Lean Spring Boot

Applications for The Cloud

Patrick Baumgartner
42talents GmbH, Zürich, Switzerland

@patbaumgartner
patrick.baumgartner@42talents.com

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-context-indexer", classpath exclusions, lazy spring beans, actuator, JMX. In addition, we also look at switching to a different JVM and other tools.

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 is 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 @ Zürcher Fachhochschule für Angewandte Wissenschaften ZHAW

@patbaumgartner

What is the problem?
Why this talk?

JAVA 😉 & Spring Boot ❤

Requirements
When Choosing a Cloud

- How many vCPUs per server are required for my application?
- How much RAM do I need?
- How much storage is necessary?
- Which technology stack should I use?

Considerations

Resources

- CPU & RAM not linearly scalable
- Image Size & Network Bandwidth

Scaleability

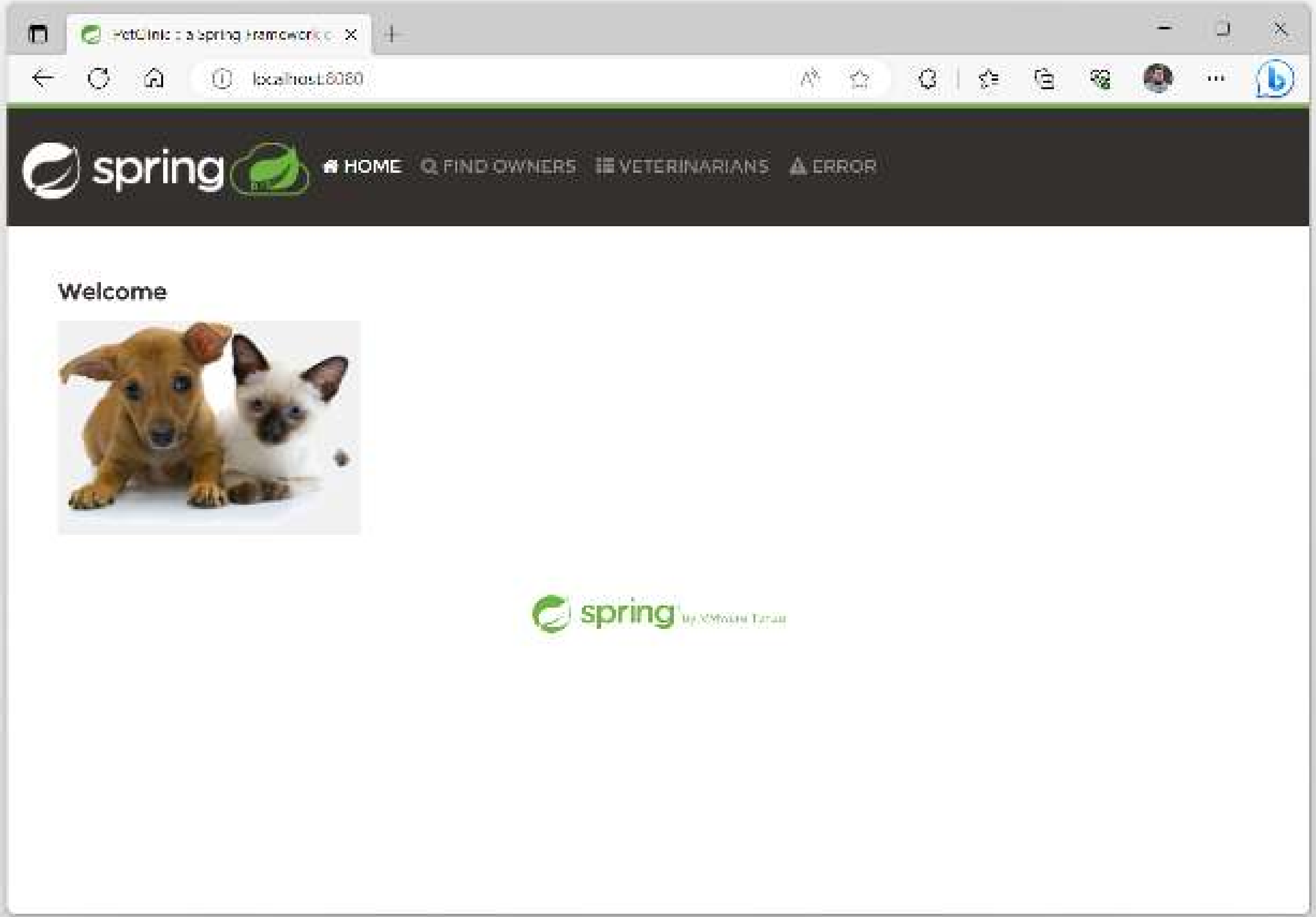
- Fast Startup
- Graceful Shutdown
- Throughput
- Latency

