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

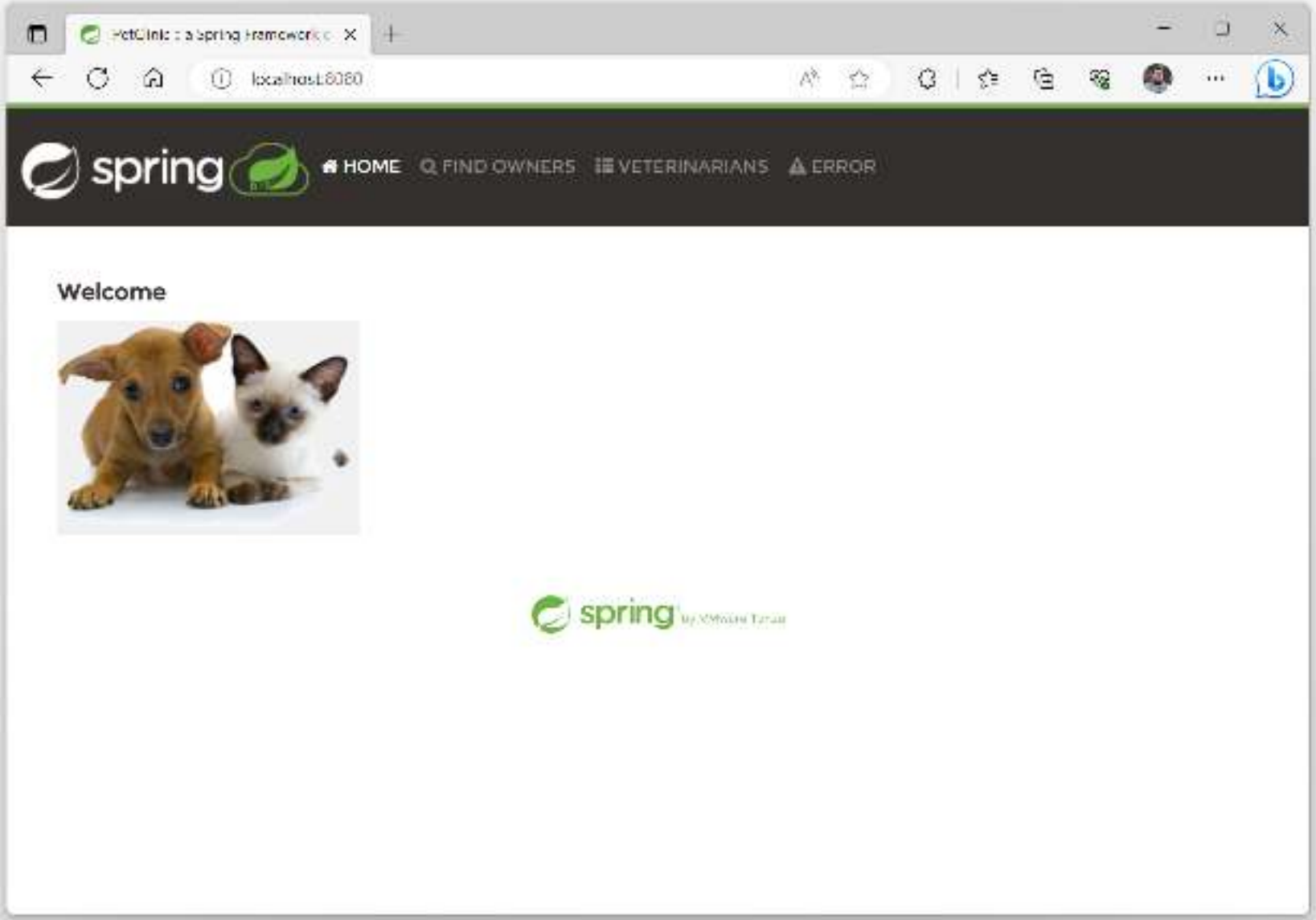
- Fast Startup
- Graceful Shutdown
- Throughput
- Latency

