10 things about Virtual Threads you might want to know

Adam Warski, October 2025
warski.org

SOFTWAREMILL

Who am I?

- Co-founder of SoftwareMill, part of VirtusLab Group
- Software engineer: distributed systems, functional programming
- OSS: Tapir, sttp, Jox, Ox, Hibernate Envers
- Technologies: Java, Scala, Kafka, messaging, event sourcing
- Blogger

I.

Why was Loom introduced?

Green/virtual Threads: history

- **1997**: Green threads added in Java 1.1
- **2000**: Green threads removed in Java 1.3

- **2017**: start of Project Loom
- **2023**: stable release of Virtual Threads in Java 21
- **2025**: some restrictions lifted in Java 24

But why?

- "traditional" threads are "heavy"
- Java used to have a synchronous model
- but, the web happened

Before Loom: Scaling threads

- Core idea: make sure threads are never idle
  - thread pools
  - `Executor`
  - `Future`, `CompletableFuture`
  - reactive programming

In the old days

```
var person = db.findById(id);

if (person.hasLicense()) {
  bankingService.transferFunds(person, dealership, amount);
  dealerService.reserveCar(person);
}
```

More recently

```
db.findById(id).thenCompose(person -> {
  if (person.hasLicense()) {
    return bankingService
      .transferFunds(person, dealership, amount)
      .thenCompose(transferResult ->
          dealerService.reserveCar(person));
  } else {
    return CompletableFuture.completedFuture(null);
  }
});
```

technical concerns > code readability

Goals of virtual threads

- maintain high thread utilization
- retain simplicity of synchronous code

By:

- reintroduce **direct syntax**
- eliminate **virality**
- eliminate **lost context**

2.

How Loom works?

JDK 21

This release is the Reference Implementation of version 21 of the Java SE Platform, as specified by JSR 396 in the Java Community Process.

JDK 21 reached General Availability on 19 September 2023. Production-ready binaries under the GPL are available from Oracle; binaries from other vendors will follow shortly.

The features and schedule of this release were proposed and tracked via the JEP Process, as amended by the JEP 2.0 proposal. The release was produced using the JDK Release Process (JEP 3).

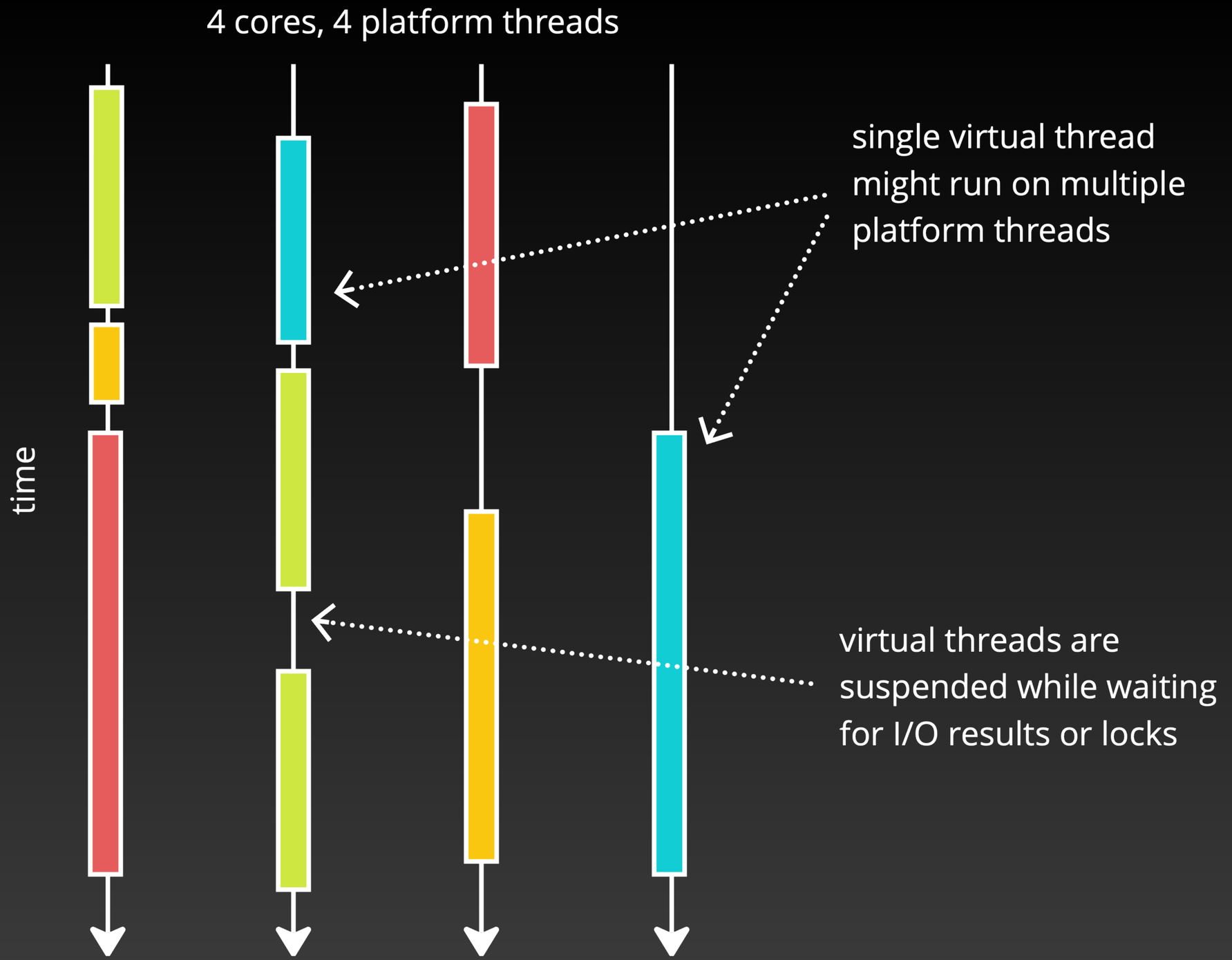
Features

- 430: String Templates (Preview)
- 431: Sequenced Collections
- 439: Generational ZGC
- 440: Record Patterns
- 441: Pattern Matching for switch
- 442: Foreign Function & Memory API (Third Preview)
- 443: Unnamed Patterns and Variables (Preview)
- 444: Virtual Threads

```
timed(() -> {
  var threads = new Thread[10_000_000];
  var results = ConcurrentHashMap.newKeySet();
  for (int i=0; i<threads.length; i++) {
    threads[i] = Thread
        .ofVirtual()
        .start(() -> results.add(0));
  }

  for (Thread thread : threads) {
    thread.join();
  }

  return null;
});
```


4 cores, 4 platform threads
time
single virtual thread
might run on multiple
platform threads
virtual threads are
suspended while waiting
for I/O results or locks

Loom contributions

- runtime: scheduler
- API: platform & virtual threads
- retrofitting blocking APIs

Blocking is once again completely fine!

In the new days

```
var person = db.findById(id);

if (person.hasLicense()) {
  bankingService.transferFunds(person, dealership, amount);
  dealerService.reserveCar(person);
}
```

3.

When to use Loom?

CPU vs IO-bound computations

- **CPU-bound**
  - e.g. complex maths, rendering, encryption/decryption
  - >10ms
- **IO-bound**
  - e.g. DB calls, networks calls, filesystem calls
  - most of the time blocked, waiting for I/O

Monopolization

- When a virtual thread stays on a carrier thread for a long time
- Hurts fairness
- Prevention, inside computation loops:
  - `Thread.yield`
  - `LockSupport.parkNanos(1)`

Choosing thread type

- **IO-bound**: virtual threads
- **CPU-bound**:
  - delegate to a dedicated platform-thread executor
  - await on the `Future`s
  - the OS scheduler ensures fairness

4.