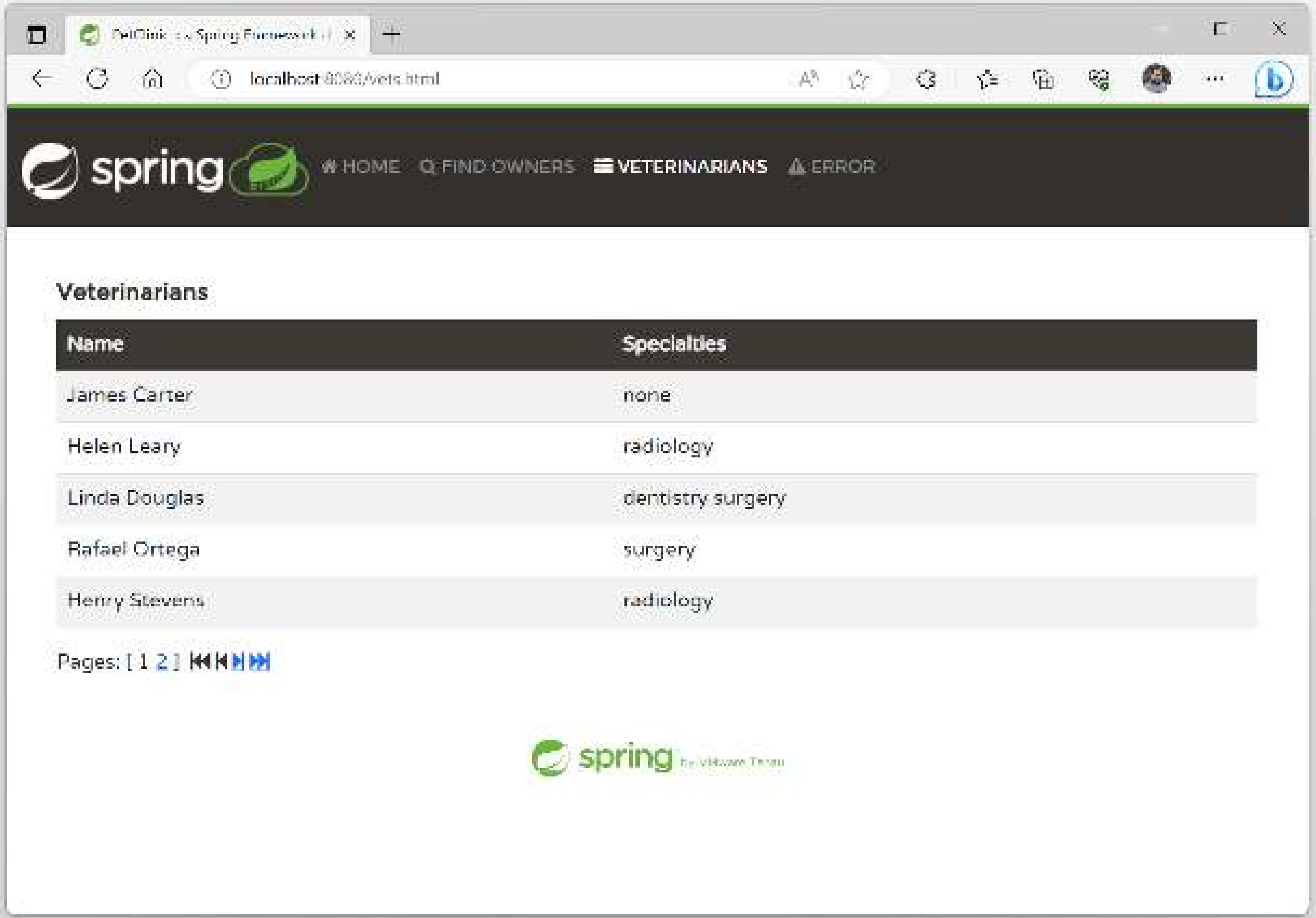
Agenda

Agenda

- Spring PetClinic & Baseline for Comparison
- Java Optimizations
- Spring Boot Optimizations
- Application Optimizations
- Other Runtimes
- Conclusions
- A Few Simple Optimizations Applied (OpenJDK Example)

Spring PetClinic


PetClinic :: a Spring Framework d
localhost:8080
spring
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

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

NO!

The official **Spring PetClinic!** 🐾🏥

Which is based on **Spring Boot**, Caffeine, Thymeleaf, **Spring Data JPA**, H2 and **Spring MVC** ...

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

Optimizing Experiments

Baseline

Technology Stack

- OCI Container (Buildpacks)
- Java JRE 17 LTS
- Spring Boot 3.1.2
- DB Migration with 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 php python

What are Buildpacks?

- Buildpacks transform your application source code into container images
- The Paketo open source project provides production-ready buildpacks for the most popular languages and frameworks
- Use Paketo Buildpacks to easily build your apps and keep them updated
- Paketo Buildpacks are leveraged by many application platforms and local development tools including Hashicorp Waypoint, the Pack CLI, Spring Boot, Tilt, and VMware Tanzu Build Service

See also: https://buildpacks.io/ and https://paketo.io/

Buildpacks FTW!

- Spring Boot plugin uses "Buildpacks" during the `build-image` task. It detects the Spring Boot App and optimizes created container:
- Optimizes the runtime by:
  - Extracting the fat JAR into exploded form.
  - Calculates and applies resource runtime tuning at container startup.
- Optimized the container image:
  - Adds layers from Buildpack, spring boot, ...
  - Subsequent builds are faster, they only build and add layers for the changed code.

```
Setting Active Processor Count to 4
Calculating JVM memory based on 16965780K 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 -Xmx16335617K -XX:MaxMetaspaceSize=118162K -XX:ReservedCodeCacheSize=240M -Xss1M (Total Memory: 16965780K, Thread Count: 250, Loaded Class Count: 18448, 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 -Xmx16335617K -XX:MaxMetaspaceSize=118162K -XX:ReservedCodeCacheSize=240M -Xss1M
-XX:+UnlockDiagnosticVMOptions -XX:NativeMemoryTracking=summary -XX:+PrintNMTStatistics -Dorg.springframework.cloud.bindings.boot.enable=true


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

:: Built with Spring Boot :: 3.1.2


2023-08-23T17:17:37.209Z  INFO 1 --- [           main] o.s.s.petclinic.PetClinicApplication     : Starting PetClinicApplication v3.1.2-SNAPSHOT using Java 17.0.7 with PID 1 (/workspace/BOOT-INF/classes started by cnb in /workspace)
2023-08-23T17:17:37.212Z  INFO 1 --- [           main] o.s.s.petclinic.PetClinicApplication     : No active profile set, falling back to 1 default profile: "default"
2023-08-23T17:17:37.927Z  INFO 1 --- [           main] .s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data JPA repositories in DEFAULT mode.
2023-08-23T17:17:37.968Z  INFO 1 --- [           main] .s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository scanning in 36 ms. Found 2 JPA repository interfaces.
2023-08-23T17:17:38.379Z  INFO 1 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port(s): 8080 (http)
2023-08-23T17:17:38.386Z  INFO 1 --- [           main] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
2023-08-23T17:17:38.386Z  INFO 1 --- [           main] o.apache.catalina.core.StandardEngine    : Starting Servlet engine: [Apache Tomcat/10.1.10]
2023-08-23T17:17:38.445Z  INFO 1 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
2023-08-23T17:17:38.447Z  INFO 1 --- [           main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 1202 ms
2023-08-23T17:17:38.613Z  INFO 1 --- [           main] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Starting...
2023-08-23T17:17:38.768Z  INFO 1 --- [           main] com.zaxxer.hikari.pool.HikariPool        : HikariPool-1 - Added connection conn0: url=jdbc:h2:mem:62f6cba7-d5f4-4c9b-8d3b-f66ce0f168e1 user=SA
2023-08-23T17:17:38.770Z  INFO 1 --- [           main] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Start completed.
2023-08-23T17:17:38.899Z  INFO 1 --- [           main] o.hibernate.jpa.internal.util.LogHelper  : HHH000204: Processing PersistenceUnitInfo [name: default]
2023-08-23T17:17:38.931Z  INFO 1 --- [           main] org.hibernate.Version                    : HHH000412: Hibernate ORM core version 6.2.5.Final
2023-08-23T17:17:38.933Z  INFO 1 --- [           main] org.hibernate.cfg.Environment            : HHH000406: Using bytecode reflection optimizer
2023-08-23T17:17:39.017Z  INFO 1 --- [           main] o.h.b.i.BytecodeProviderInitiator        : HHH000021: Bytecode provider name : bytebuddy
2023-08-23T17:17:39.107Z  INFO 1 --- [           main] o.s.o.j.p.SpringPersistenceUnitInfo      : No LoadTimeWeaver setup: ignoring JPA class transformer
2023-08-23T17:17:39.330Z  INFO 1 --- [           main] o.h.b.i.BytecodeProviderInitiator        : HHH000021: Bytecode provider name : bytebuddy
2023-08-23T17:17:39.915Z  INFO 1 --- [           main] o.h.e.t.j.p.i.JtaPlatformInitiator       : HHH000490: Using JtaPlatform implementation: [org.hibernate.engine.transaction.jta.platform.internal.NoJtaPlatform]
2023-08-23T17:17:39.917Z  INFO 1 --- [           main] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persistence unit 'default'
2023-08-23T17:17:40.090Z  INFO 1 --- [           main] o.s.d.j.r.query.QueryEnhancerFactory     : Hibernate is in classpath; If applicable, HQL parser will be used.
2023-08-23T17:17:40.881Z  INFO 1 --- [           main] o.s.b.a.e.web.EndpointLinksResolver      : Exposing 13 endpoint(s) beneath base path '/actuator'
2023-08-23T17:17:40.953Z  INFO 1 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port(s): 8080 (http) with context path ''
2023-08-23T17:17:40.967Z  INFO 1 --- [           main] o.s.s.petclinic.PetClinicApplication     : Started PetClinicApplication in 4.069 seconds (process running for 4.351)
```