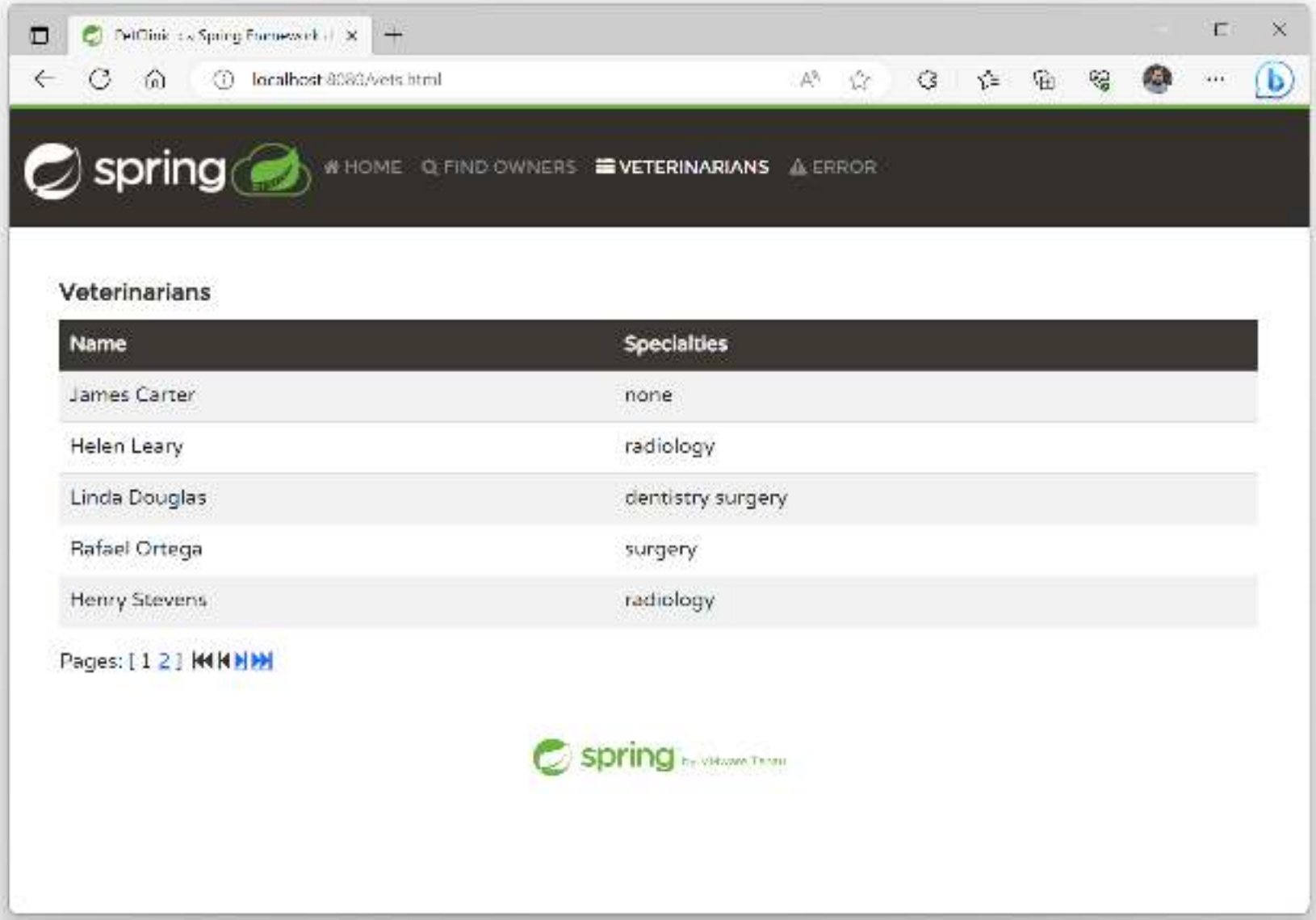
Agenda

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


localhost:8080
spring
HOME
FIND OWNERS
VETERINARIANS
ERROR
Welcome
spring


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

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 3.1.0
- Initialize SQL Scripts

Examination

- Build Time
- Startup Time
- Resource Usage
- Container Image Size
- Throughput


paketo
buildpacks

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.