What is pinning?

Pinning

Occurs when a VT thread does not release the carrier thread when blocking

JDK 24 (March 2025)

JDK 24

This release is the Reference Implementation of version 24 of the Java SE Platform, as specified by JSR 399 in the Java Community Process.

JDK 24 reached General Availability on 18 March 2025. Production-ready binaries under the GPL are available from Oracle; binaries from other vendors will follow shortly.

The features and schedule of this release were proposed and tracked via the JEP Process, as amended by the JEP 2.0 proposal. The release was produced using the JDK Release Process (JEP 3).

Features

404: Generational Shenandoah (Experimental)
450: Compact Object Headers (Experimental)
472: Prepare to Restrict the Use of JNI
475: Late Barrier Expansion for G1
478: Key Derivation Function API (Preview)
479: Remove the Windows 32-bit x86 Port
483: Ahead-of-Time Class Loading & Linking
484: Class-File API
485: Stream Gatherers
486: Permanently Disable the Security Manager
487: Scoped Values (Fourth Preview)
488: Primitive Types in Patterns, instanceof, and switch (Second Preview)
489: Vector API (Ninth Incubator)
490: ZGC: Remove the Non-Generational Mode
491: Synchronize Virtual Threads without Pinning
492: Flexible Constructor Bodies (Third Preview)
493: Linking Run-Time Images without JMODs
494: Module Import Declarations (Second Preview)
495: Simple Source Files and Instance Main Methods (Fourth Preview)
496: Quantum-Resistant Module-Lattice-Based Key Encapsulation Mechanism
497: Quantum-Resistant Module-Lattice-Based Digital Signature Algorithm
498: Warn upon Use of Memory-Access Methods in sun.misc.Unsafe
499: Structured Concurrency (Fourth Preview)

Pinning

- **Java 21:**
  - `synchronized` blocks
  - datagram sockets
  - DNS lookups
  - native blocking code
  - file operations
- **Java 24:**
  - ~~`synchronized` blocks~~
  - datagram sockets
  - DNS lookups
  - native blocking code
  - file operations

How to trace pinning?

```
-Djdk.tracePinnedThreads=short
```

Unescapable pinning

- Page faults are inherently blocking
  - outside the control of the JVM
- From the Kernel's perspective, file operations never block
- Solution:
  - compensating threads
  - `io_uring`

What is blocking in Loom?

Async file IO with Java and io_uring

5.

How to propagate context?

JDK 21 (LTS, September 2023)

JDK 21

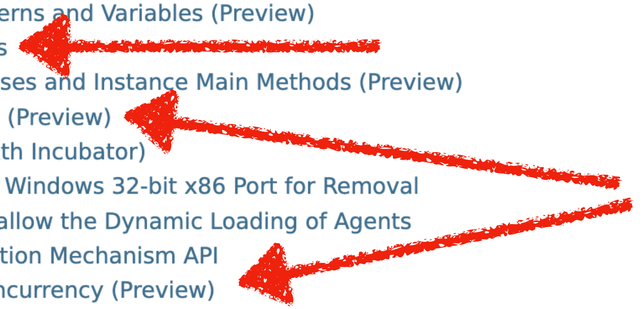
This release is the Reference Implementation of version 21 of the Java SE Platform, as specified by JSR 396 in the Java Community Process.

JDK 21 reached General Availability on 19 September 2023. Production-ready binaries under the GPL are available from Oracle; binaries from other vendors will follow shortly.

The features and schedule of this release were proposed and tracked via the JEP Process, as amended by the JEP 2.0 proposal. The release was produced using the JDK Release Process (JEP 3).

Features

- 430: String Templates (Preview)
- 431: Sequenced Collections
- 439: Generational ZGC
- 440: Record Patterns
- 441: Pattern Matching for switch
- 442: Foreign Function & Memory API (Third Preview)
- 443: Unnamed Patterns and Variables (Preview)
- 444: Virtual Threads
- 445: Unnamed Classes and Instance Main Methods (Preview)
- 446: Scoped Values (Preview)
- 448: Vector API (Sixth Incubator)
- 449: Deprecate the Windows 32-bit x86 Port for Removal
- 451: Prepare to Disallow the Dynamic Loading of Agents
- 452: Key Encapsulation Mechanism API
- 453: Structured Concurrency (Preview)



Java 25 (LTS, September 2025)

JDK 25

This release is the Reference Implementation of version 25 of the Java SE Platform, as specified by JSR 400 in the Java Community Process.

JDK 25 reached General Availability on 16 September 2025. Production-ready binaries under the GPL are available from Oracle; binaries from other vendors will follow shortly.

The features and schedule of this release were proposed and tracked via the JEP Process, as amended by the JEP 2.0 proposal. The release was produced using the JDK Release Process (JEP 3).

Features

470: PEM Encodings of Cryptographic Objects (Preview)
502: Stable Values (Preview)
503: Remove the 32-bit x86 Port
505: Structured Concurrency (Fifth Preview)
506: Scoped Values
507: Primitive Types in Patterns, instanceof, and switch (Third Preview)
508: Vector API (Tenth Incubator)
509: JFR CPU-Time Profiling (Experimental)
510: Key Derivation Function API
511: Module Import Declarations
512: Compact Source Files and Instance Main Methods
513: Flexible Constructor Bodies
514: Ahead-of-Time Command-Line Ergonomics
515: Ahead-of-Time Method Profiling
518: JFR Cooperative Sampling
519: Compact Object Headers
520: JFR Method Timing & Tracing
521: Generational Shenandoah

Scoped values

```
class Framework {
  private static final ScopedValue<FrameworkContext> CONTEXT
    = ScopedValue.newInstance();

  void serve(Request request, Response response) {
    var context = createContext(request);
    where(CONTEXT, context)
      .run(() -> Application.handle(request, response));
  }

  PersistedObject readKey(String key) {
    var context = CONTEXT.get();
    var db = getDBConnection(context);
    db.readKey(key);
  }
}
```

Scoped values JEP

`ScopedValue` as a `ThreadLocal` replacement

- Not a drop-in
- Only structured usage
- Immutable data
- Thread locals: often used to cache expensive objects
- Inherited within concurrency scopes

6.

How to manage a large number of virtual threads?

Structured concurrency

```
Response handle() throws InterruptedException {
    try (var scope = new StructuredTaskScope.open()) {
        Subtask<String>  user  = scope.fork(() -> findUser());
        Subtask<Integer> order = scope.fork(() -> fetchOrder());

        scope.join();

        return new Response(user.get(), order.get());
    }
}
```

Structured concurrency JEP

Structured concurrency

- Group related tasks as single units of work
- Eliminate thread leaks
- Eliminate cancellation delays
- Concept known from Python, Kotlin
  - Syntactic structure of code determines the lifetime of threads

Structured concurrency

- Evolving API
- Misuse possible

```
Response handle() throws InterruptedException {
  try (var scope = StructuredTaskScope.open()) {
    Subtask<String> user = scope.fork(() -> findUser());
    Subtask<Integer> order = scope.fork(() -> fetchOrder());

    scope.join();

    return new Response(user.get(), order.get());
  }
}
```

