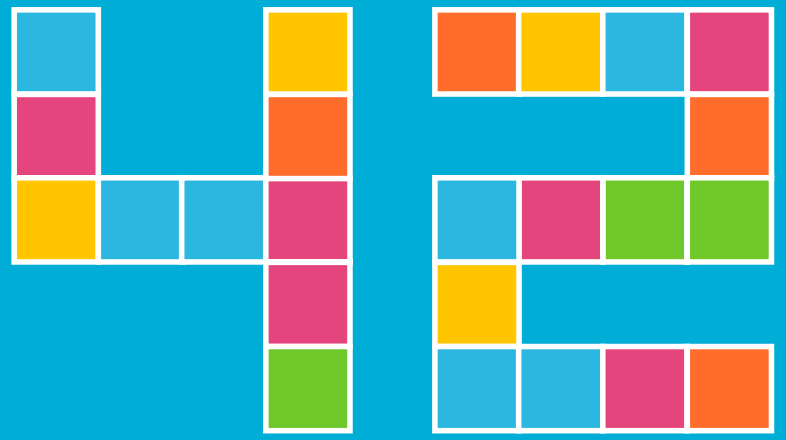
Lean Spring Boot

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-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

Lean Spring Boot

Applications for The Cloud

Patrick Baumgartner

42talents GmbH, Zürich, Schweiz

⚠ 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

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

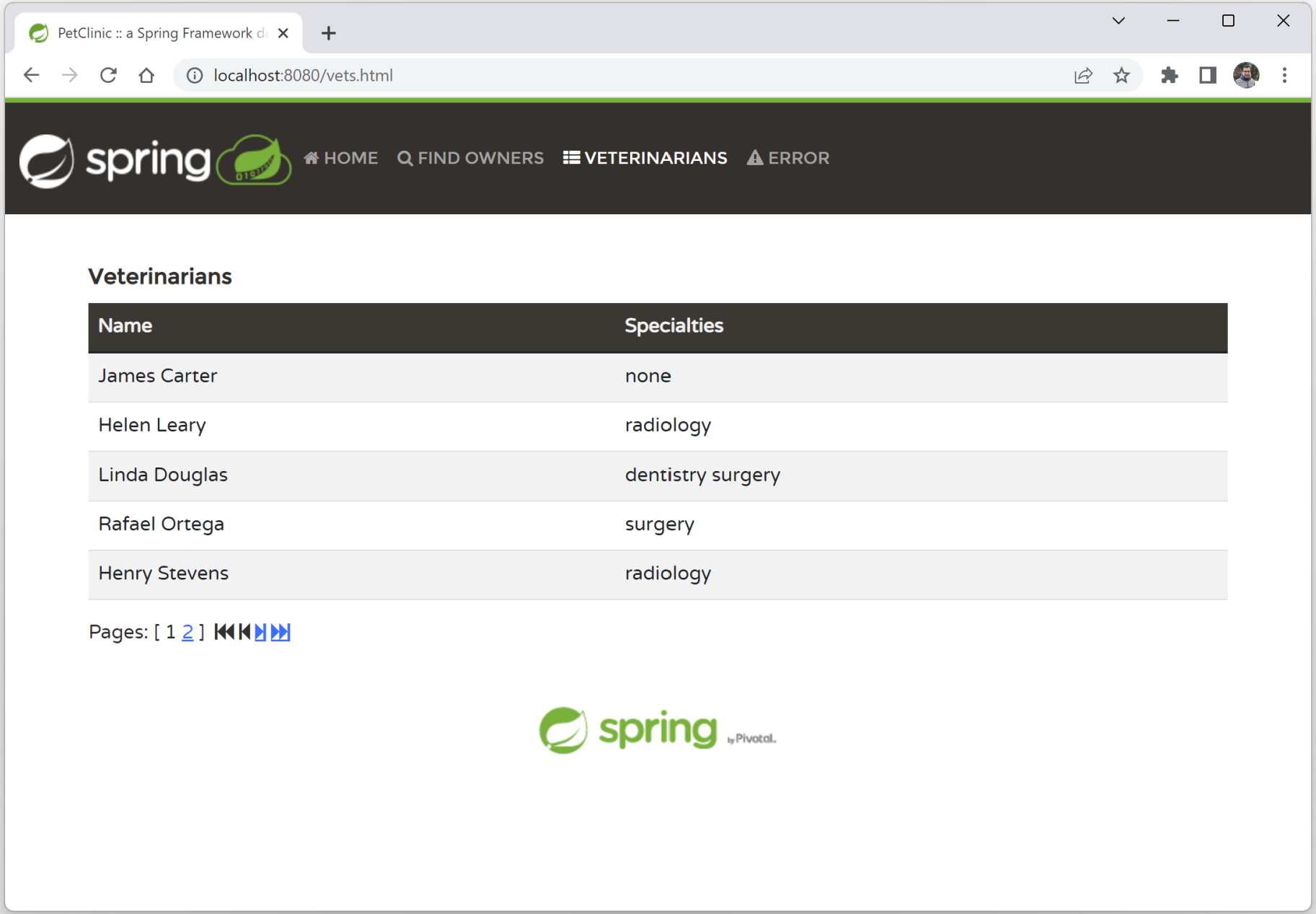
PetClinic :: a Spring Framework d

localhost:8080

spring HOME FIND OWNERS VETERINARIANS ERROR

Welcome

spring by Pivotal


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 Pivotal.

Spring Petclinic Community

- spring-framework-petclinic
- spring-petclinic-angular
- 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

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 3.0.6
- 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/