See also: https://buildpacks.io/

```
Setting Active Processor Count to 4
Calculated JVM Memory Configuration: -XX:MaxDirectMemorySize=10M -Xmx418600K -XX:MaxMetaspaceSize=117975K -XX:ReservedCodeCacheSize=240M -Xss1M
(Total Memory: 1G, Thread Count: 250, Loaded Class Count: 18415, 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 -Xmx418600K -XX:MaxMetaspaceSize=117975K -XX:ReservedCodeCacheSize=240M -Xss1M
-XX:+UnlockDiagnosticVMOptions -XX:NativeMemoryTracking=summary -XX:+PrintNMTStatistics -Dorg.springframework.cloud.bindings.boot.enable=true


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

:: Built with Spring Boot :: 3.1.0


2023-06-19T19:55:38.079Z  INFO 1 --- [           main] o.s.s.petclinic.PetClinicApplication     : Starting PetClinicApplication v3.1.0-SNAPSHOT using Java 17.0.7 with PID 1
(/workspace/BOOT-INF/classes started by cnb in /workspace)
2023-06-19T19:55:38.083Z  INFO 1 --- [           main] o.s.s.petclinic.PetClinicApplication     : No active profile set, falling back to 1 default profile: "default"
2023-06-19T19:55:38.923Z  INFO 1 --- [           main] .s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data JPA repositories in DEFAULT mode.
2023-06-19T19:55:38.968Z  INFO 1 --- [           main] .s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository scanning in 39 ms. Found 2 JPA repository interfaces.
2023-06-19T19:55:39.483Z  INFO 1 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port(s): 8080 (http)
2023-06-19T19:55:39.490Z  INFO 1 --- [           main] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
2023-06-19T19:55:39.490Z  INFO 1 --- [           main] o.apache.catalina.core.StandardEngine    : Starting Servlet engine: [Apache Tomcat/10.1.8]
2023-06-19T19:55:39.566Z  INFO 1 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
2023-06-19T19:55:39.567Z  INFO 1 --- [           main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 1439 ms
2023-06-19T19:55:39.733Z  INFO 1 --- [           main] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Starting...
2023-06-19T19:55:39.896Z  INFO 1 --- [           main] com.zaxxer.hikari.pool.HikariPool        : HikariPool-1 - Added connection conn0: url=jdbc:h2:mem:1d5c8bbd-0c1a-4bf3-9548-e4396bd914ea user=SA
2023-06-19T19:55:39.898Z  INFO 1 --- [           main] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Start completed.
2023-06-19T19:55:40.020Z  INFO 1 --- [           main] o.hibernate.jpa.internal.util.LogHelper  : HHH000204: Processing PersistenceUnitInfo [name: default]
2023-06-19T19:55:40.076Z  INFO 1 --- [           main] org.hibernate.Version                    : HHH000412: Hibernate ORM core version 6.2.2.Final
2023-06-19T19:55:40.078Z  INFO 1 --- [           main] org.hibernate.cfg.Environment            : HHH000406: Using bytecode reflection optimizer
2023-06-19T19:55:40.187Z  INFO 1 --- [           main] o.h.b.i.BytecodeProviderInitiator        : HHH000021: Bytecode provider name : bytebuddy
2023-06-19T19:55:40.306Z  INFO 1 --- [           main] o.s.o.j.p.SpringPersistenceUnitInfo      : No LoadTimeWeaver setup: ignoring JPA class transformer
2023-06-19T19:55:40.349Z  INFO 1 --- [           main] org.hibernate.orm.dialect                : HHH035001: Using dialect: org.hibernate.dialect.H2Dialect, version: 2.1.214
2023-06-19T19:55:40.521Z  INFO 1 --- [           main] o.h.b.i.BytecodeProviderInitiator        : HHH000021: Bytecode provider name : bytebuddy
2023-06-19T19:55:41.127Z  INFO 1 --- [           main] o.h.e.t.j.p.i.JtaPlatformInitiator       : HHH000490: Using JtaPlatform implementation: [org.hibernate.engine.transaction.jta.platform.internal.NoJtaPlatform]
2023-06-19T19:55:41.129Z  INFO 1 --- [           main] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persistence unit 'default'
2023-06-19T19:55:41.310Z  INFO 1 --- [           main] o.s.d.j.r.query.QueryEnhancerFactory     : Hibernate is in classpath; If applicable, HQL parser will be used.
2023-06-19T19:55:42.097Z  INFO 1 --- [           main] o.s.b.a.e.web.EndpointLinksResolver      : Exposing 13 endpoint(s) beneath base path '/actuator'
2023-06-19T19:55:42.180Z  INFO 1 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port(s): 8080 (http) with context path ''
2023-06-19T19:55:42.195Z  INFO 1 --- [           main] o.s.s.petclinic.PetClinicApplication     : Started PetClinicApplication in 4.463 seconds (process running for 4.92)
```

