372MB	61ms	81ms	98ms	172ms	239ms

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: 448
  - Spring Pet Clinic without actuators: 272 🔥

```
sdk use java 17.0.12-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	Requests/s	RAM	50%	75%	90%	99%	99.9%
⏱️ Java 17	68s	360MB	5.219s	255MB	845/s	369MB	59ms	81ms	98ms	169ms	247ms
Java 17	64s	358MB	4.104s	246MB	1011/s	352MB	44ms	69ms	88ms	152ms	212ms

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 17.0.12-librca

mvn spring-boot:build-image

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

	Build	Image	Startup	Initial RAM	Requests/s	RAM	50%	75%	90%	99%	99.9%
⏱ Java 17	68s	360MB	5.219s	255MB	845/s	369MB	59ms	81ms	98ms	169ms	247ms
Java 17	73s	361MB	4.654s	253MB	886/s	366MB	55ms	77ms	95ms	166ms	236ms
Java 21	72s	389MB	4.673s	273MB	938/s	379MB	50ms	74ms	92ms	165ms	235ms

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.

```
<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 [11]:
        com.h2database:h2:2.2.224:runtime (2 MB)
        com.mysql:mysql-connector-j:8.3.0:runtime (2 MB)
        org.postgresql:postgresql:42.7.4:runtime (1 MB)
        org.springframework.boot:spring-boot-devtools:3.3.4:test (198 KB)
        org.springframework.boot:spring-boot-docker-compose:3.3.4:test (192 KB)
        jakarta.xml.bind:jakarta.xml.bind-api:4.0.2:compile (127 KB)
        org.springframework.boot:spring-boot-testcontainers:3.3.4:test (117 KB)
...
USED TRANSITIVE DEPENDENCIES [62]:
        org.testcontainers:testcontainers:1.19.8:test (16 MB)
        net.bytebuddy:byte-buddy:1.14.19:runtime (4 MB)
        com.github.docker-java:docker-java-transport-zerodep:3.3.6:test (2 MB)
        org.springframework.boot:spring-boot-autoconfigure:3.3.4:compile (1 MB)
        org.springframework:spring-web:6.1.13:compile (1 MB)
        org.springframework:spring-core:6.1.13:compile (1 MB)
        net.java.dev.jna:jna:5.13.0:test (1 MB)
...
USED INHERITED DIRECT DEPENDENCIES [0]:
USED INHERITED TRANSITIVE DEPENDENCIES [0]:
POTENTIALLY UNUSED DIRECT DEPENDENCIES [9]:
        org.webjars.npm:bootstrap:5.3.3:compile (1 MB)
        com.github.ben-manes.caffeine:caffeine:3.1.8:compile (868 KB)
        org.webjars.npm:font-awesome:4.7.0:compile (665 KB)
        org.springframework.boot:spring-boot-starter-actuator:3.3.4:compile (4 KB)
        org.springframework.boot:spring-boot-starter-web:3.3.4:compile (4 KB)
        org.springframework.boot:spring-boot-starter-thymeleaf:3.3.4:compile (4 KB)
...
POTENTIALLY UNUSED TRANSITIVE DEPENDENCIES [49]:
        org.hibernate.orm:hibernate-core:6.5.3.Final:compile (11 MB)
        org.apache.tomcat.embed:tomcat-embed-core:10.1.30:compile (3 MB)
        org.aspectj:aspectjweaver:1.9.22.1:compile (2 MB)
        com.fasterxml.jackson.core:jackson-databind:2.17.2:compile (1 MB)
        org.hibernate.validator:hibernate-validator:8.0.1.Final:compile (1 MB)
        org.thymeleaf:thymeleaf:3.1.2.RELEASE:compile (916 KB)
        io.micrometer:micrometer-core:1.13.4:compile (842 KB)
...
POTENTIALLY UNUSED INHERITED DIRECT DEPENDENCIES [0]:
POTENTIALLY UNUSED INHERITED TRANSITIVE DEPENDENCIES [0]:
[INFO] Analysis done in 0min 15s
```

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 17.0.12-librca

mvn spring-boot:build-image

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

	Build	Image	Startup	Initial RAM	Requests/s	RAM	50%	75%	90%	99%	99.9%
⏱ Java 17	68s	360MB	5.219s	255MB	845/s	369MB	59ms	81ms	98ms	169ms	247ms
Java 17	63s	355MB	3.878s	239MB	983/s	347MB	47ms	71ms	89ms	153ms	223ms

More Application Optimisations

More Application Optimisations

- Application Optimizations
- Better Connection Pooling
- Improved Data Caching
- Database Queries Optimizations
- Batch Processing Optimizations
- Reduce Garbage Collection
- Monitor and Tune JVM Settings
- Reduce Overall Memory Footprint
- Async Logging
- ...

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 17.0.12-librca
```

```
m
d
```