Critique of JEP 505

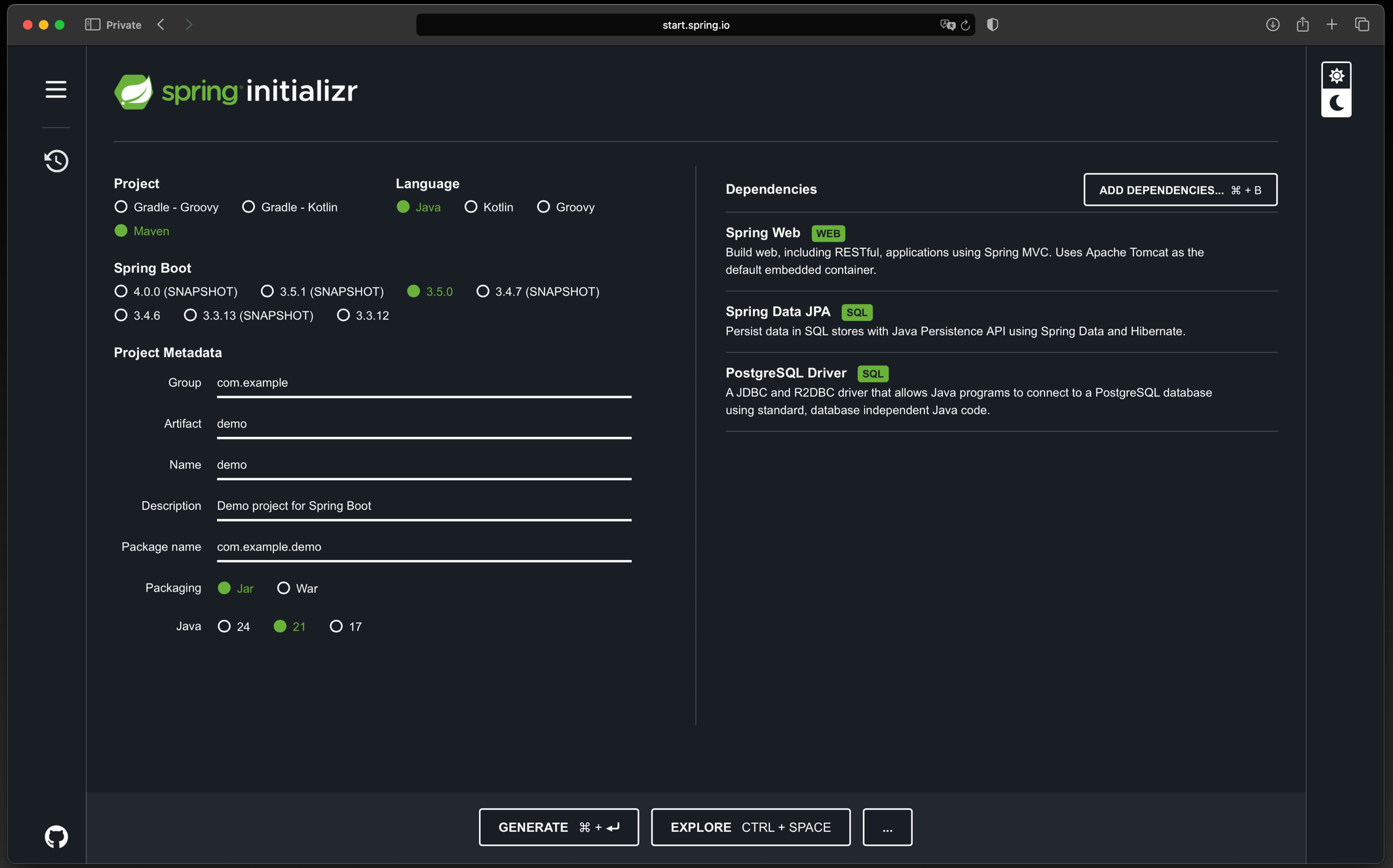
Jox library

```
Response handle() throws InterruptedException {
  return supervised(scope -> {
    Fork<String> user = scope.fork(() -> findUser());
    Fork<Integer> order = scope.fork(() -> fetchOrder());

   return new Response(user.join(), order.join());
  });
}
```

7.

Which web frameworks work with Loom?


Private
start.spring.io
spring initializr
Project
Gradle - Groovy
Gradle - Kotlin
Maven
Language
Java
Kotlin
Groovy
Spring Boot
4.0.0 (SNAPSHOT)
3.5.1 (SNAPSHOT)
3.5.0
3.4.7 (SNAPSHOT)
3.4.6
3.3.13 (SNAPSHOT)
3.3.12
Project Metadata
Group
com.example
Artifact
demo
Name
demo
Description
Demo project for Spring Boot
Package name
com.example.demo
Packaging
Jar
War
Java
24
21
17
Dependencies
ADD DEPENDENCIES... ⌘ + B
Spring Web
WEB
Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.
Spring Data JPA
SQL
Persist data in SQL stores with Java Persistence API using Spring Data and Hibernate.
PostgreSQL Driver
SQL
A JDBC and R2DBC driver that allows Java programs to connect to a PostgreSQL database using standard, database independent Java code.
GENERATE ⌘ + ↵
EXPLORE CTRL + SPACE
...

```java
@RestController
public class CustomerController {
  private static final Logger logger =
    LoggerFactory.getLogger(CustomerController.class);

  private final CustomerRepository customerRepository;

  private void logThread() {
    var current = Thread.currentThread();
    logger.info("Thread name: {}, id: {}, isVirtual: {}",
      current.getName(), current.threadId(), current.isVirtual());
  }

  @GetMapping("/test")
  public String test(@RequestParam(value = "id") Long id) {
    var result = customerRepository.findById(id);
    logThread();

    // …
  }
}
```

```
2025-05-23T10:51:11.998+02:00  INFO 17417 --- [demo] [nio-8080-exec-1] com.example.demo.CustomerController      : Thread name: http-nio-8080-exec-1, id: 47, isVirtual: false
2025-05-23T10:51:11.998+02:00  INFO 17417 --- [demo] [nio-8080-exec-1] com.example.demo.CustomerController      : Thread name: http-nio-8080-exec-1, id: 47, isVirtual: false
2025-05-23T10:51:11.998+02:00  INFO 17417 --- [demo] [nio-8080-exec-1] com.example.demo.CustomerController      : Customer not found
```

spring.threads.virtual.enabled=true

```
2025-05-23T11:00:15.454+02:00  INFO 20598 --- [demo] [omcat-handler-0] com.example.demo.CustomerController      : Thread name: tomcat-handler-0, id: 55, isVirtual: true
2025-05-23T11:00:15.454+02:00  INFO 20598 --- [demo] [omcat-handler-0] com.example.demo.CustomerController      : Thread name: tomcat-handler-0, id: 55, isVirtual: true
2025-05-23T11:00:15.454+02:00  INFO 20598 --- [demo] [omcat-handler-0] com.example.demo.CustomerController      : Customer not found
```

Guides

Whatever you're building, these guides are designed to get you productive as quickly as possible – using the latest Spring project releases and techniques as recommended by the Spring team.

virtual

No result found.

Spring
https://spring.io › blog › 2022/10/11 › embracing-virtu...

Embracing Virtual Threads

11 Oct 2022 — **Virtual Threads are much more lightweight to allocate**, and the number of threads is no longer a primary limitation for scalability.

Baeldung
https://www.baeldung.com › spring-6-virtual-threads

Working with Virtual Threads in Spring

13 Jun 2024 — In this short tutorial, we'll see how to leverage the great power of **virtual threads** in a **Spring** Boot Application.

BellSoft
https://bell-sw.com › blog › a-guide-to-using-virtual-thr...

How to use virtual threads with Spring Boot

14 Dec 2023 — You can **set the virtual threads configuration in application.properties**. The first property enables the standard configuration from Spring Boot.

can I use virtual threads with spring boot?