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

```
<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

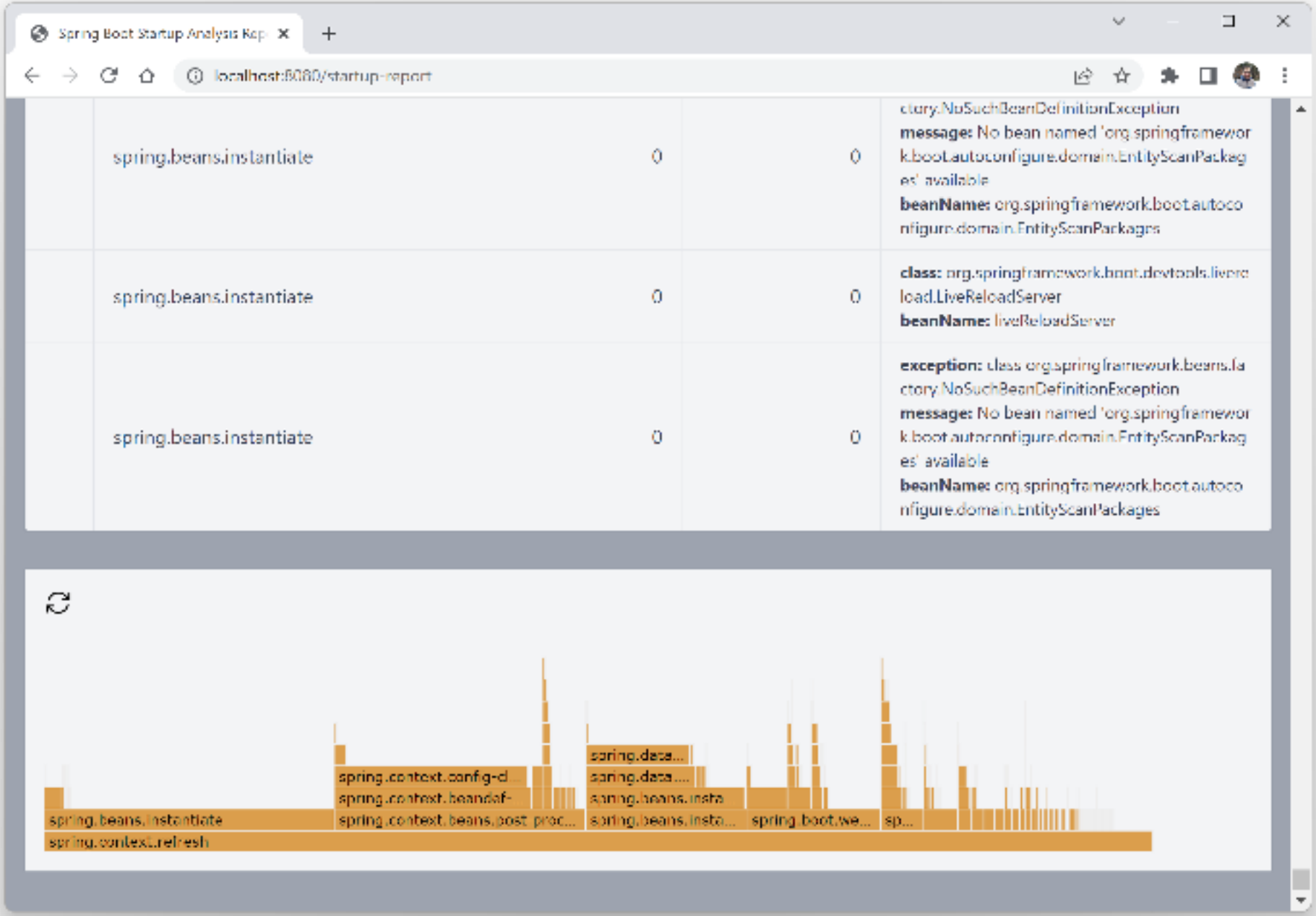
localhost:8080/startup-report

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.beans.instantiate
spring.context.refresh
spring.context.beans.post.proc...
spring.boot.we...

Benchmarks

Benchmarks

- Build
  - Maven build time
  - Artifact size
- Startup
  - Startup time
  - Initial memory usage
- Throughput & Latency
  - wrk2 -t4 -c200 -d30s -R2000 --latency
  - 2 min warmup, 30 sec 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 17.0.7-tem

mvn spring-boot:build-image

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

Image Size	Build	RAM	Startup	Throughput	Latency 99%	Latency 99.9%	Latency 99.99%
⏱ 315MB	30.2s	282MB	4.117s	1986/s	37.578ms	57.343ms	83.785ms

No Optimizing - Baseline JRE 20

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