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

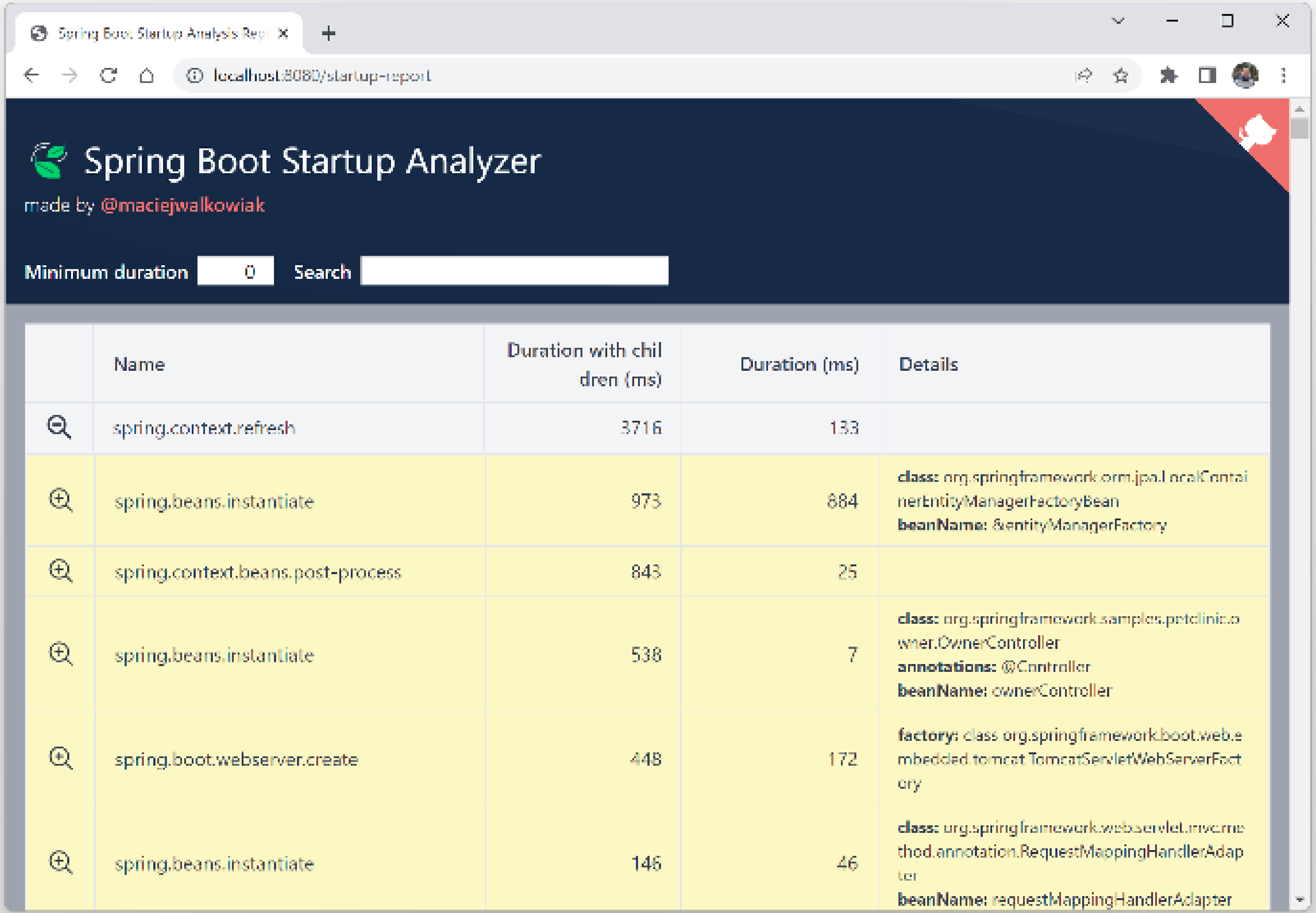
Spring Boot Startup Report

By Maciej Walkowiak

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

```xml
<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

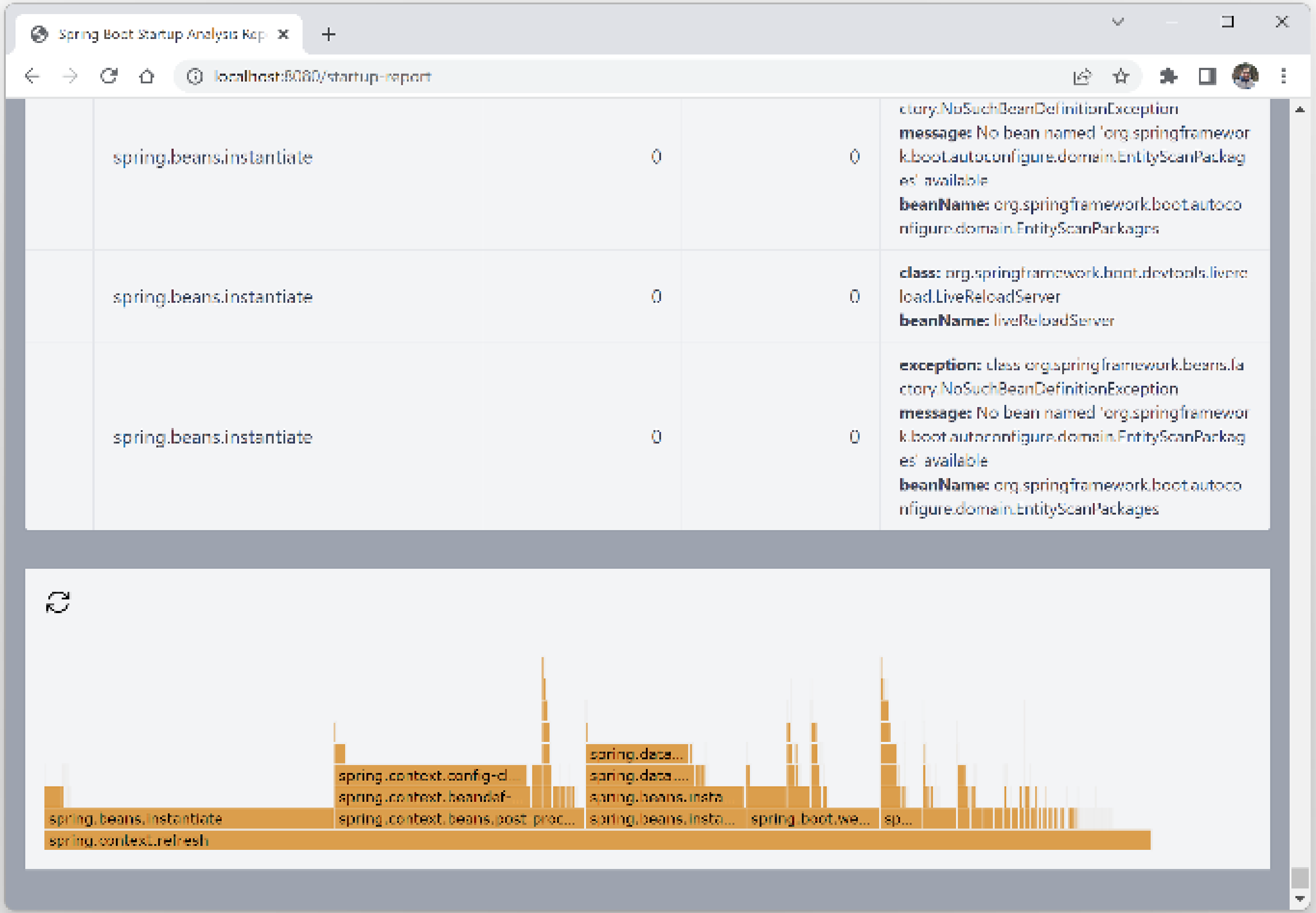

Spring Boot Startup Analysis Rep
localhost:8080/startup-report

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.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

Benchmarks

Benchmarks

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

No Optimizing - Baseline

No Optimizing - Baseline JRE 17

- Spring PetClinic (no adjustments)
- Bellsoft Liberica JRE 17.0.7
- Java Memory Calculator

```
sdk use java 17.0.8-librca

mvn spring-boot:build-image

docker run -p 8080:8080 -t spring-petclinic:3.1.2-SNAPSHOT
```

Build	Image Size	Startup	Initial RAM	Requests	RAM	99%	99.9%	99.99%
⏱ 53.2s	316MB	3.971s	273MB	1990/s	447MB	30ms	51ms	87ms

No Optimizing - Baseline JRE 20

- Spring PetClinic (JDK 20 adjustments)
- Bellsoft Liberica JRE 20.0.1
- Java Memory Calculator

```
sdk use java 20.0.2-librca

mvn -Djava.version=20 spring-boot:build-image \
