



What does

Spring Boot 3.2 + Java 21

offer compared to the previous versions and goLang/echo?

Ufuk Uzun

Trendyol Tech

2023 December



Agenda

- Java 21
- Spring Boot 3.x
- Migration Guide
- Demo & Load Test Results



What's new in Java 21?

- Released features:
 - Virtual Threads (Project Loom)
 - The Record Patterns (Project Amber)
 - Pattern Matching for switch Statements (Project Amber)
 - SequencedCollection (direct access to an ordered collection's first and last elements)
- New preview features:
 - String Templates
 - Structured Concurrency (Project Loom)
 - Scoped Values
 - Unnamed Patterns and Variables
 - Unnamed Classes and Instance Main Methods
- Demo: <https://github.com/ufuk/java-new-features-demos>



Java 21 - Virtual Threads

- Previously known as 'fibers'
- Part of [Project Loom](#): High-throughput, lightweight concurrency model in Java
- “Non-blocking IO, but looks and feels like blocking IO” ~Josh Long
- Platform Threads vs. Virtual Threads vs:

Implementation	OS-managed	JVM-managed
Weight	Heavyweight	Lightweight
Scalability	Limited	High
Blocking impact	Blocks entire OS thread	Non-blocking
Maturity	Traditional approach	Emerging technology

- Demo: <https://github.com/ufuk/java-new-features-demos/blob/main/src/test/java/io/github/ufuk/java19/Java19Tests.java>



What's so special about Spring Boot 3.x?

- [GraalVM](#) Native Support ([3.0](#))
- Min. Java 17 ([3.0](#))
- Min. [Jakarta EE](#) 9 ([3.0](#)) (A goodbye to “~~java~~”, god bless “jakarta”)
- Docker Compose support ([3.1](#))
- Simplified configuration of Testcontainers in integration tests ([3.1](#))
- Virtual Threads ([3.2](#))
- ...



Spring Boot 3.0 - GraalVM Native Support

- Support for compiling Spring applications to native executables using the GraalVM native-image compiler
- Ahead-of-Time (AOT) vs. Just-in-Time (JIT) Compilation
 - Native executable vs. Bytecode

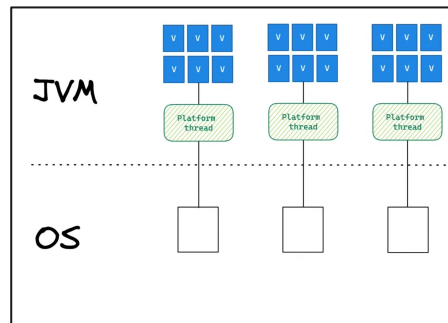
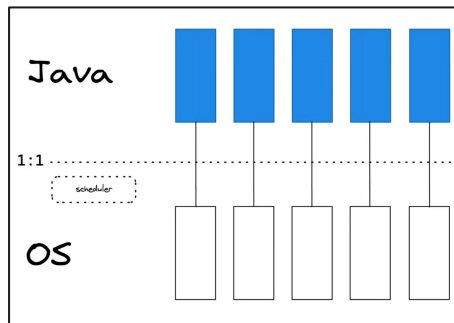
GraalVM Native Support

DEVELOPER TOOLS

Support for compiling Spring applications to native executables using the GraalVM native-image compiler.

Spring Boot 3.2 - Virtual Threads

- Still Thread (Virtual one) per request model, but without blocking the underlying Platform Thread
 - More: [Significant Scalability Benefits in Spring Boot 3.2 using Virtual Threads](#)
- Tomcat has already started supporting: [StandardVirtualThreadExecutor](#)
- `spring.threads.virtual.enabled=true`





Migration Guide

1. First, migrate to Spring Boot 2.7 + Java 17 (from <2.7 & <17)
2. Then, migrate to Spring Boot 3.0 + Java 17
3. Finally, migrate to Spring Boot 3.2 + Java 21

More: <https://github.com/spring-projects/spring-boot/wiki/Spring-Boot-3.0-Migration-Guide>


It's Demo Time

<https://github.com/ufuk/load-test-demos>



Spring Boot 3.2 Demo

<https://start.spring.io/>

 **spring** initializr

Project

☐ Gradle - Groovy ☐ Gradle - Kotlin ☒ **Java** ☐ Kotlin ☐ Groovy

☒ **Maven**

Spring Boot

☐ 3.2.1 (SNAPSHOT) ☒ **3.2.0** ☐ 3.1.7 (SNAPSHOT) ☐ 3.1.6

Project Metadata

Group

io.github.ufuk.springboot3

Artifact

spring-boot-3-demo

Name

spring-boot-3-demo

Description

Demo project for Spring Boot

Package name

io.github.ufuk.springboot3.demo

Packaging

☒ **Jar** ☐ War

Java

☒ **21** ☐ 17

Dependencies

ADD DEPENDENCIES... % + B

GraalVM Native Support **DEVELOPER TOOLS**

Support for compiling Spring applications to native executables using the GraalVM native-image compiler.

Lombok **DEVELOPER TOOLS**

Java annotation library which helps to reduce boilerplate code.

Spring Boot Actuator **OPS**

Supports built in (or custom) endpoints that let you monitor and manage your application - such as application health, metrics, sessions, etc.

Spring Web **WEB**

Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.