```
sdk use java 20.0.1-tem

mvn -Djava.version=20 spring-boot:build-image \
    -Dspring-boot.build-image.imageName=spring-petclinic:3.1.0-SNAPSHOT-jdk20

docker run -p 8080:8080 -t spring-petclinic:3.1.0-SNAPSHOT-jdk20
```

Image Size	Build	RAM	Startup	Throughput	Latency 99%	Latency 99.9%	Latency 99.99%
⏱ 315MB	30.2s	282MB	4.117s	1986/s	37.578ms	57.343ms	83.785ms
351MB	28.2s	284MB	3.997s	1976/s	35.042ms	100.136ms	132.162ms

-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.

In order to disable C2 and only use 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:3.1.0-SNAPSHOT
```

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

Image Size	Build	RAM	Startup	Throughput	Latency 99%	Latency 99.9%	Latency 99.99%
⏱ 315MB	30.2s	282MB	4.117s	1986/s	37.578ms	57.343ms	83.785ms
-	-	201MB	3.458s	1453/s	9618ms	10556ms	10958ms

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>
    <option
</dependenc
```

Image Size	Build	RAM	Startup	Throughput	Latency 99%	Latency 99.9%	Latency 99.99%
⏱ 315MB	30.2s	282MB	4.117s	1986/s	37.578ms	57.343ms	83.785ms
315MB	28.0s	283MB	3.984s	1988/s	38.338ms	66.992ms	90.725ms

```
sdk use java 17.0.7-tem

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:3.1.0-SNAPSHOT
```

Image Size	Build	RAM	Startup	Throughput	Latency 99%	Latency 99.9%	Latency 99.99%
⏱ 315MB	30.2s	282MB	4.117s	1986/s	37.578ms	57.343ms	83.785ms
-	-	242MB	2.212s	1990/s	33.396ms	54.514ms	73.740ms

Lazy Spring Beans (2)

Pros

- Faster startup usefull 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 missconfigurations 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 17.0.7-tem

mvn spring-boot:build-image

docker run -p 8080:8080 -t spring-petclinic-no-actuator:3.1.0-SNAPSHOT
```