Yes, you can use virtual threads with Spring Boot, provided you're using Java 21 or newer and Spring Boot 3.2 or later. Virtual threads, introduced as a stable feature in Java 21, offer a lightweight concurrency model that can significantly improve the scalability of I/O-bound applications by allowing the creation of millions of concurrent threads with minimal overhead. Java Code Geeks

✅ Enabling Virtual Threads in Spring Boot

To enable virtual threads in your Spring Boot application, follow these steps:

1. **Ensure Java 21+ and Spring Boot 3.2+**: Virtual threads are a stable feature starting from Java 21, and Spring Boot 3.2 provides built-in support for them. Read Medium articles with AI +3
2. **Set the Configuration Property**: In your `application.properties` or `application.yml`, add: Stack Overflow

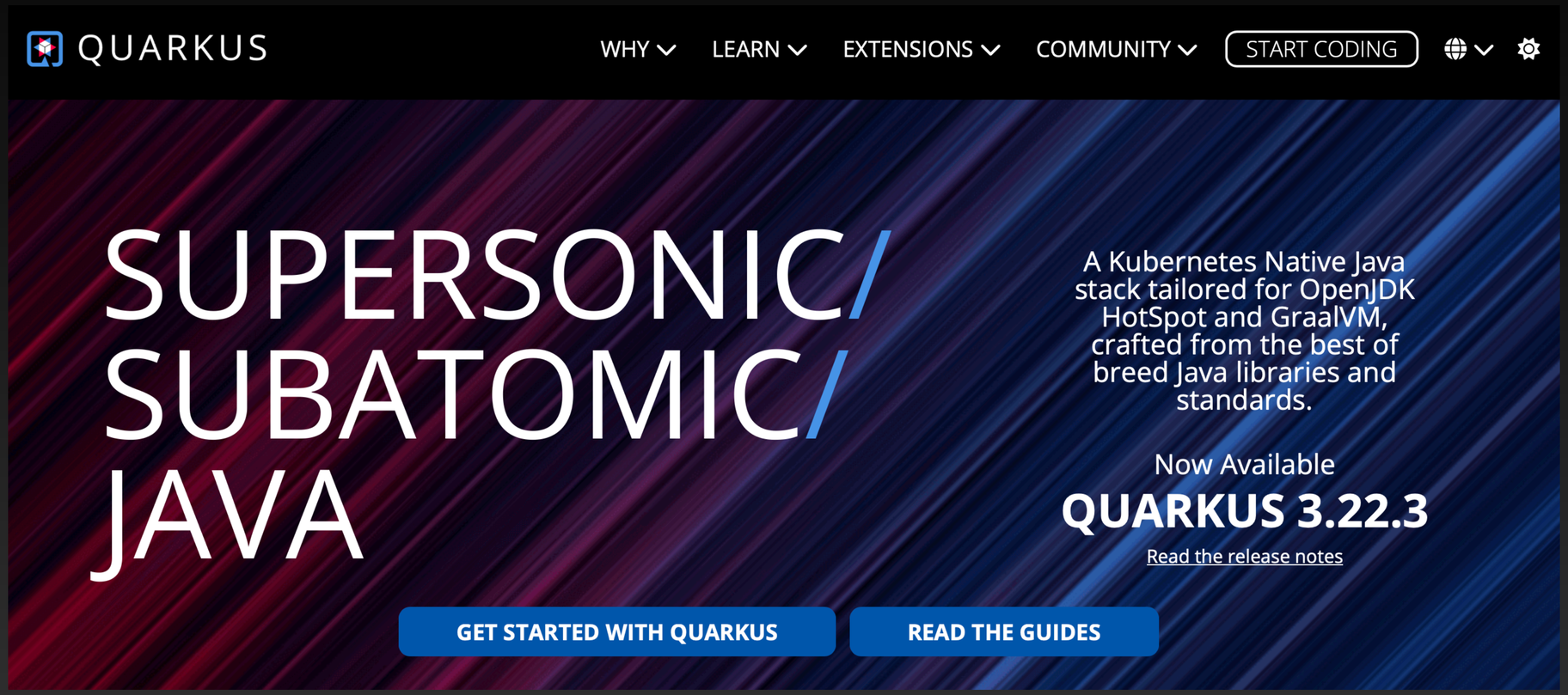
properties — Copy — Edit

```
spring.threads.virtual.enabled=true
```

This property instructs Spring Boot to utilize virtual threads where applicable, including for embedded web servers like Tomcat or Jetty, `@Async` methods, and task schedulers .

Spring Boot

- Using Virtual Threads is very easy
- But is not the default, and doesn’t seem to be encouraged
- Default Java is 17
- Spring keeps the conservative side here
- Reactive is opt-in, default is "old-school" synchronous


QUARKUS
WHY
LEARN
EXTENSIONS
COMMUNITY
START CODING
SUPERSONIC/
SUBATOMIC/
JAVA
A Kubernetes Native Java stack tailored for OpenJDK HotSpot and GraalVM, crafted from the best of breed Java libraries and standards.
Now Available
QUARKUS 3.22.3
Read the release notes
GET STARTED WITH QUARKUS
READ THE GUIDES

```
__  ____  __  _____   ___  __ ____  ______
 --/ __ \/ / / / _ | / _ \/ //_/ / / / __/
 -/ /_/ / /_/ / __ |/ , _/ ,< / /_/ /\ \
--\___\_\____/_/ |_/_/|_/_/|_|\____/___/
2025-05-26 09:12:26,446 INFO  [io.quarkus] (Quarkus Main Thread) quarkus 1.0.0-SNAPSHOT on JVM (powered by Quarkus 3.22.3) started in 0.208s. Listening on: http://localhost:8080

2025-05-26 09:12:26,447 INFO  [io.quarkus] (Quarkus Main Thread) Profile dev activated. Live Coding activated.
2025-05-26 09:12:26,447 INFO  [io.quarkus] (Quarkus Main Thread) Installed features: [cdi, rest, smallrye-context-propagation, vertx]
2025-05-26 09:12:26,448 INFO  [io.qua.dep.dev.RuntimeUpdatesProcessor] (vert.x-worker-thread-8) Live reload total time: 0.304s
2025-05-26 09:12:26,449 INFO  [org.acm.GreetingResource] (executor-thread-1) Thread name: executor-thread-1, id: 229, isVirtual: false
```

```java
@GET
@Produces(MediaType.TEXT_PLAIN)
@RunOnVirtualThread
public String hello() {
    // …
}
```

```
__  ____  __  _____   ___  __ ____  ______
 --/ __ \/ / / / _ | / _ \/ //_/ / / / __/
 -/ /_/ / /_/ / __ |/ , _/ ,< / /_/ /\ \
--\___\_\____/_/ |_/_/|_/_/|_|\____/___/
2025-05-26 09:14:36,287 INFO  [io.quarkus] (Quarkus Main Thread) quarkus 1.0.0-SNAPSHOT on JVM (powered by Quarkus 3.22.3) started in 0.213s. Listening on: http://localhost:8080

2025-05-26 09:14:36,288 INFO  [io.quarkus] (Quarkus Main Thread) Profile dev activated. Live Coding activated.
2025-05-26 09:14:36,288 INFO  [io.quarkus] (Quarkus Main Thread) Installed features: [cdi, rest, smallrye-context-propagation, vertx]
2025-05-26 09:14:36,291 INFO  [io.qua.dep.dev.RuntimeUpdatesProcessor] (vert.x-worker-thread-10) Live reload total time: 0.238s
2025-05-26 09:14:36,293 INFO  [org.acm.GreetingResource] (quarkus-virtual-thread-0) Thread name: quarkus-virtual-thread-0, id: 463, isVirtual: true
```

QUARKUS

WHY LEARN EXTENSIONS COMMUNITY START CODING

By Version 3.22.3 - Latest By Category virtual

Guides - Latest

Virtual Thread support reference

This guide explains how to benefit from Java 21+ **virtual** threads in Quarkus application.