Spring Boot Demo: A Simple Blocking Operation

```
@RestController
@RequestMapping("demo")
public class DemoController {

    @GetMapping("sum")
    public DemoSumResponse sum(@RequestParam Long value1, @RequestParam Long value2) throws Exception {
        Thread.sleep(100);
        return new DemoSumResponse(value1 + value2);
    }
}
```



golang/echo Demo: A Simple Blocking Operation

```
e.GET("/demo/sum", func(c echo.Context) (err error) {  
    // Bind request  
    req := new(SumRequest)  
    if err = c.Bind(req); err != nil {  
        return echo.NewHTTPError(http.StatusBadRequest, err.Error())  
    }  
  
    // Blocking operation  
    time.Sleep(100 * time.Millisecond)  
    sum := req.Value1 + req.Value2  
  
    // Return response  
    return c.JSON(http.StatusOK, SumResponse{Result: sum})  
})
```



Load Test Results #1

Run test with <u>100</u> concurrent user for 30 seconds	Spring Boot 2.6 + Java 11 (jvm)	Spring Boot 3.2 + Java 21 (jvm) (no virtual threads)	Spring Boot 3.2 + Java 21 (jvm) (virtual threads enabled)	Spring Boot 3.2 + Java 21 (native) (no virtual threads)	Spring Boot 3.2 + Java 21 (native) (virtual threads enabled)	Echo v4 + Go 1.21
Startup time	2269ms	2269ms	1874ms	119ms	112ms	a few ms
Memory (min, avg, max)	min=206MB, avg=352MB, max=416MB	min=159MB, avg=310MB, max=354MB	min=159MB, avg=243MB, max=251MB	min=67MB, avg=135MB, max=149MB	min=67MB, avg=100MB, max=112MB	min=3MB, avg=13MB, max=15MB
Total request	28722	28537	28300	29024	28614	29600
RPS	954/s	949/s	941/s	965/s	950/s	985/s



Load Test Results #2

Run test with <u>1000</u> concurrent user for 30 seconds	Spring Boot 2.6 + Java 11 (jvm)	Spring Boot 3.2 + Java 21 (jvm) (no virtual threads)	Spring Boot 3.2 + Java 21 (jvm) (virtual threads enabled)	Spring Boot 3.2 + Java 21 (native) (no virtual threads)	Spring Boot 3.2 + Java 21 (native) (virtual threads enabled)	Echo v4 + Go 1.21
Memory (min, avg, max)	min=210MB, avg=541MB, max=626MB	min=159MB, avg=314MB, max=325MB	min=159MB, avg=1175MB, max=1420MB	min=67MB, avg=217MB, max=248MB	min=67MB, avg=484MB, max=929MB	min=3MB, avg=53MB, max=60MB
Total request	64974	63443	281236	64879	204408	297547
Failed request	6520 (10.0%)	4901 (7.7%)	1983 (0.7%)	6324 (9.7%)	6784 (3.3%)	3440 (1.1%)
RPS	2129/s	2081/s	9344/s	2125/s	6768/s	9885/s



Load Test Results #3

Run test with <u>1000</u> concurrent user for 30 seconds. server.tomcat.threads: min=200, max=1000	Spring Boot 3.2 + Java 21 (jvm) (no virtual threads)	Spring Boot 3.2 + Java 21 (jvm) (virtual threads enabled)	Spring Boot 3.2 + Java 21 (native) (no virtual threads)	Spring Boot 3.2 + Java 21 (native) (virtual threads enabled)
Memory (min, avg, max)	min=178MB, avg=820MB, max=883MB	min=159MB, avg=923MB, max=1083MB	min=78MB, avg=432MB, max=501MB	min=68MB, avg=467MB, max=859MB
Total request	285328	286986	202284	206118
Failed request	6223 (2.2%)	4195 (1.4%)	8629 (4.2%)	7690 (3.7%)
RPS	9479/s	9538/s	6716/s	6847/s

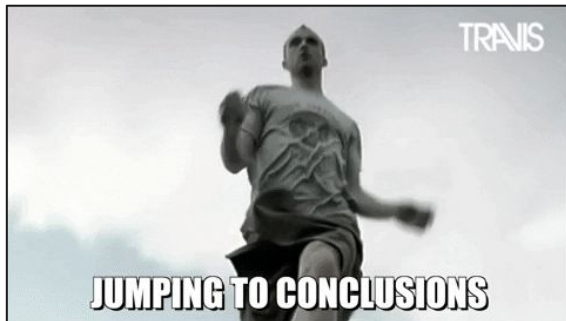


Load Test Results #4

Run test with <u>1000</u> concurrent user for 30 seconds.	Spring Boot 2.6 + Java 11 (jvm) (no virtual threads) <u>server.tomcat.threads: min=100, max=500</u>	Spring Boot 3.2 + Java 21 (jvm) (no virtual threads) <u>server.tomcat.threads: min=100, max=500</u>	Spring Boot 3.2 + Java 21 (jvm) (virtual threads enabled) <u>server.tomcat.threads: min=10, max=200</u> (default values)
Memory (min, avg, max)	min=196MB, avg=643MB, max=702MB	min=168MB, avg=368MB, max=379MB	min=159MB, avg=1211MB, max=1428MB
Total request	148576	148956	286168
Failed request	4801 (3.2%)	5562 (3.7%)	2426 (0.8%)
RPS	4919/s	4936/s	9508/s

Conclusions

- Native compilation support is preferable for:
 - Faster startup time
 - Low resource usage
- Virtual Thread support is preferable for:
 - Scalability
 - Resource utilization





Q&A

Any question?