	Build	Image	Startup	Initial RAM	Requests/s	RAM	50%	75%	90%	99%	99.9%
⏱ Java 17	68s	360MB	5.219s	255MB	845/s	369MB	59ms	81ms	98ms	169ms	247ms
Java 17	78s	271MB	5.235s	275MB	861/s	381MB	58ms	79ms	96ms	166ms	235ms
Java 21	77s	280MB	5.228s	285MB	1000/s	403MB	45ms	70ms	89ms	158ms	223ms

Java Dependency Analysis Tool

`jdeps` is a Java command-line tool that analyzes class and jar files to list the package-level or class-level dependencies, helping developers understand module dependencies and improve code modularity.

```
jar xvf target/spring-petclinic-3.3.0-SNAPSHOT.jar

jdeps --ignore-missing-deps -q  \
    --recursive  \
    --multi-release 17  \
    --print-module-deps  \
    --class-path 'BOOT-INF/lib/*'  \
    target/spring-petclinic-3.3.0-SNAPSHOT.jar
```

```
java.base,java.compiler,java.desktop,java.instrument,java.net.http,java.prefs,java.rmi,java.scripting,
java.security.jgss,java.sql.rowset,jdk.jfr,jdk.management,jdk.net,jdk.unsupported
```

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 java.base,java.compiler,java.desktop,java.instrument,
                java.net.http,java.prefs,java.rmi,java.scripting,java.security.jgss,java.sql.rowset,
                jdk.jfr,jdk.management,jdk.net,jdk.unsupported --compress=2 --no-header-files --no-man-pages
                --strip-debug</BP_JVM_JLINK_ARGS>
            </env>
        </image>
    </configuration>
</plugin>
```

	Build	Image	Startup	Initial RAM	Requests/s	RAM	50%	75%	90%	99%	99.9%
⏱ Java 17	68s	360MB	5.219s	255MB	845/s	369MB	59ms	81ms	98ms	169ms	247ms
Java 17	76s	260MB	5.284s	271MB	841/s	376MB	60ms	81ms	97ms	165ms	243ms
Java 21	76s	268MB	5.153s	282MB	1012/s	398MB	44ms	69ms	88ms	152ms	222ms

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.

```
<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 17.0.12-librca
```

	Build	Image	Startup	Initial RAM	Requests/s	RAM	50%	75%	90%	99%	99.9%
⏱ Java 17	68s	360MB	5.219s	255MB	845/s	369MB	59ms	81ms	98ms	169ms	247ms
Java 17	81s	511MB	3.439s	247MB	860/s	357MB	57ms	79ms	97ms	169ms	250ms
Java 21	80s	538MB	3.487s	256MB	985/s	385MB	46ms	71ms	90ms	159ms	220ms

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

	Build	Image	Startup	Initial RAM	Requests/s	RAM	50%	75%	90%	99%	99.9%
⏱ Java 17	68s	360MB	5.219s	255MB	845/s	369MB	59ms	81ms	98ms	169ms	247ms
Java 17	73s	335MB	5.819s	191MB	1404/s	341MB	25ms	51ms	73ms	120ms	225ms
Java 17	72s	353MB	6.104s	182MB	1420/s	338MB	24ms	49ms	73ms	131ms	211ms

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>
```

	Build	Image	Startup	Initial RAM	Requests/s	RAM	50%	75%	90%	99%	99.9%
⏱ Java 17	68s	360MB	5.219s	255MB	845/s	369MB	59ms	81ms	98ms	169ms	247ms
Java 17	77s	758MB	5.152s	289MB	893/s	376MB	55ms	77ms	94ms	166ms	240ms
Java 21	77s	743MB	5.317s	308MB	977/s	400MB	47ms	71ms	89ms	159ms	228ms

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>
```

	Build	Image	Startup	Initial RAM	Requests/s	RAM	50%	75%	90%	99%	99.9%
⏱ Java 17	68s	360MB	5.219s	255MB	845/s	369MB	59ms	81ms	98ms	169ms	247ms
Java 17	71s	499MB	5.178s	259MB	819/s	377MB	62ms	82ms	99ms	171ms	249ms
Java 21	69s	528MB	5.069s	275MB	989/s	393MB	46ms	71ms	90ms	155ms	217ms

Other Buildpack Builders

Bellsoft Buildpack Builder

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	Requests/s	RAM	50%	75%	90%	99%	99.9%
⏱ Java 17	68s	360MB	5.219s	255MB	845/s	369MB	59ms	81ms	98ms	169ms	247ms
musl	63s	174MB	5.438s	267MB	794/s	349MB	66ms	84ms	99ms	163ms	207ms
glibc	64s	194MB	5.15s	273MB	886/s	388MB	56ms	78ms	95ms	163ms	225ms

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>
```