...annotation, such as: **Virtual** threads in REST applications **Virtual** threads in reactive messaging applications **Virtual** threads in gRPC...

...are not run on the **virtual** thread). However, thread locals are not propagated. **Virtual** thread names **Virtual** threads are created...

Quarkus Virtual Thread support for gRPC services

This guide explains how to benefit from Java **virtual** threads when implementing a gRPC service.

...with Quarkus **Virtual** Thread support to read more about Java **virtual** threads in general and the Quarkus **Virtual** Thread support...

...Page Quarkus **Virtual** Thread support for gRPC services This guide explains how to benefit from Java **virtual** threads when ...

Use virtual threads in REST applications

How to use **virtual** threads in a REST application

...Edit this Page Use **virtual** threads in REST applications In this guide, we see how you can use **virtual** threads in a REST ...

...This section is not about **virtual** threads. Because we need to do some I/O to demonstrate **virtual** threads usage, we need...

Quarkus Virtual Thread support with Reactive Messaging

This guide explains how to benefit from Java **virtual** threads when writing message processing applications in Quarkus.

...with Quarkus **Virtual** Thread support to read more about Java **virtual** threads in general and the Quarkus **Virtual** Thread support...

...Page Quarkus **Virtual** Thread support with Reactive Messaging This guide explains how to benefit from Java **virtual** threads when...

Writing REST Services with Quarkus REST (formerly RESTEasy Reactive)

Discover how to develop highly scalable reactive REST services with Jakarta REST and Quarkus REST.

Dev Services for RabbitMQ

Dev Services for RabbitMQ automatically starts a RabbitMQ broker in dev mode and when running tests.

...a default **virtual** host of /. To define additional RabbitMQ **virtual** hosts, provide the names of the **virtual** hosts in the...

...can define **Virtual** Hosts, Exchanges, Queues, and Bindings through standard Quarkus configuration. Defining **virtual** hosts RabbitMQ...

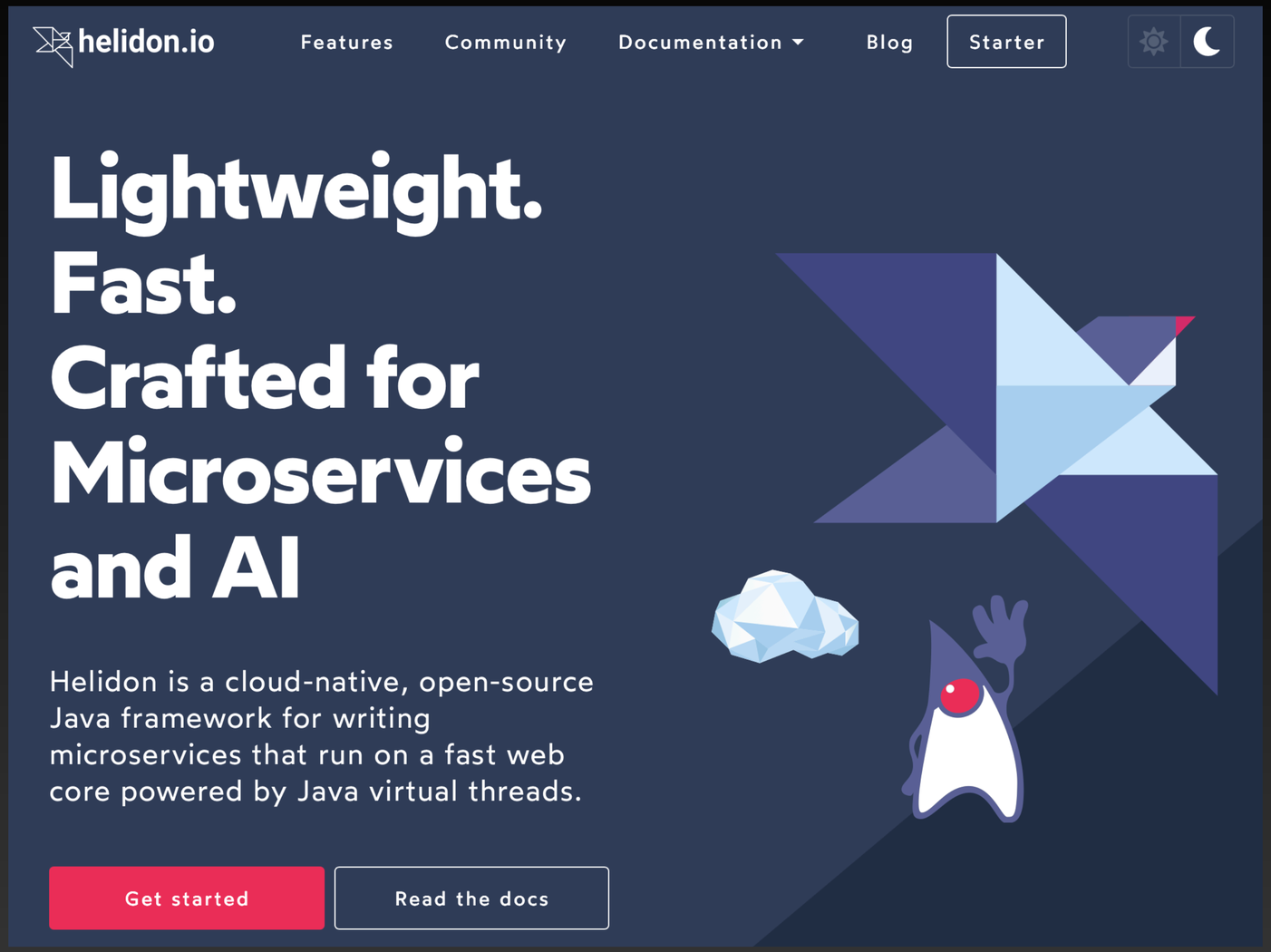
Quarkus

- Reactive core
- Getting started guide: an "old-school" synchronous app
- Virtual threads, thread pooling and reactive can be mixed
  - 3 thread pools
- Detailed guide with discussion & tradeoffs
  - monopolization, pinning, `ThreadLocal`s
  - Java ecosystem needs to become VT-friendly
  - VT useful for I/O-bound workloads only

Quarkus VT guide

Works OOTB

	Starting & joining 10m threads
Temurin 21.0.7	3s
GraalVM 24.0.1	4s
GraalVM 24.0.1 native image	17s

