How to optimize your Spring Boot App ?

Xavier Bouclet

- ~17 years of IT experience
- Developer / Technical Leader
- Co-organizer of the Montreal JUG & Devoxx4Kids Quebec
- Blogger https://xavierbouclet.com
- Twitter @XavierBouclet

How to optimize your Spring Boot App?

Elevate your Spring Boot application performances! Join me for an insightful presentation on performance enhancement tips. Don't miss this opportunity to optimize your Spring Boot apps for peak performance. Reserve your spot today!

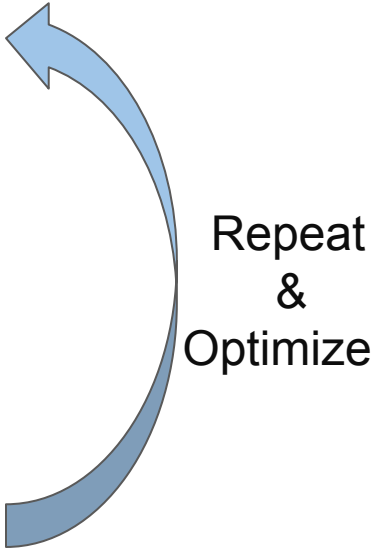
Performance Tests

Perform regular load and performance testing to identify and resolve performance issues

- JMeter (Java)
- Gatling (Scala)
- Locust (Python)
- K6 (Go)
- Artillery (Node)
- …

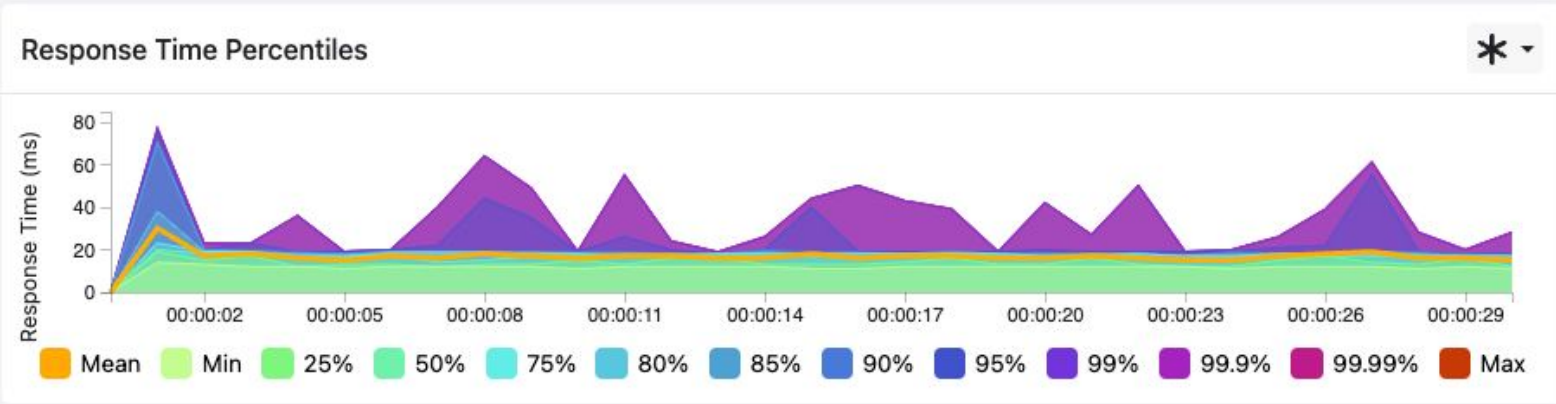
Performance Tests - strategy

- Define performance goals
- Create test scenarios
- Configure **prod*~~test~~*** environment
- Execute tests
- Analyse results



Performance Tests - analyse

- pXX : Means XX % of the request are handled in a time equals or less than the time at pXX