    -Dspring-boot.build-image.imageName=spring-petclinic:3.1.2-SNAPSHOT-jdk20

docker run -p 8080:8080 -t spring-petclinic:3.1.2-SNAPSHOT-jdk20
```

Build	Image Size	Startup	Initial RAM	Requests	RAM	99%	99.9%	99.99%
⏱ 53.2s	316MB	3.971s	273MB	1990/s	447MB	30ms	51ms	87ms
54.3s	351MB	3.921s	281MB	1989/s	447MB	36ms	56ms	77ms

-XX:TieredStopAtLevel=1

-XX:TieredStopAtLevel=1

The 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 only use the C1 compiler with no profiling overhead, we can apply the `-XX:TieredStopAtLevel=1` parameter.

```
docker run -p 8080:8080 -e "JAVA_TOOL_OPTIONS=-XX:TieredStopAtLevel=1" \
  -t spring-petclinic:3.1.2-SNAPSHOT
```

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

Build	Image Size	Startup	Initial RAM	Requests	RAM	99%	99.9%	99.99%
⏱ 53.2s	316MB	3.971s	273MB	1990/s	447MB	30ms	51ms	87ms
-	-	3.404s	201MB	1347/s	318MB	21ms	22ms	23ms

Spring Context Indexer

Spring Context Indexer (1)

The `spring-context-indexer` artifact generates a `META-INF/spring.components` file that is included in the JAR file. When the `ApplicationContext` detects such an index, it automatically uses it rather than scanning the classpath.