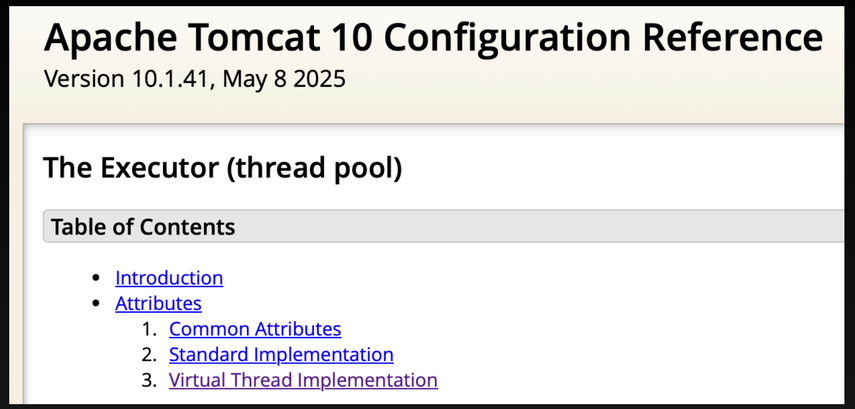

helidon.io
Features
Community
Documentation
Blog
Starter
Lightweight.
Fast.
Crafted for
Microservices
and AI
Helidon is a cloud-native, open-source Java framework for writing microservices that run on a fast web core powered by Java virtual threads.
Get started
Read the docs

```
WebServer.builder()
          .host("localhost")
          .port(8080)
          .routing(routing -> routing
            .get("/hello", (req, res) -> {
              logThread();
              res.send("Hello World!");
            }))
          .build()
          .start();
```

```
16:12:02.595 [[0×1645022f 0×3dd9164b] WebServer socket] INFO demo.Main -- Thread name: [0×1645022f 0×3dd9164b] WebServer socket, id: 37, isVirtual: true
16:12:02.596 [[0×1645022f 0×3dd9164b] WebServer socket] DEBUG io.helidon.webserver.http1.Http1LoggingConnectionListener.send -- [0×1645022f 0×3dd9164b] send status: 200 OK
```

Helidon

- A collection of Java libraries for writing microservices
- Nima: a clean-slate implementation of a web server
  - using virtual threads
  - replaces Netty
  - HTTP 1, HTTP2, gRPC
- SE (Standard Edition), MP (MicroProfile)




```
@ManagedExecutorDefinition(
  name = "java:module/concurrent/MyExecutor",
  maxAsync = 8,
  virtual = true)
```

- compatibility concerns
- `ThreadLocal`s
- deeper integration possible in v12

- threading model-agnostic
- no priority
- subclasses of `Thread` cannot be virtual

8.

Which libraries work with Loom?

Most libraries "just work"

e.g.: Resilience4j

some implemented better compatibility,
e.g. JDBC drivers

- Usually works
- By default, otel's Context is propagated `ThreadLocals`
- When using VT, the host library/framework must set it correctly
- What about structured concurrency?

```
Response handle() throws InterruptedException {
  try (var scope = new StructuredTaskScope.open()) {
    Subtask<String>  user  = scope.fork(() -> findUser());
    Subtask<Integer> order = scope.fork(() -> fetchOrder());

    scope.join();

    return new Response(user.get(), order.get());
  }
}
```

Propagating context in OTEL w/ VT

- We need to propagate by hand
- Scoped values + `ContextStorage`?

```
Response handle() throws InterruptedException {
  try (var scope = new StructuredTaskScope.open(
      null, new PropagatingVirtualThreadFactory())) {
    Subtask<String>  user  = scope.fork(() -> findUser());
    Subtask<Integer> order = scope.fork(() -> fetchOrder());

    scope.join();

    return new Response(user.get(), order.get());
  }
}
```

Logging + MDC

- Similar problem
- Some LogBack hacking needed (to replace `MDCAdapter`)
- Using `ScopedValue`s, only structured usage

```
InheritableMDC.where("key", "value", () -> {
    // …
});
```

Inheritable MDC source

9.

What about reactive programming & streaming?

Reactive

RxJava	Integration module available (`RxJavaFiberInterop`), doesn't seem used much Possibility to use VT-executor as a scheduler Await on `Future`s
Project Reactor	Use VT as the elastic scheduler Motivation for reactive programming: avoid wasting expensive resources
Vert.X	Guide available Submit to executor Await on `Future`s
Netty	Use a VT-based executor Await on `Future`s

Is reactive necessary?

Java AMA from 2021

Streaming

- Java streams
- Stream gatherers (JEP 485)
- Synchronous

```
List<Integer> numbers = …

List<List<Integer>> result =
  numbers.stream()
    .filter(n -> n % 2 != 0)
    .gather(mapConcurrent(4, n -> n * n))
    .gather(windowFixed(3))
    .collect(Collectors.toList());
```

JEP 485: Stream Gatherers

- Jox Flows
- Asynchronous
- Reactive streams-compatible

```
Flows.range(1, 100, 1)
  .throttle(1, Duration.ofSeconds(1))
  .mapPar(4, i -> {
    Thread.sleep(5000);
    var j = i*3;
    return j+1;
  })
  .filter(i -> i % 2 == 0)
  .runToList();
```

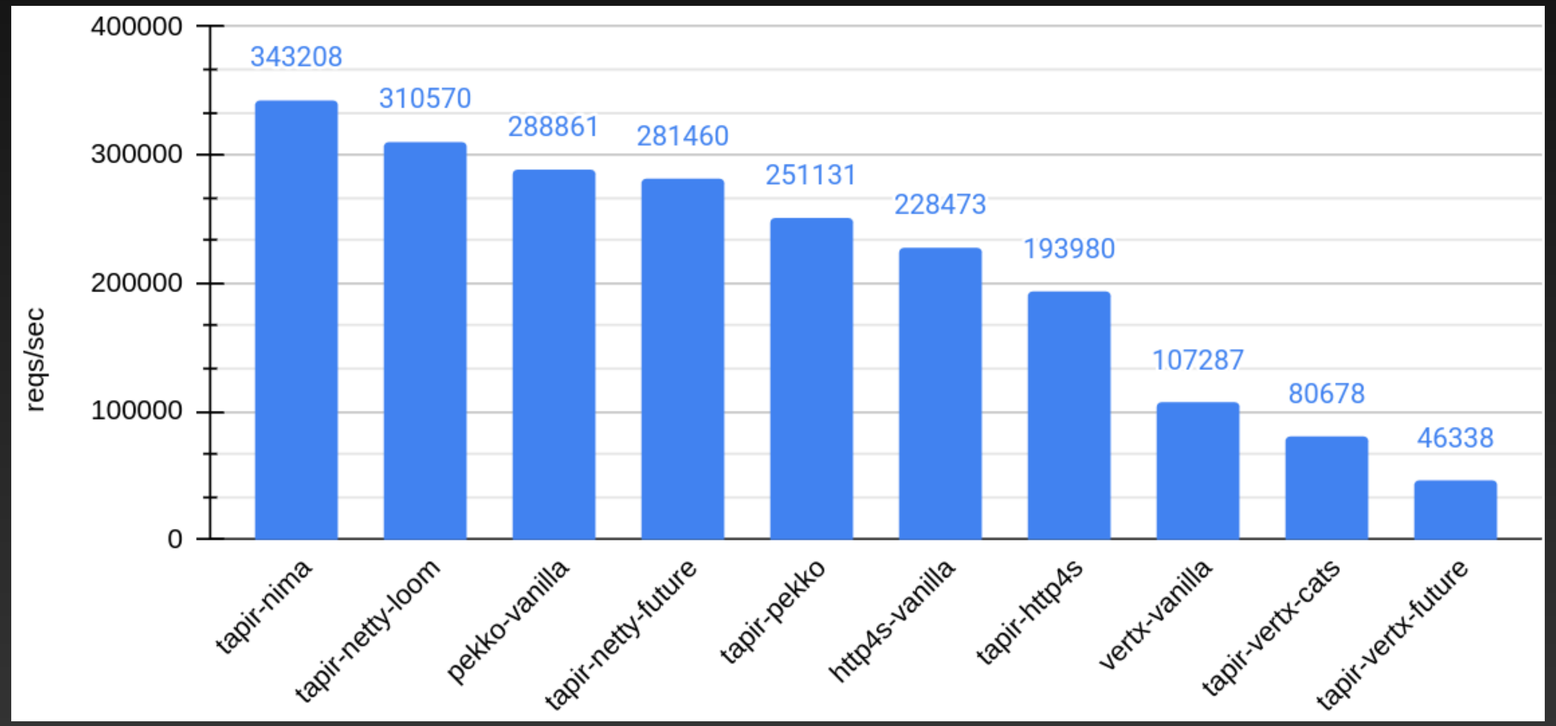
Comparing Java Streams with Jox Flows

Jox Docs

IO.