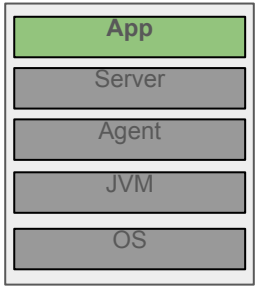


credits:https://gatling.io/2023/06/latency-percentiles-for-load-testing-analysis/

Analyse the JVM

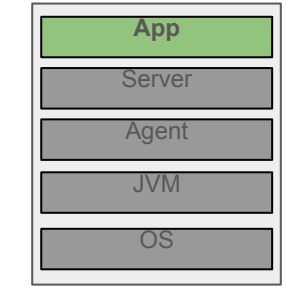
- Profiling
  - VisualVM, YourKit, Java Flight Recorder, Java Mission Control, …
- Enabling JVM Logs
  - `-verbose:gc`, `-XX:+PrintGCDetails`, & `-XX:+PrintGCApplicationStoppedTime`
- Analyse Memory Dump
- …

Actuator



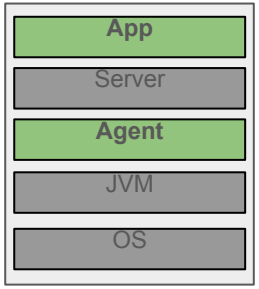
- Heap Dump, Thread Dump
- Change logs level
- See content of specific log file
- Exposes metrics (Prometheus)
- Specific Metrics

Actuator - Micrometer Specific Metric



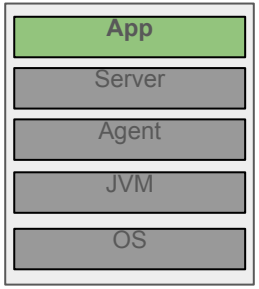

```java
import io.micrometer.core.instrument.Counter;
import io.micrometer.core.instrument.MeterRegistry;
import org.springframework.stereotype.Component;

@Component
public class MyCustomMetrics {

    private final Counter myEventCounter;

    public MyCustomMetrics(MeterRegistry registry) {
        myEventCounter = Counter.builder("name.of.the.metric")
                .description("Description of the metric")
                .tags("tag1", "value1")
                .register(registry);
    }

    public void incrementEventCounter() {
        myEventCounter.increment();
    }
}
```

Metrics & Monitoring



- Have the right picture of your production
  - Micrometer
  - Prometheus
  - Java Agent :
    - Open Telemetry
    - Datadog
    - New Relic
  - …

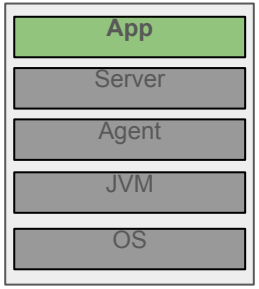
Devtools



- Optimize your local devtime
- Allow remote debugging

```
<build>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
            <configuration>
                <excludeDevtools>false</excludeDevtools>
            </configuration>
        </plugin>
    </plugins>
</build>
```

```
spring:
  devtools:
    remote:
      debug:
        local-port: 8010
```



Profiles

- Use profiles
  - dev, non-prod, prod
  - test

- application-dev.yml

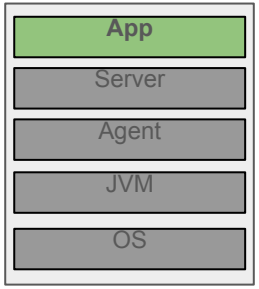
```
app:
  refresh-rate: 5s
```

- application-prod.yml

```
app:
  refresh-rate: 1h
```

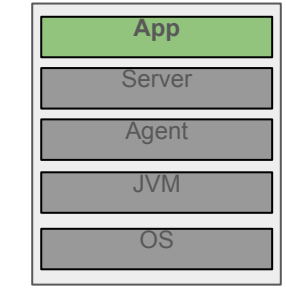
Lazy initialization

- Improve startup time
- May reduce first time response




```yaml
spring:
  main:
    lazy-initialization: true
```