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

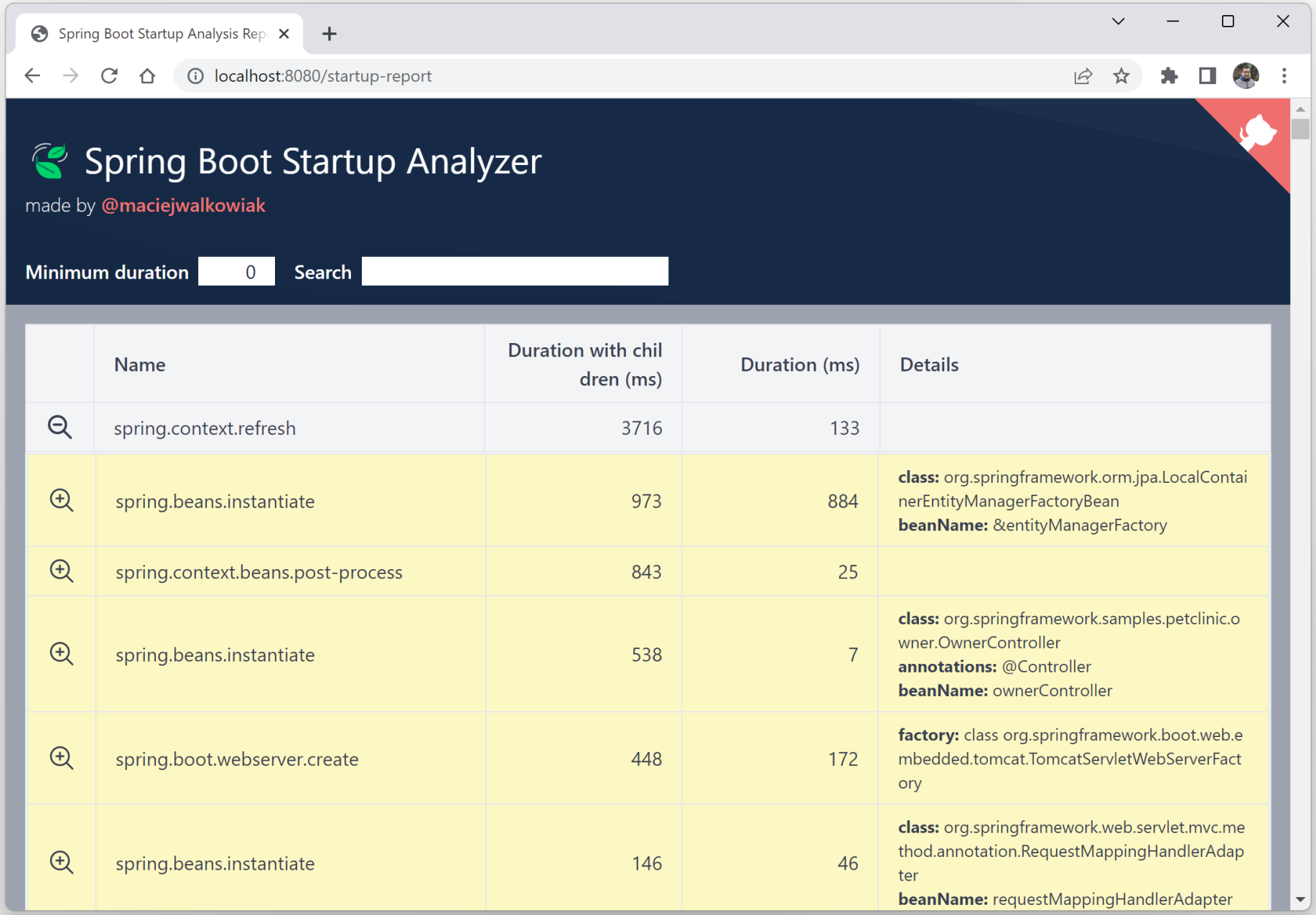
Spring Boot Startup Report

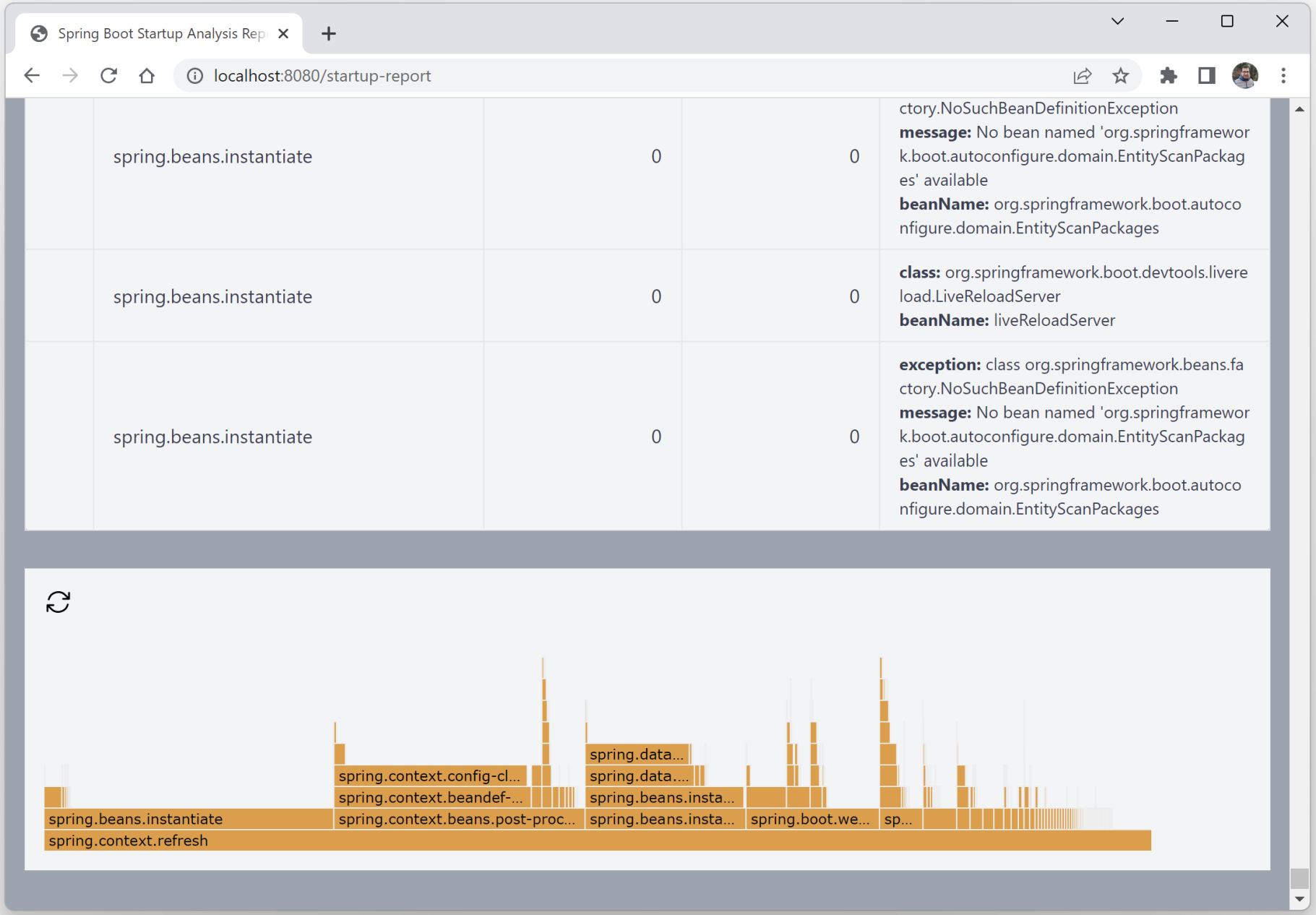
By Maciej Walkowiak

- 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.data...
spring.context.config-cl...
spring.data....
spring.context.beandef-...
spring.beans.insta...
spring.beans.instantiate
spring.context.beans.post-proc...
spring.beans.insta...
spring.boot.we...
sp...
spring.context.refresh