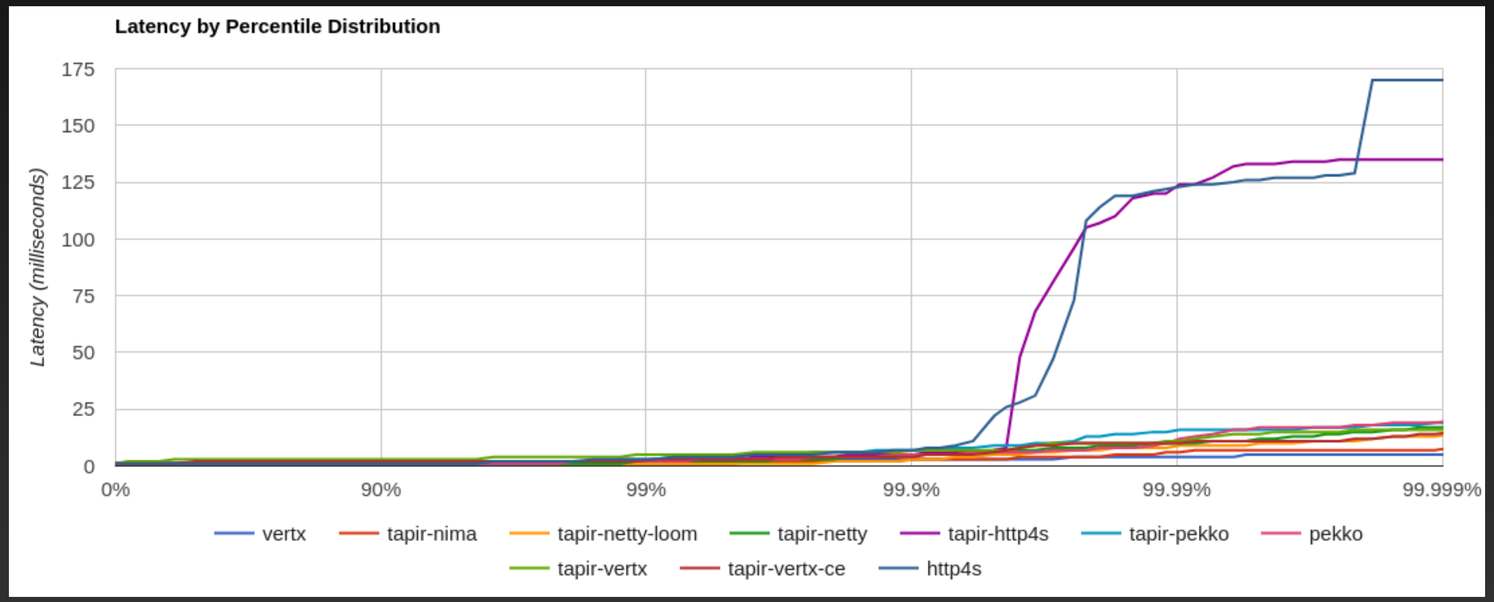
Are virtual threads fast?