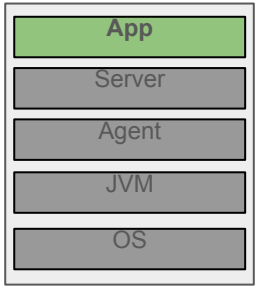
```java
@Configuration
public class MyConfiguration {

    @Bean
    @Lazy
    public MyBean myBean() {
        return new MyBean();
    }
}
```



Optimize Spring Configuration

- Exclude useless AutoConfiguration
- Avoid class path scanning

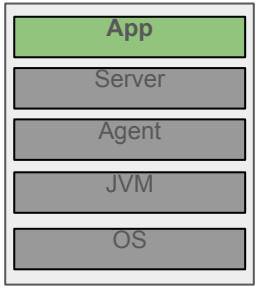
```
spring:
  autoconfigure:
    exclude:
      - org.springframework.boot.autoconfigure.security.servlet.SecurityAutoConfiguration
      - org.springframework.boot.autoconfigure.h2.H2ConsoleAutoConfiguration
```



Avoid (some) Magic

- Declare your beans improve startup time vs @ComponentScan
- Prefer constructor injection vs @Autowired and @Value
- Use @Configuration(proxyBeanMethods = false) when possible

Log optimization



- Use asynchronous logging (Log4j2, Logback)
- Use the appropriate log level
- Be careful with concatenation
- Don’t use logs 😅

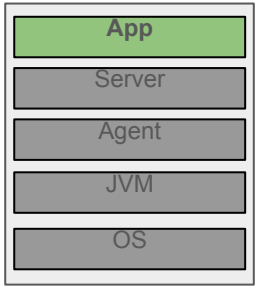
- application-dev.yml

```yaml
logging:
  level:
    root: DEBUG
    org.springframework.web: DEBUG
    org.confoo: DEBUG
```

- application-prod.yml

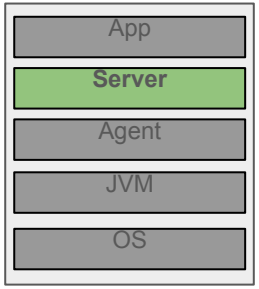
```yaml
logging:
  level:
    root: WARN
    org.springframework.web: WARN
    org.confoo: INFO
```

Effective database management



- spring-data-jdbc over spring-data-jpa
- Use reactive alternative
- Optimize your sql or no-sql requests
- Use caching on frequent calls
- Fine tune thread pool

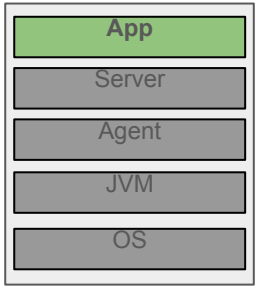
Request & Response Compression



- Can improve performance
- Faster data transfer times
- Reduced bandwidth consumption
- Lower latency

```yaml
server:
  compression:
    enabled: true
    min-response-size: 1024
    mime-types:
      - text/html
      - text/xml
      - text/plain
      - application/json
      - application/javascript
      - text/css
    # Add other content if needed
```

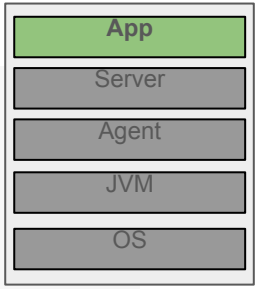
Dependency management



- Avoid useless dependencies
- Upgrade your dependencies
- Delegate your dependencies