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:3.0.0-SNAPSHOT
```

Image Size	Build	RAM	Startup #1	Startup #2	Startup #3	Throughput
⏱️ 316MB	26s	492MB	4.002s	3.775s	3.888s	355.5/s

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:3.0.0-SNAPSHOT-jdk20

docker run -p 8080:8080 -t spring-petclinic:3.0.0-SNAPSHOT-jdk20
```

Image Size	Build	RAM	Startup #1	Startup #2	Startup #3	Throughput
⏱️ 316MB	26s	492MB	4.002s	3.775s	3.888s	355.5/s
352MB	25s	497MB	3.617s	3.812s	3.763s	359.1/s

-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:3.0.0-SNAPSHOT
```

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

Image Size	Build	RAM	Startup #1	Startup #2	Startup #3	Throughput
⏱ 316MB	26s	492MB	4.002s	3.775s	3.888s	355.5/s
-	-	367MB	3.228s	3.270s	3.477s	197.4/s

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-conte
    <scope>optional</scope>
</dependency>
```

Image Size	Build	RAM	Startup #1	Startup #2	Startup #3	Throughput
⏱ 316MB	26s	492MB	4.002s	3.775s	3.888s	355.5/s
316MB	27s	501MB	3.805s	3.678s	3.689s	348.8/s

```
sdk use java 17.0.7-tem

mvn spring-boot:build-image

docker run -p 8080:8080 -t spring-petclinic-indexer:3.0.0-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 \
  -t spring-petclinic:3.0.0-SNAPSHOT
```

Image Size	Build	RAM	Startup #1	Startup #2	Startup #3	Throughput
⏱ 316MB	26s	492MB	4.002s	3.775s	3.888s	355.5/s
-	-	525MB	3.404s	3.814s	3.653s	341.7/s

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: 426
  - Spring Pet Clinic no Actuators: 264 🔥

```
sdk use java 17.0.7-tem

mvn spring-boot:build-image

docker run -p 8080:8080 -t spring-petclinic-no-actuator:3.0.0-SNAPSHOT
```

Image Size	Build	RAM	Startup #1	Startup #2	Startup #3	Throughput
⏱ 316MB	26s	492MB	4.002s	3.775s	3.888s	355.5/s
314MB	25s	481MB	3.292s	3.280s	3.489s	370.1/s

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

See also: https://docs.spring.io/spring-boot/docs/current/reference/html/features.html#features.external-config

```
docker run -p 8080:8080 -e spring.config.location=classpath:application.properties \
  -t spring-petclinic:3.0.0-SNAPSHOT
```

Image Size	Build	RAM	Startup #1	Startup #2	Startup #3	Throughput
⏱ 316MB	26s	492MB	4.002s	3.775s	3.888s	355.5/s
-	-	493MB	3.830s	3.983s	3.814s	353.9/s

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=fals
  -t spring-petclinic:3.0.0
```

Image Size	Build	RAM	Startup #1	Startup #2	Startup #3	Throughput
⏱ 316MB	26s	492MB	4.002s	3.775s	3.888s	355.5/s
-	-	-	-	-	-	-

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 [4]:
        com.h2database:h2:2.1.214:runtime (2 MB)
        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:3.0.6:test (4 KB)
USED TRANSITIVE DEPENDENCIES [47]:
        net.bytebuddy:byte-buddy:1.12.23:runtime (3 MB)
        org.springframework:spring-core:6.0.8:compile (1 MB)
        org.springframework:spring-web:6.0.8:compile (1 MB)
        org.springframework.boot:spring-boot-autoconfigure:3.0.6:compile (1 MB)
        org.springframework.boot:spring-boot:3.0.6:compile (1 MB)
        ...
USED INHERITED DIRECT DEPENDENCIES [0]:
USED INHERITED TRANSITIVE DEPENDENCIES [0]:
POTENTIALLY UNUSED DIRECT DEPENDENCIES [10]:
        org.webjars.npm:bootstrap:5.2.3:compile (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)
        org.springframework.boot:spring-boot-devtools:3.0.6:compile (228 KB)
        org.springframework.boot:spring-boot-starter-web:3.0.6:compile (4 KB)
        org.springframework.boot:spring-boot-starter-thymeleaf:3.0.6:compile (4 KB)
        ...
POTENTIALLY UNUSED TRANSITIVE DEPENDENCIES [36]:
        org.hibernate.orm:hibernate-core:6.1.7.Final:compile (9 MB)
        org.apache.tomcat.embed:tomcat-embed-core:10.1.8:compile (3 MB)
        org.aspectj:aspectjweaver:1.9.19: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)
        ...
POTENTIALLY UNUSED INHERITED DIRECT DEPENDENCIES [0]:
POTENTIALLY UNUSED INHERITED TRANSITIVE DEPENDENCIES [0]:
[INFO] Analysis done in 0min 6s
```