Image Size	Build	RAM	Startup	Throughput	Latency 99%	Latency 99.9%	Latency 99.99%
⏱ 315MB	30.2s	282MB	4.117s	1986/s	37.578ms	57.343ms	83.785ms
313MB	27s	263MB	3.473s	1979/s	26.642ms	44.960ms	71.448ms

Fixing Spring Boot Config Location

Fixing Spring Boot Config Location

Fix the location of the Spring Boot config file(s).

Considered in 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:3.1.0-SNAPSHOT
```

Image Size	Build	RAM	Startup	Throughput	Latency 99%	Latency 99.9%	Latency 99.99%
⏱ 315MB	30.2s	282MB	4.117s	1986/s	37.578ms	57.343ms	83.785ms
-	-	292MB	3.987s	1993/s	35.994ms	54.898ms	78.090ms

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:3.1.0-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.

```
<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 [7]:
        com.h2database:h2:2.1.214:runtime (2 MB)
        com.mysql:mysql-connector-j:8.0.33:runtime (2 MB)
        org.postgresql:postgresql:42.6.0:runtime (1 MB)
        com.github.ben-manes.caffeine:caffeine:3.1.6:compile (734 KB)
        jakarta.xml.bind:jakarta.xml.bind-api:4.0.0:compile (124 KB)
        javax.cache:cache-api:1.1.1:compile (50 KB)
        org.springframework.boot:spring-boot-starter-test:3.1.0:test (4 KB)
USED TRANSITIVE DEPENDENCIES [69]:
        net.bytebuddy:byte-buddy:1.14.4:runtime (3 MB)
        org.apache.tomcat.embed:tomcat-embed-core:10.1.8:compile (3 MB)
        org.aspectj:aspectjweaver:1.9.19:compile (1 MB)
        org.springframework:spring-core:6.0.9:compile (1 MB)
        org.springframework.boot:spring-boot-autoconfigure:3.1.0:compile (1 MB)
        org.springframework:spring-web:6.0.9:compile (1 MB)
        ...
USED INHERITED DIRECT DEPENDENCIES [0]:
USED INHERITED TRANSITIVE DEPENDENCIES [0]:
POTENTIALLY UNUSED DIRECT DEPENDENCIES [9]:
        org.webjars.npm:bootstrap:5.2.3:compile (1 MB)
        org.webjars.npm:font-awesome:4.7.0:compile (665 KB)
        org.springframework.boot:spring-boot-devtools:3.1.0:compile (228 KB)
        org.springframework.boot:spring-boot-starter-actuator:3.1.0:compile (4 KB)
        ...
POTENTIALLY UNUSED TRANSITIVE DEPENDENCIES [28]:
        org.hibernate.orm:hibernate-core:6.2.2.Final:compile (10 MB)
        com.fasterxml.jackson.core:jackson-databind:2.15.0:compile (1 MB)
        org.hibernate.validator:hibernate-validator:8.0.0.Final:compile (1 MB)
        org.thymeleaf:thymeleaf:3.1.1.RELEASE:compile (915 KB)
        io.micrometer:micrometer-core:1.11.0:compile (845 KB)
        org.springframework.boot:spring-boot-actuator-autoconfigure:3.1.0:compile (686 KB)
        ...
POTENTIALLY UNUSED INHERITED DIRECT DEPENDENCIES [0]:
POTENTIALLY UNUSED INHERITED TRANSITIVE DEPENDENCIES [0]:
[INFO] Analysis done in 0min 7s
```

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 17.0.7-tem

mvn spring-boot:build-image

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

Image Size	Build	RAM	Startup	Throughput	Latency 99%	Latency 99.9%	Latency 99.99%
⏱ 315MB	30.2s	282MB	4.117s	1986/s	37.578ms	57.343ms	83.785ms
307MB	33.6s	259MB	3.326s	1988/s	28.486ms	43.474ms	60.032ms

Ahead-of-Time Processing (AOT)

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

Spring AOT is a process that analyzes your application at build-time and generate an optimized 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) (1)

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