```
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context-indexer</artifactId>
    <optional>true<
</dependency>
```

Build	Image Size	Startup	Initial RAM	Requests	RAM	99%	99.9%	99.99%
⏱ 53.2s	316MB	3.971s	273MB	1990/s	447MB	30ms	51ms	87ms
51.7s	316MB	3.881s	282MB	1989/s	444MB	40ms	69ms	94ms

```
sdk use java 17.0.8-librca

mvn spring-boot:build-image

docker run -p 8080:8080 -t spring-petclinic-indexer:3.1.0-SNAPSHOT
```

Spring Context Indexer (2)

`META-INF/spring.components`

```
org.springframework.samples.petclinic.PetClinicApplication=org.springframework.stereotype.Component,org.springframework.boot.SpringBootConfiguration
org.springframework.samples.petclinic.model=package-info
org.springframework.samples.petclinic.model.BaseEntity=jakarta.persistence.MappedSuperclass
org.springframework.samples.petclinic.model.NamedEntity=jakarta.persistence.MappedSuperclass
org.springframework.samples.petclinic.model.Person=jakarta.persistence.MappedSuperclass
org.springframework.samples.petclinic.owner.Owner=jakarta.persistence.Entity,jakarta.persistence.Table
org.springframework.samples.petclinic.owner.OwnerController=org.springframework.stereotype.Component
org.springframework.samples.petclinic.owner.OwnerRepository=org.springframework.data.repository.Repository
org.springframework.samples.petclinic.owner.Pet=jakarta.persistence.Entity,jakarta.persistence.Table
org.springframework.samples.petclinic.owner.PetController=org.springframework.stereotype.Component
org.springframework.samples.petclinic.owner.PetType=jakarta.persistence.Entity,jakarta.persistence.Table
org.springframework.samples.petclinic.owner.PetTypeFormatter=org.springframework.stereotype.Component
org.springframework.samples.petclinic.owner.Visit=jakarta.persistence.Entity,jakarta.persistence.Table
org.springframework.samples.petclinic.owner.VisitController=org.springframework.stereotype.Component
org.springframework.samples.petclinic.system.CacheConfiguration=org.springframework.stereotype.Component
org.springframework.samples.petclinic.system.CrashController=org.springframework.stereotype.Component
org.springframework.samples.petclinic.system.WelcomeController=org.springframework.stereotype.Component
org.springframework.samples.petclinic.vet.Specialty=jakarta.persistence.Entity,jakarta.persistence.Table
org.springframework.samples.petclinic.vet.Vet=jakarta.persistence.Entity,jakarta.persistence.Table
org.springframework.samples.petclinic.vet.VetController=org.springframework.stereotype.Component
org.springframework.samples.petclinic.vet.VetRepository=org.springframework.data.repository.Repository
org.springframework.samples.petclinic.vet.Vets=jakarta.xml.bind.annotation.XmlRootElement
```

Lazy Spring Beans

Lazy Spring Beans (1)

Configure lazy initialization across the whole application. A Spring Boot property makes all Beans lazy by default and only initializes them when they are needed. `@Lazy` can be used to override this behavior with 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:3.1.2-SNAPSHOT
```

Build	Image Size	Startup	Initial RAM	Requests	RAM	99%	99.9%	99.99%
⏱ 53.2s	316MB	3.971s	273MB	1990/s	447MB	30ms	51ms	87ms
-	-	2.207s	233MB	1993/s	438MB	30ms	60ms	109ms

Lazy Spring Beans (2)

Pros

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

Cons

- The initial requests may take more time
- Class loading issues and misconfigurations unnoticed at startup
- Beans creation errors only be found at the time of loading the bean

No Spring Boot Actuators

No Spring Boot Actuators

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

- No. of Spring Beans
  - Spring Pet Clinic with Actuators: 415
  - Spring Pet Clinic no Actuators: 270 🔥

```
sdk use java 17.0.8-librca

mvn spring-boot:build-image

docker run -p 8080:8080 -t spring-petclinic-no-actuator:3.1.2-SNAPSHOT
```

Build	Image Size	Startup	Initial RAM	Requests	RAM	99%	99.9%	99.99%
⏱ 53.2s	316MB	3.971s	273MB	1990/s	447MB	30ms	51ms	87ms
49.4s	313MB	3.404s	273MB	1994/s	435MB	29ms	59ms	100ms

Fixing Spring Boot Config Location

Fixing Spring Boot Config Location

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

- Application properties packaged inside your jar
- Profile-specific application properties packaged inside your jar
- Application properties outside of your packaged jar
- Profile-specific application properties outside of your packaged jar

```
docker run -p 8080:8080 -e spring.config.location=classpath:application.properties \
  -t spring-petclinic:3.1.2-SNAPSHOT
```

Build	Image Size	Startup	Initial RAM	Requests	RAM	99%	99.9%	99.99%
⏱ 53.2s	316MB	3.971s	273MB	1990/s	447MB	30ms	51ms	87ms
-	-	3.955s	274MB	1989/s	438MB	29ms	56ms	84ms

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:3.1.2-SNAPSHOT
```

I ❤️

Spring Boot 🍃 & Buildpacks

Dependency Cleanup

Dependency Cleanup (2)

DepClean detects and removes all the unused dependencies declared in the `pom.xml` file of a project or imported from its parent. It does not touch the original `pom.xml` file.