Circuit breakers

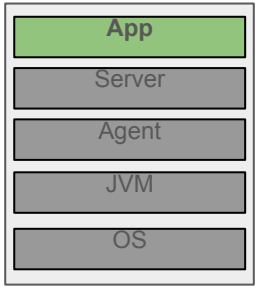
- Reduce latency when a service is failing
- Improve recovery time




```yaml
resilience4j.circuitbreaker:
  instances:
    myCircuitBreaker:
      registerHealthIndicator: true
      slidingWindowSize: 100
      minimumNumberOfCalls: 10
      permittedNumberOfCallsInHalfOpenState: 3
      automaticTransitionFromOpenToHalfOpenEnabled: true
      waitDurationInOpenState: 10s
      failureRateThreshold: 50
      eventConsumerBufferSize: 10
```

```xml
<dependencies>
    <!-- Spring Cloud Starter Circuit Breaker Resilience4J -->
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-circuitbreaker-resilience4j</artifactId>
    </dependency>
</dependencies>
```

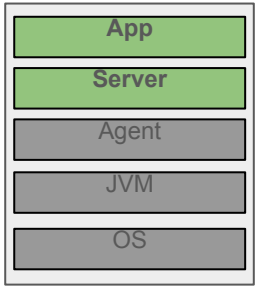
```java
import io.github.resilience4j.circuitbreaker.annotation.CircuitBreaker;
import org.springframework.stereotype.Service;

@Service
public class MyService {

    @CircuitBreaker(name = "myCircuitBreaker", fallbackMethod = "fallbackMethod")
    public String someMethod() {
        // Logic that might fail
    }

    public String fallbackMethod(Exception ex) {
        // Fallback logic
        return "Fallback response";
    }
}
```



Asynchronous and multithreading and Reactive Programming

- Webflux
- R2dbc
- Coroutines (Kotlin)
- Annotation : `org.springframework.scheduling.annotation.Async`
- ...

Threads pool



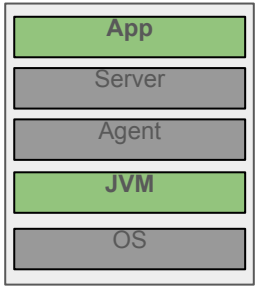
- Performance Optimization
- Scalability Improvement
- System Resources Management
- Environment-Specific Considerations
- Configuration Best Practices

```yaml
#Tomcat
server:
  tomcat:
    max-threads: 200
    min-spare-threads: 10
```

```yaml
#Jetty
server:
  jetty:
    max-threads: 200
    min-threads: 10
```

```yaml
#Undertow
server:
  undertow:
    io-threads: 10
    worker-threads: 200
```

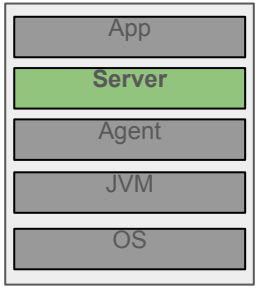
Virtual thread (Loom project - Java 21)