```
sdk use java 22.3.r17-grl

mvn spring-boot:build-image

docker run -e spring.aot.enabled=true -p 8080:8080 -t spring-petclinic-aot:3.1.0-SNAPSHOT
```

Image Size	Build	RAM	Startup	Throughput	Latency 99%	Latency 99.9%	Latency 99.99%
⏱ 315MB	30.2s	282MB	4.117s	1986/s	37.578ms	57.343ms	83.785ms
317MB	31.9s	291MB	4.013s	1990/s	30.104ms	46.820ms	67.300ms

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)

```
<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 17.0.7-tem

mvn spring-boot:build-image

docker run -p 8080:8080 -t spring-petclinic-jlink:3.1.0-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>
```

Image Size	Build	RAM	Startup	Throughput	Latency 99%	Latency 99.9%	Latency 99.99%
⏱ 315MB	30.2s	282MB	4.117s	1986/s	37.578ms	57.343ms	83.785ms
236MB	36.4s	282MB	4.047s	1990/s	32.832ms	60.216ms	89.636ms

I ❤️

Spring Boot 🍃 & Buildpacks

Eclipse OpenJ9

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

Read performance details

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>
```

Image Size	Build	RAM	Startup	Throughput	Latency 99%	Latency 99.9%	Latency 99.99%
⏱ 315MB	30.2s	282MB	4.117s	1986/s	37.578ms	57.343ms	83.785ms
305MB	35.8s	165MB	6.952s	1976/s	46.708ms	82.330ms	136.140ms

```
sdk use jav

mvn spring-boot:build-image

docker run -p 8080:8080 -t spring-petclinic-custom-jvm-openj9:3.1.0-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:3.1.0-SNAPSHOT
```

Image Size	Build	RAM	Startup	Throughput	Latency 99%	Latency 99.9%	Latency 99.99%
⏱ 315MB	30.2s	282MB	4.117s	1986/s	37.578ms	57.343ms	83.785ms
305MB	35.8s	160MB	5.272s	1258/s	12112ms	13166ms	13816ms

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>
```

Image Size	Build	RAM	Startup	Throughput	Latency 99%	Latency 99.9%	Latency 99.99%
⏱ 315MB	30.2s	282MB	4.117s	1986/s	37.578ms	57.343ms	83.785ms
649MB	38.3s	237MB	3.86s	1988/s	26.564ms	46.36ms	66.202ms

```
sdk use jav

mvn spring-boot:build-image

docker run -p 8080:8080 -t spring-petclinic-custom-jvm-graalvm:3.1.0-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:3.1.0-SNAPSHOT
```

Image Size	Build	RAM	Startup	Throughput	Latency 99%	Latency 99.9%	Latency 99.99%
⏱ 315MB	30.2s	282MB	4.117s	1986/s	37.578ms	57.343ms	83.785ms
202MB	295s	218MB	0.275s	1990/s	114.93ms	203.162ms	282.392ms

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-3.1.0.jar
```

```
jcmd target/spring-petclinic-crac-3.1.0.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
  - Only patched Tomcat 10.1.7 available
- Does not work on Windows or on macOS
  - But Ubuntu 20.04 LTS and also WSL2
- 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
  - 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 funcutional bean definitions
- More JVM tunining

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, ...
- 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 17.0.7-tem

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:3.1.0-SNAPSHOT
```

Image Size	Build	RAM	Startup	Throughput	Latency 99%	Latency 99.9%	Latency 99.99%
⏱ 315MB	30.2s	282MB	4.117s	1986/s	37.578ms	57.343ms	83.785ms
227MB	48.8s	235MB	1.697s	1985/s	30.476ms	54.124ms	73.540ms