```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
```

```
[INFO] Starting DepClean dependency analysis
-------------------------------------------------------
 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 [5]:
        jakarta.xml.bind:jakarta.xml.bind-api:4.0.0:compile (124 KB)
        org.springframework.boot:spring-boot-testcontainers:3.1.2:test (85 KB)
        javax.cache:cache-api:1.1.1:compile (50 KB)
        ...
USED TRANSITIVE DEPENDENCIES [22]:
        org.springframework:spring-core:6.0.10:compile (1 MB)
        org.springframework.boot:spring-boot-autoconfigure:3.1.2:compile (1 MB)
        org.springframework:spring-web:6.0.10:compile (1 MB)
        org.springframework.boot:spring-boot:3.1.2:compile (1 MB)
        org.springframework.data:spring-data-commons:3.1.2:compile (1 MB)
        org.assertj:assertj-core:3.24.2:test (1 MB)
        org.springframework:spring-context:6.0.10:compile (1 MB)
        org.springframework.data:spring-data-jpa:3.1.2:compile (1 MB)
        ...
USED INHERITED DIRECT DEPENDENCIES [0]:
USED INHERITED TRANSITIVE DEPENDENCIES [0]:
POTENTIALLY UNUSED DIRECT DEPENDENCIES [15]:
        com.h2database:h2:2.1.214:runtime (2 MB)
        com.mysql:mysql-connector-j:8.0.33:runtime (2 MB)
        org.webjars.npm:bootstrap:5.2.3:compile (1 MB)
        org.postgresql:postgresql:42.6.0:runtime (1 MB)
        com.github.ben-manes.caffeine:caffeine:3.1.6:compile (734 KB)
        org.webjars.npm:font-awesome:4.7.0:compile (665 KB)
        ...
POTENTIALLY UNUSED TRANSITIVE DEPENDENCIES [87]:
        org.testcontainers:testcontainers:1.18.3:test (11 MB)
        org.hibernate.orm:hibernate-core:6.2.5.Final:compile (10 MB)
        net.bytebuddy:byte-buddy:1.14.5:runtime (3 MB)
        org.apache.tomcat.embed:tomcat-embed-core:10.1.10:compile (3 MB)
        org.aspectj:aspectjweaver:1.9.19:compile (1 MB)
        com.github.docker-java:docker-java-transport-zerodep:3.3.0:test (1 MB)
        net.java.dev.jna:jna:5.12.1:test (1 MB)
        com.fasterxml.jackson.core:jackson-databind:2.15.2:compile (1 MB)
        org.hibernate.validator:hibernate-validator:8.0.0.Final:compile (1 MB)
        org.apache.commons:commons-compress:1.23.0:test (1 MB)
        org.thymeleaf:thymeleaf:3.1.2.RELEASE:compile (915 KB)
        ...
POTENTIALLY UNUSED INHERITED DIRECT DEPENDENCIES [0]:
POTENTIALLY UNUSED INHERITED TRANSITIVE DEPENDENCIES [0]:
[INFO] Analysis done in 0min 8s
```

Dependency Cleanup (2)

But there are some challenges:

- Spring uses reflection to load classes
- Spring Boot uses `META-INF/spring-boot/org.springframework.boot.autoconfigure.AutoConfiguration` to load classes
- Spring Context Indexer uses `META-INF/spring.components`
- Component & Entity Scanning through Classpath Scanning

```
sdk use java 17.0.8-librca

mvn spring-boot:build-image

docker run -p 8080:8080 -t spring-petclinic-depclean:3.1.2-SNAPSHOT
```

Build	Image Size	Startup	Initial RAM	Requests	RAM	99%	99.9%	99.99%
⏱ 53.2s	316MB	3.971s	273MB	1990/s	447MB	30ms	51ms	87ms
71.5s	307MB	3.383s	261MB	1997/s	413MB	33ms	53ms	92ms

Ahead-of-Time Processing (AOT)

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

Spring AOT is a process that analyzes your application at build-time and generates an optimized version of it.

As the `BeanFactory` is fully prepared at build-time, conditions are also evaluated.

```xml
<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) (1)

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

```
sdk use java 17.0.8-librca

mvn spring-boot:build-image

docker run -e spring.aot.enabled=true -p 8080:8080 -t spring-petclinic-aot:3.1.2-SNAPSHOT
```

Build	Image Size	Startup	Initial RAM	Requests	RAM	99%	99.9%	99.99%
⏱ 53.2s	316MB	3.971s	273MB	1990/s	447MB	30ms	51ms	87ms
54.8s	317MB	3.966s	288MB	1996/s	455MB	34ms	71ms	109ms

JLink

JLink (1)

Jlink assembles and optimizes 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)

```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>
            </env>
        </image>
    </configuration>
</plugin>
```

```
sdk use java 17.0.8-librca

mvn spring-boot:build-image

docker run -p 8080:8080 -t spring-petclinic-jlink:3.1.2-SNAPSHOT
```

JLink (3)