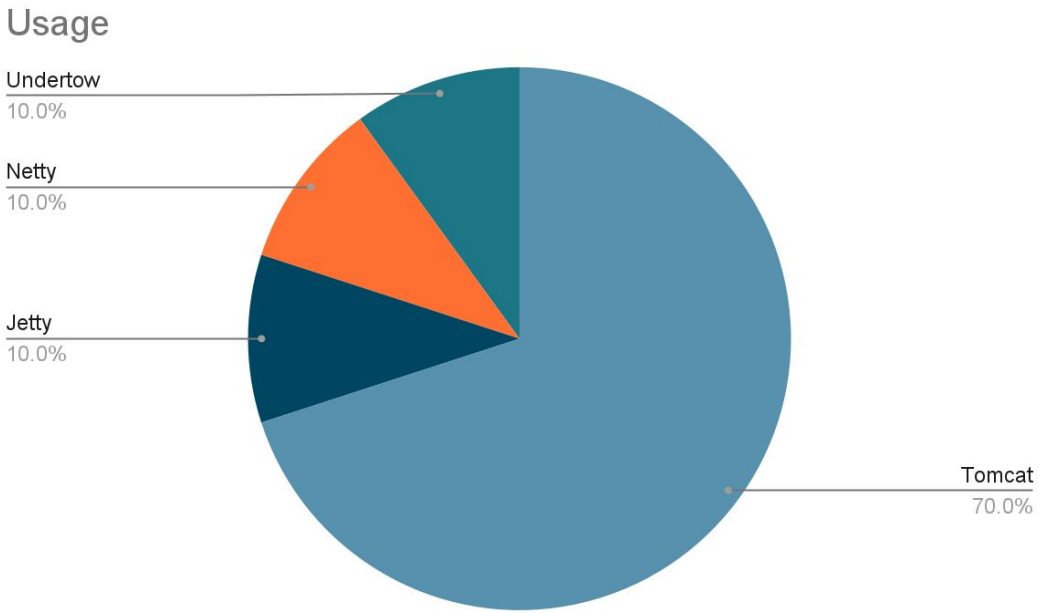


- Java Thread not platform based
- Should improve blocking operation IO, Thread.sleep
- Use the same API
- Can be activated to be used inside the spring boot code

```
spring:
  threads:
    virtual:
      enabled: true
```

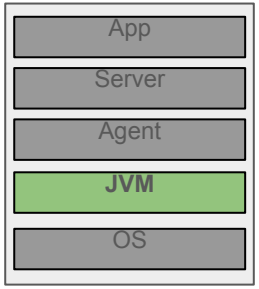
Choose the right server for your use case

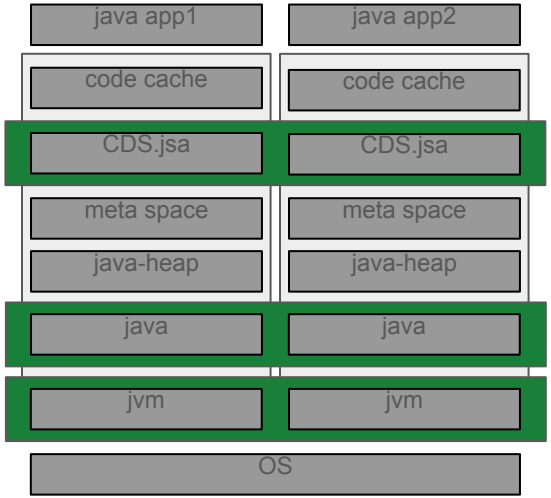




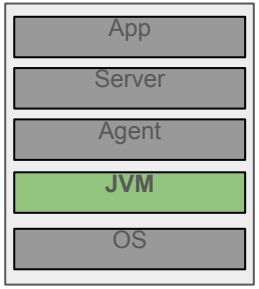
Class data sharing

- Could improve startup time
- Could reduce memory consumption
- Improved runtime performance



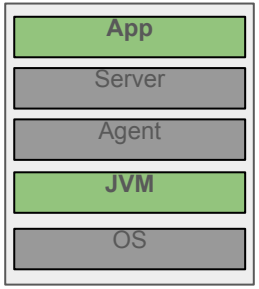


Project GraalVM/Leyden

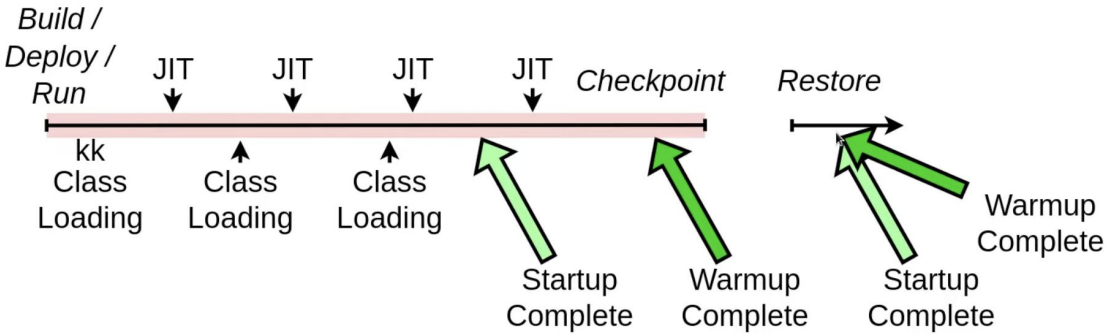


- GraalVM
  - Supported since Spring Boot 3
  - Ahead of time compilation
  - Improve startup time using native
  - Could be more efficient in JVM mode than a classic JVM
  - Could be less efficient than a JVM when throughput is needed
- Leyden
  - Early project
  - Similar to GraalVM
  - Would be included in all JVM