Did I miss something? 🧐

Let me/us know! 🙋

… or not! 🙊

Lean Spring Boot

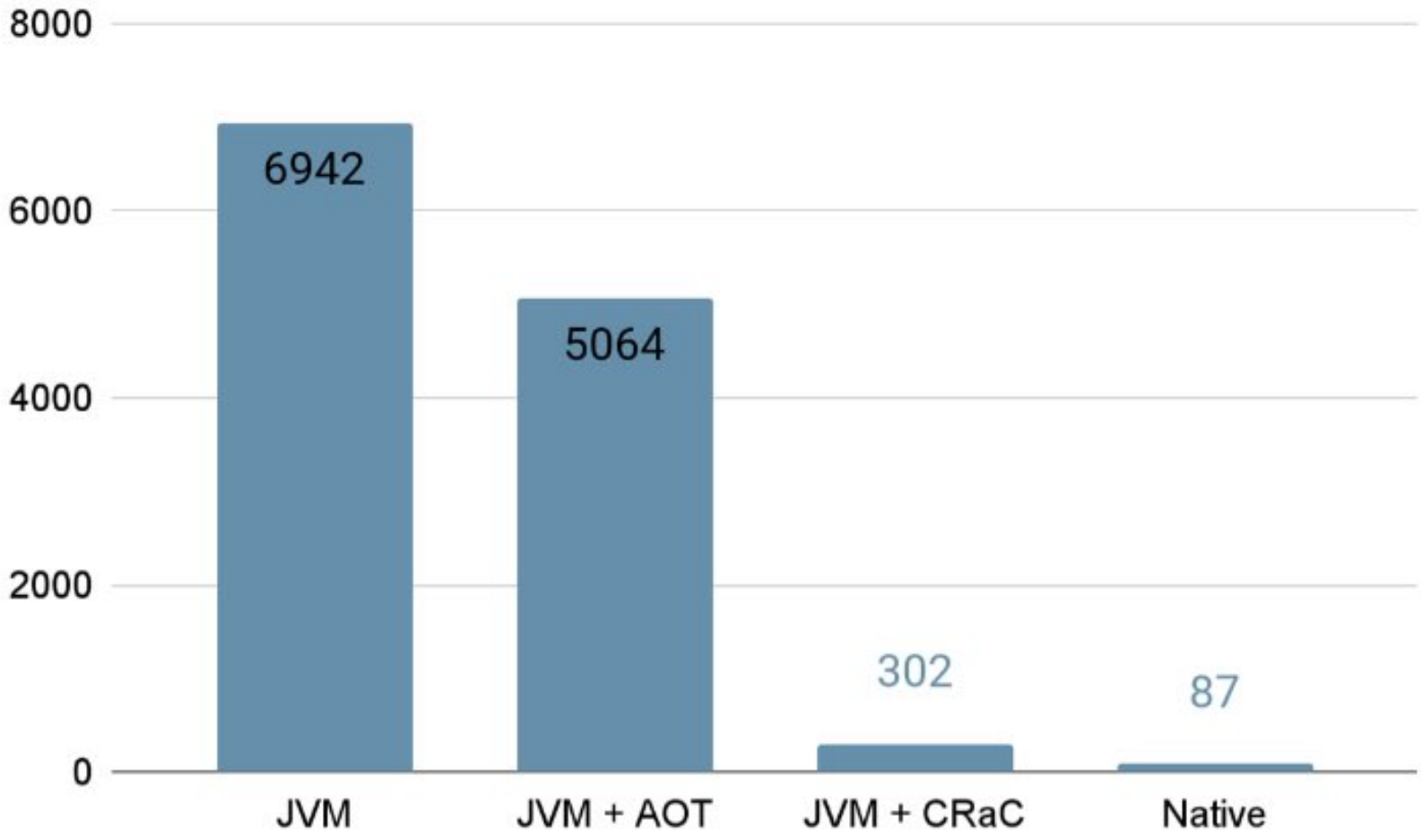
Applications for The Cloud

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