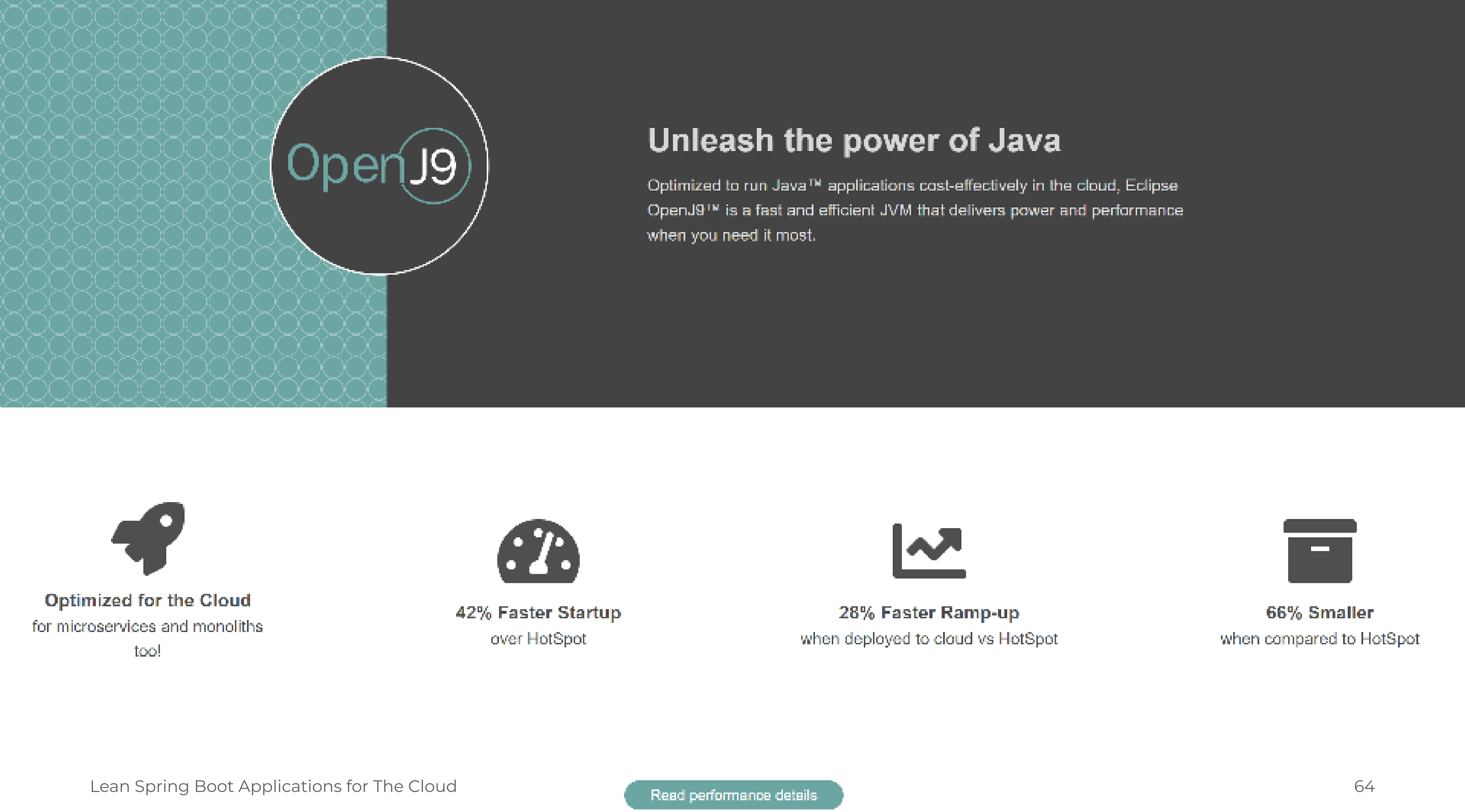
```
<plugin>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-maven-plugin</artifactId>
    <configuration>
        <image>
            <env>
                <BP_JVM_JLINK_ENABLED>true</BP_JVM_JLINK_ENABLED>
                <BP_JVM_JLINK_ARGS>--add-modules jdk.management.agent,java.base,java.logging,
                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 Size	Startup	Initial RAM	Requests	RAM	99%	99.9%	99.99%
⏱ 53.2s	316MB	3.971s	273MB	1990/s	447MB	30ms	51ms	87ms
68.5s	236MB	4.015s	295MB	1993/s	456MB	28ms	45ms	76ms

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!

42% Faster Startup
over HotSpot

28% Faster Ramp-up
when deployed to cloud vs HotSpot

66% Smaller
when compared to HotSpot

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

Build	Image Size	Startup	Initial RAM	Requests	RAM	99%	99.9%	99.99%
⏱ 53.2s	316MB	3.971s	273MB	1990/s	447MB	30ms	51ms	87ms
75.0s	305MB	6.997s	165MB	1996/s	350MB	61ms	104ms	161ms

```
sdk use java 17.0.8

mvn spring-boot:build-image

docker run -p 8080:8080 -t spring-petclinic-custom-jvm-openj9:3.1.2-SNAPSHOT
```

Eclipse OpenJ9 Optimized

Eclipse OpenJ9 Optimized

`-Xquickstart` causes the JIT compiler to run with a subset of optimizations, which can improve the performance of short-running applications.

Use the `-Xshareclasses` option to enable, disable, or modify class-sharing behavior. Class data sharing is enabled by default for bootstrap classes only, *unless your application is running in a container.*

```
docker run -p 8080:8080 -e "JAVA_TOOL_OPTIONS=-Xshareclasses -Xquickstart" \
  -t spring-petclinic-custom-jvm:3.1.2-SNAPSHOT
```

Build	Image Size	Startup	Initial RAM	Requests	RAM	99%	99.9%	99.99%
⏱ 53.2s	316MB	3.971s	273MB	1990/s	447MB	30ms	51ms	87ms
-	-	5.181s	162MB	1177/s	309MB	26ms	27ms	28ms

GraalVM

GraalVM

```xml
<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>
```

Build	Image Size	Startup	Initial RAM	Requests	RAM	99%	99.9%	99.99%
⏱ 53.2s	316MB	3.971s	273MB	1990/s	447MB	30ms	51ms	87ms
76.0s	734MB	3.841s	239MB	1997/s	429MB	32ms	56ms	81ms

```
sdk use java 17.0.8

mvn spring-boot:build-image

docker run -p 8080:8080 -t spring-petclinic-custom-jvm-graalvm:3.1.2-SNAPSHOT
```

Other Buildpack Builders

Bellsoft Buildpack Builder

Bellsoft provides an optimized 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 Size	Startup	Initial RAM	Requests	RAM	99%	99.9%	99.99%
⏱ 53.2s	316MB	3.971s	273MB	1990/s	447MB	30ms	51ms	87ms
51.6s	174MB	4.407s	250MB	1993/s	398MB	28ms	56ms	76ms

```
sdk use java 17.0.8-librca

mvn spring-boot:build-image

docker run -p 8080:8080 -t spring-petclinic-bellsoft-buildpack:3.1.2-SNAPSHOT
```

GraalVM Native Image

GraalVM Native Image

A native image is a technology to build Java code to a standalone executable. This executable includes the application classes, classes from its dependencies, runtime library classes, and statically linked native code from JDK. The JVM is packaged into the native image, so there's no need for any Java Runtime Environment at 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:3.1.2-SNAPSHOT
```

Build	Image Size	Startup	Initial RAM	Requests	RAM	99%	99.9%	99.99%
⏱ 53.2s	316MB	3.971s	273MB	1990/s	447MB	30ms	51ms	87ms
319.5s	216MB	0.291s	231MB	1995/s	461MB	244ms	379ms	514ms

CRaC - OpenJDK

CRaC - OpenJDK (1)

CRaC (Checkpoint and Restart in Java) is a feature that allows to checkpoint the state of a Java application and restart it from the checkpointed state.

The application starts within milliseconds!