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:3.0.0-SNAPSHOT
```

Image Size	Build	RAM	Startup #1	Startup #2	Startup #3	Throughput
⏱ 316MB	26s	492MB	4.002s	3.775s	3.888s	355.5/s
309MB	31s	487MB	3.149s	3.316s	3.190s	355.4/s

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>
```

Image Size	Build	RAM	Startup #1	Startup #2	Startup #3	Throughput
⏱ 316MB	26s	492MB	4.002s	3.775s	3.888s	355.5/s
249MB	41s	489MB	3.880s	4.008s	3.796s	355.1/s

```
sdk use java 17.0.7-tem

mvn spring-boot:build-image

docker run -p 8080:8080 -t spring-petclinic-jlink:3.0.0-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 #1	Startup #2	Startup #3	Throughput
⏱ 316MB	26s	492MB	4.002s	3.775s	3.888s	355.5/s
236MB	36s	500MB	3.854s	3.755s	3.901s	343.6/s

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

Read performance details

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>
```

Image Size	Build	RAM	Startup #1	Startup #2	Startup #3	Throughput
⏱ 316MB	26s	492MB	4.002s	3.775s	3.888s	355.5/s
305MB	27s	188MB	6.474s	6.478s	6.557s	355.5/s

```
sdk use java 17.0.7-tem

mvn spring-boot:build-image

docker run -p 8080:8080 -t spring-petclinic-custom-jvm-openj9:3.0.0-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:3.0.0-SNAPSHOT
```

Image Size	Build	RAM	Startup #1	Startup #2	Startup #3	Throughput
⏱ 316MB	26s	492MB	4.002s	3.775s	3.888s	355.5/s
305MB	27s	365MB	5.089s	5.151s	5.107s	161.5/s

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>
```

Image Size	Build	RAM	Startup #1	Startup #2	Startup #3	Throughput
⏱ 316MB	26s	492MB	4.002s	3.775s	3.888s	355.5/s
650MB	39s	451MB	3.728s	3.741s	3.788s	361.5/s

```
sdk use java 17.0.7-tem

mvn spring-boot:build-image

docker run -p 8080:8080 -t spring-petclinic-custom-jvm-graalvm:3.0.0-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:3.0.0-SNAPSHOT
```

Image Size	Build	RAM	Startup #1	Startup #2	Startup #3	Throughput
⏱ 316MB	26s	492MB	4.002s	3.775s	3.888s	355.5/s
202MB	344s	262MB	0.207s	0.207s	0.199s	193.7/s

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-3.0.6.jar
```

```
jcmd target/spring-petclinic-crac-3.0.6.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.

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, ...

A Few Simple Optimizations Applied

A Few Simple Optimizations Applied

- Dependency Cleanup
  - DB Drivers, Spring Boot Actuator, Jackson, Tomcat Websocket, ...
- JVM Parameters
- JLink

Image Size	Build	RAM	Startup #1	Startup #2	Startup #3	Throughput
⏱ 316MB	26s	492MB	4.002s	3.775s	3.888s	355.5/s
228MB	42s	221MB	2.591s	2.673s	2.570s	244.9/s

```
sdk use java 17.0.7-tem

mvn spring-boot:build-image

docker run -p 8080:8080 \
  -e "JAVA_TOOL_OPTIONS=-XX:TieredStopAtLevel=1" \
  -e spring.main.lazy-initialization=true \
  -e spring.config.location=classpath:application.properties \
  -t spring-petclinic-optimized:3.0.0-SNAPSHOT
```

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