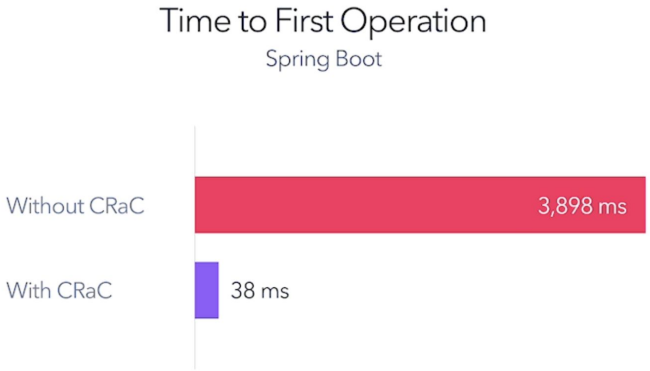
Crac - Coordinated Restore at Checkpoint



- Initiative from Azul
- Improve startup time
- Doesn’t sacrifice throughput



credits:https://www.youtube.com/watch?app=desktop&v=GZ-uxWoFr6w



credits:https://www.azul.com/blog/reduce-java-application-startup-and-warmup-times-with-crac/

Questions ?

The Digit'Owl Factory

- Founded by 4 IT domain associates in 2018
- Clients :
  - Desjardins, Intact, Quebecor, ExPretio, Flo AddEnergie
- Values
  - Creativity
  - Digital disruption
  - The human factor
- Expertise
  - Technical Lead, Kafka, Agility, Mobile Development
  - Kafka, Kotlin training (Spring Boot coming soon) internally or through ETS

Thank you

Ressources

- https://spring.io/blog/2023/12/04/cds-with-spring-framework-6-1/
- https://medium.com/@harshqajjar7110/supercharge-your-spring-boot-app-5-proven-tactics-to-optimize-performance-and-boost-speed-3e4309761358
- https://dev.to/jackynote/efficiently-optimizing-spring-boot-applications-faster-startup-and-lower-memory-usage-hjo
- https://spring.io/blog/2023/10/16/runtime-efficiency-with-spring/
- https://spring.io/blog/2015/12/10/spring-boot-memory-performance/
- https://medium.com/@RamLakshmanan/spring-boot-pet-clinic-app-performance-study-88dd5ceaf162
- https://openjdk.org/projects/crac/
- https://www.graalvm.org/
- https://openjdk.org/projects/leyden/
- https://www.linkedin.com/posts/ivanlopezmartin_java-activity-7152577007313182722-O1dx?utm_source=share&utm_medium=member_desktop
- https://www.linkedin.com/advice/1/what-best-performance-tuning-techniques-optimizing-qy4he
- https://docs.oracle.com/en/java/javase/17/vm/class-data-sharing.html
- https://docs.spring.io/spring-framework/reference/integration/class-data-sharing.html#:~:text=Class%20Data%20Sharing%20(CDS)%20is,the%20creation%20of%20the%20archive.
- https://spring.io/blog/2023/09/20/hello-java-21
- https://medium.com/@toparvion/appcds-for-spring-boot-applications-first-contact-6216db6a4194
- https://www.callicoder.com/spring-boot-actuator/
- https://www.baeldung.com/spring-boot-devtools
- https://www.youtube.com/watch?app=desktop&v=GZ-uxWoFr6w
- https://medium.com/@toparvion/appcds-for-spring-boot-applications-first-contact-6216db6a4194
- https://gatling.io/2023/06/latency-percentiles-for-load-testing-analysis/