CRaC - OpenJDK (2)

```
export JAVA_HOME=/opt/openjdk-17-crac+5_linux-x64/
export PATH=$JAVA_HOME/bin:$PATH
```

```
mvn clean verify
```

```
java -XX:CRaCCheckpointTo=crac-files -jar target/spring-petclinic-crac-3.1.2.jar
```

```
jcmd target/spring-petclinic-crac-3.1.2.jar JDK.checkpoint
```

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

CRaC - OpenJDK (3)

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

- Works with Spring Boot 3.0 & 3.1
    - Only patched Tomcat 10.1.7 is available
- Support for Spring Boot 3.2 is in progress
    - Spring Framework 6.1.0-SNAPSHOT
- Does not work on Windows or on macOS
    - But on Ubuntu 22.04 LTS
- Does not work in Docker containers via WSL (yet)

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

Summary

Summary

- No Optimizations with JRE 17 & JRE 20
- JVM Tuning with `-XX:TieredStopAtLevel=1`
- Spring Context Indexer
- 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 might not need a full CPU at runtime
- It will need multiple CPUs to start up as quickly as possible (at least 2, 4 are better)
- If you don't mind a slower startup you could throttle the CPUs down 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
- Using proper load testing can help to find the optimal configuration for your application

Conclusions (3)

Other Runtimes

- CRIU Support for OpenJDK and OpenJ9 is promising
- 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 a great option for running apps with less memory
  - But startup times are longer than with HotSpot
- Depending on the distribution, you might get other interesting features
  - Oracle GraalVM Enterprise Edition, Azul Platform Prime, IBM Semeru Runtime, …

Conclusions (4)

Other Ideas

- Using an Obfuscator like ProGuard
- Importing `AutoConfiguration` classes individually
- Using functional bean definitions
- More JVM tuning

See also: https://spring.io/blog/2019/01/21/manual-bean-definitions-in-spring-boot

A Few Simple Optimizations Applied

A Few Simple Optimizations Applied (1)

- Dependency Cleanup
    - DB Drivers, Spring Boot Actuator, Jackson, Tomcat Websocket, …
- Bellsoft Buildpack
- JLink
- JVM Parameters (java-memory-calculator)
- Spring AOT
- Lazy Spring Beans
- Fix Spring Boot Config Location

A Few Simple Optimizations Applied (2)

```
sdk use java 17.0.8-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:3.1.2-SNAPSHOT
```

Build	Image Size	Startup	Initial RAM	Requests	RAM	99%	99.9%	99.99%
⏱ 53.2s	316MB	3.971s	273MB	1990/s	447MB	30ms	51ms	87ms
62.5s	136MB	1.866s	190MB	1994/s	357MB	32ms	61ms	91ms

Did I miss something? 🧐

Let me/us know! 🙋

… or not! 🙊

Lean Spring Boot

Applications for The Cloud

Patrick Baumgartner

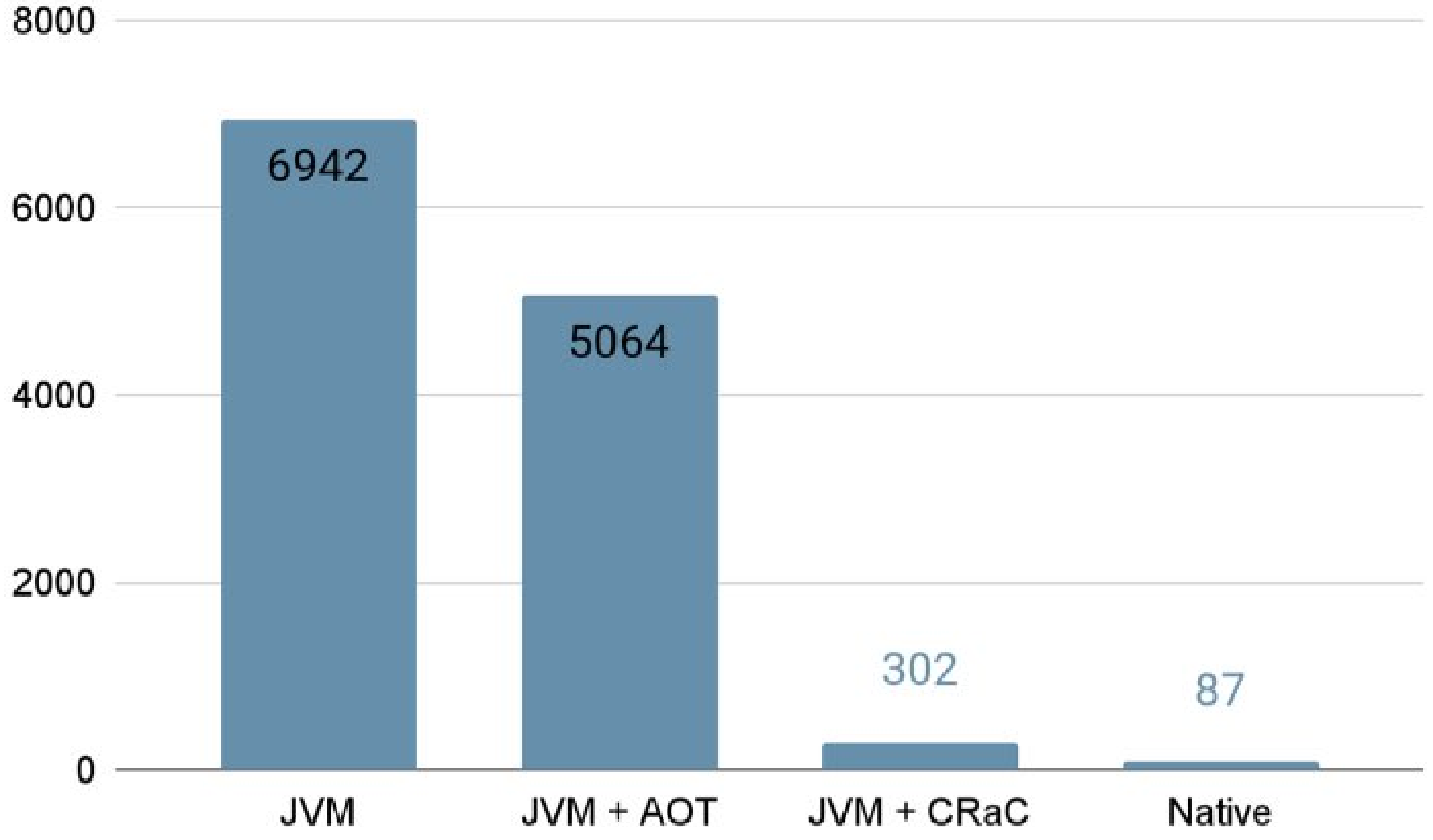
42talents GmbH, Zürich, Switzerland

@patbaumgartner

patrick.baumgartner@42talents.com

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