	Build	Image	Startup	Initial RAM	Requests/s	RAM	50%	75%	90%	99%	99.9%
⏱️ Java 17	68s	360MB	5.219s	255MB	845/s	369MB	59ms	81ms	98ms	169ms	247ms
Java 17	69s	279MB	5.181s	260MB	840/s	371MB	60ms	81ms	98ms	169ms	232ms
Java 21	68s	307MB	5.171s	274MB	974/s	396MB	47ms	72ms	90ms	156ms	221ms

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	Requests/s	RAM	50%	75%	90%	99%	99.9%
⏱️ Java 17	68s	360MB	5.219s	255MB	845/s	369MB	59ms	81ms	98ms	169ms	247ms
CE 21	350s	243MB	0.419s	240MB	1212/s	312MB	33ms	58ms	77ms	124ms	191ms
Oracle 21	532s	250MB	0.319s	221MB	1586/s	256MB	23ms	45ms	61ms	98ms	151ms

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!

.. with many other challenges like: embedded configs and secrets, same OS (Linux) and CPU Architecture, same JVM versions, and more ...

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
```

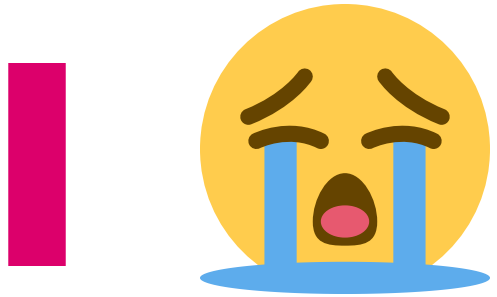
```
java -XX:CRaCRestoreFrom=crac-files
```

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.



No CRaC Buildpacks for Java & Spring Boot

Virtual Threads (Java 21)

Virtual Threads (1)

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 21.0.4-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	Requests/s	RAM	50%	75%	90%	99%	99.9%
⏱️ Java 17	68s	360MB	5.219s	255MB	845/s	369MB	59ms	81ms	98ms	169ms	247ms
Java 21	68s	388MB	5.302s	271MB	2291/s	380MB	19ms	22ms	27ms	58ms	65ms

Virtual Threads (2)

CPU	VT	Startup	Initial RAM	Request/s	RAM	50%	75%	90%	99%	99.9%	99.99%
4 vCPU		4.903s	297MB	2956/s	414MB	15ms	19ms	26ms	52ms	86ms	124ms
4 vCPU		5.002s	305MB	3203/s	425MB	15ms	17ms	19ms	29ms	72ms	127ms
2 vCPU		5.178s	273MB	1375/s	394MB	28ms	51ms	68ms	111ms	193ms	282ms
2 vCPU		5.277s	273MB	2665/s	377MB	18ms	19ms	22ms	27ms	38ms	48ms
1 vCPU		11.559s	269MB	264/s	381MB	187ms	217ms	308ms	527ms	757ms	1004ms
1 vCPU		12.291s	275MB	730/s	392MB	78ms	94ms	104ms	193ms	256ms	435ms
0.5 vCPU		34.56s	264MB	64/s	366MB	700ms	1030ms	1369ms	2067ms	2628ms	3028ms
0.5 vCPU		35.674s	266MB	73/s	378MB	661ms	768ms	1064ms	1531ms	2257ms	5460ms
0.25 vCPU		79.299s	269MB	22/s	344MB	2032ms	2932ms	4200ms	7224ms	9172ms	9570ms
0.25 vCPU		90.067s	267MB	18/s	347MB	2600ms	3400ms	4033ms	6383ms	13168ms	19398ms

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)

Request/s

- 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 21 & Bellsoft Builder (musl)

```
sdk use java 21.0.4-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	Requests/s	RAM	50%	75%	90%	99%	99.9%
⏱️ Java 17	68s	360MB	5.219s	255MB	845/s	369MB	59ms	81ms	98ms	169ms	247ms
Java 21	76s	203MB	3.304s	226MB	1081/s	318MB	39ms	65ms	84ms	141ms	204ms

Java 21 & Base Builder Tiny

```
sdk use java 21.0.4-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	Requests/s	RAM	50%	75%	90%	99%	99.9%
⏱️ Java 17	68s	360MB	5.219s	255MB	845/s	369MB	59ms	81ms	98ms	169ms	247ms
Java 21	83s	313MB	3.259s	228MB	1201/s	373MB	33ms	58ms	77ms	132ms	208ms

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