HTTP server benchmark: different backends



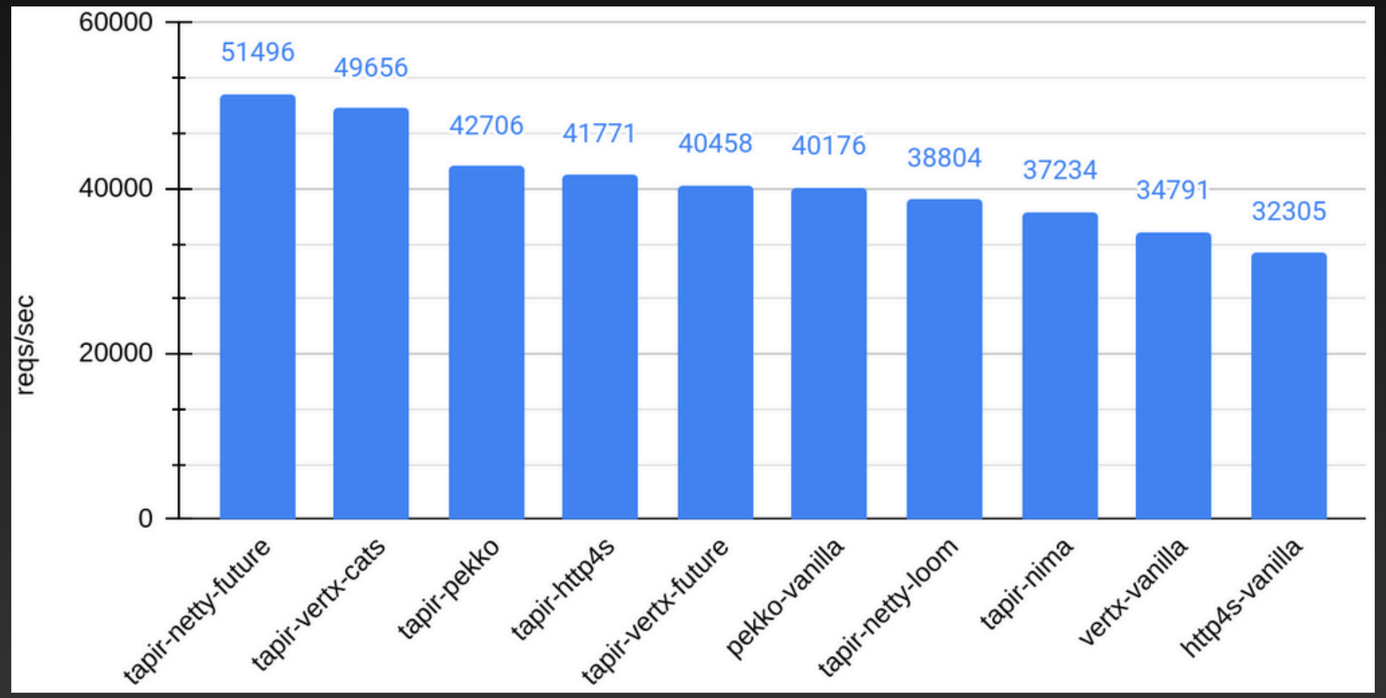
SimpleGet

HTTP server benchmark: different backends



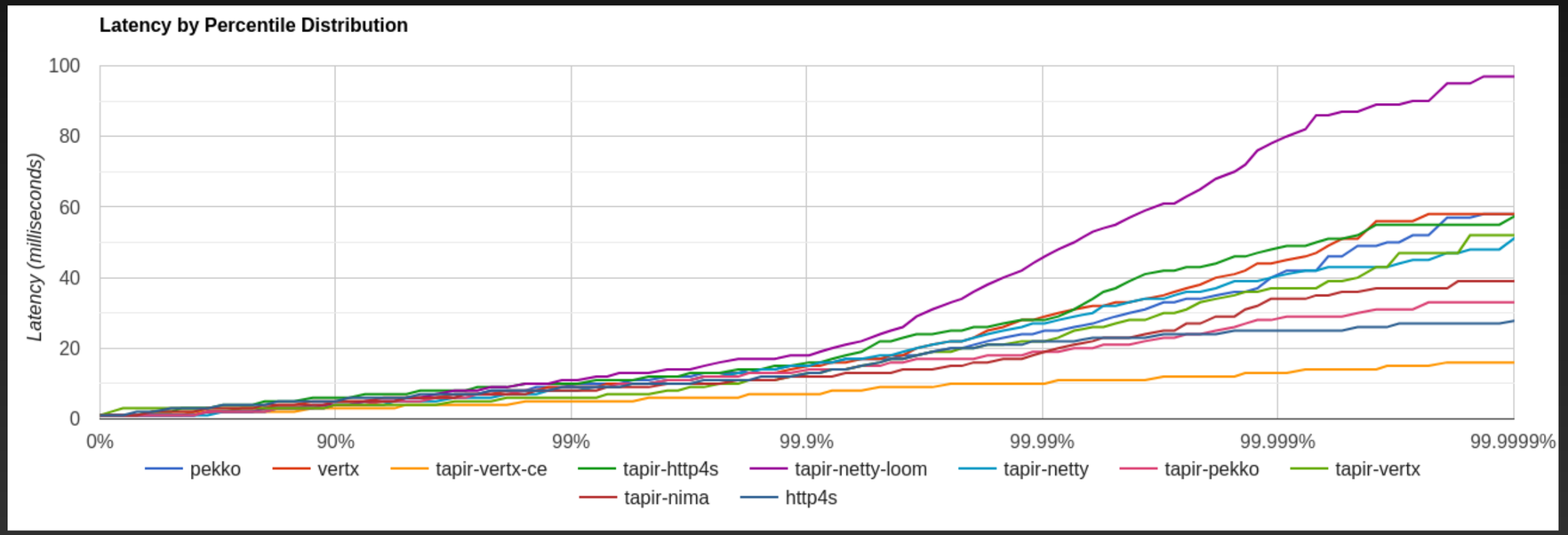
SimpleGet

HTTP server benchmark: different backends



PostBytes

HTTP server benchmark: different backends



PostBytes

Benchmarking Tapir

Spring Boot + DB application

Test B: Moderate Concurrency (500 users, 20 DB connections)

Java Version	Spring Boot Version	Platform Threads (req/s)	Virtual Threads (req/s)	Error Rate (Virtual)
19	3.3.12	71.59	71.52	0.55%
19	3.5.0	61.79	62.94	0.68%
24	3.5.0	61.43	64.32	0.65%

Key Finding: Even under moderate load, version differences are minimal. Virtual threads show slight performance variations but no clear version-based pattern.

Test C: High Concurrency (1000 users, 20 DB connections)

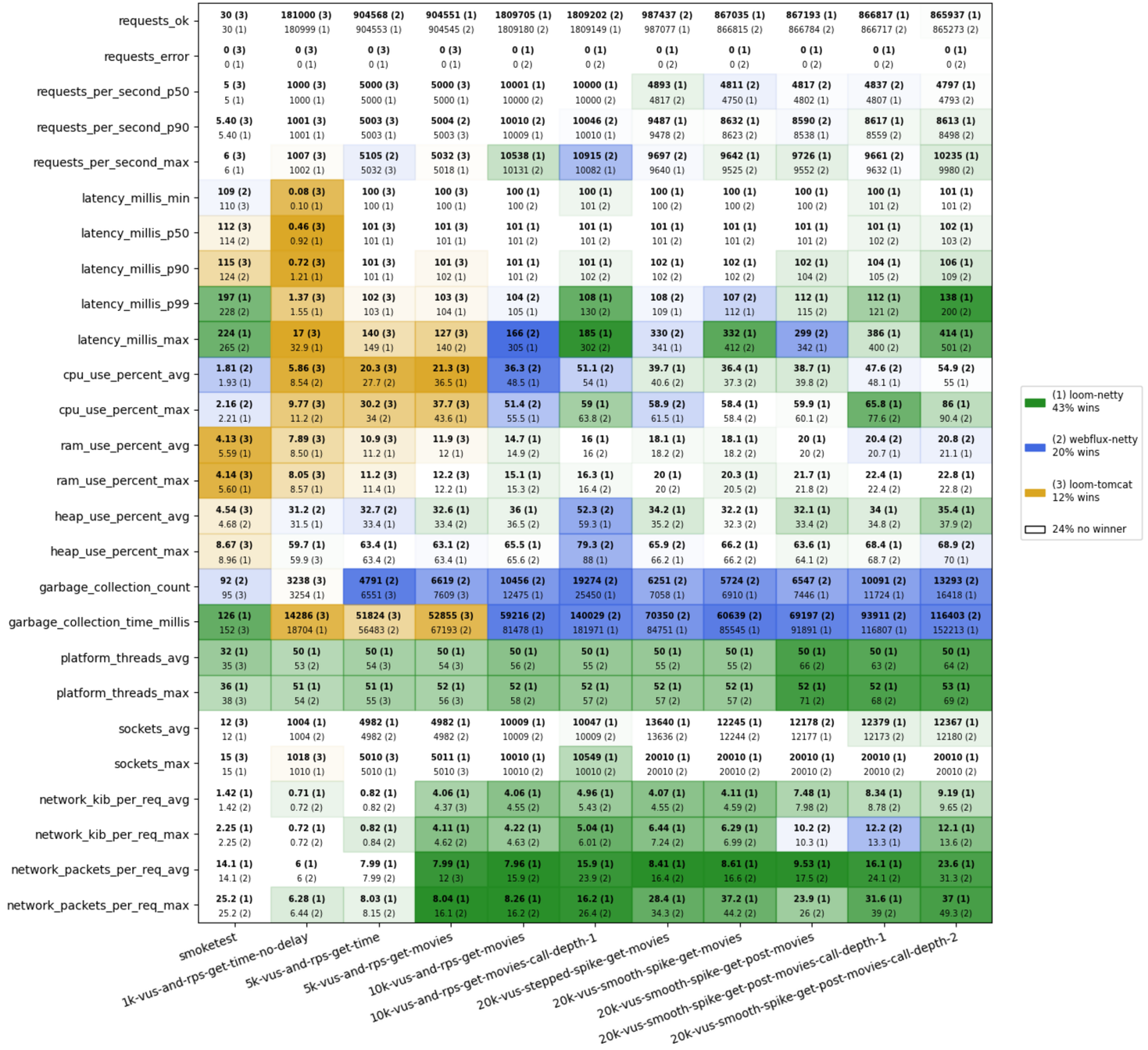
Java Version	Spring Boot Version	Platform Threads (req/s)	Virtual Threads (req/s)	Virtual Thread Error Rate
19	3.3.12	61.07	88.04	23.93%
19	3.5.0	49.92	85.19	29.94%
24	3.5.0	49.01	87.11	28.76%

Key Finding: Under extreme concurrency, we have a significantly higher error rates (23-30%) with Virtual Threads regardless of Java or Spring Boot version.

VTs With Java 24 - Will it Scale?

Best Approaches by Metric and Scenario

Each cell shows metric value for best approach above runner-up. Color saturation is based on win margin.
Approach ranking based on wins is shown in legend and cells. Red values indicate request errors.



	smoketest	1k-vus-and-rps-get-time-no-delay	5k-vus-and-rps-get-time	5k-vus-and-rps-get-movies	10k-vus-and-rps-get-movies	10k-vus-and-rps-get-movies-call-depth-1	20k-vus-stepped-spike-get-movies	20k-vus-smooth-spike-get-movies	20k-vus-smooth-spike-get-post-movies	20k-vus-smooth-spike-get-post-movies-call-depth-1	20k-vus-smooth-spike-get-post-movies-call-depth-2
requests_ok	**30 (3)** 30 (1)	**181000 (3)** 180999 (1)	**904568 (2)** 904553 (1)	**904551 (1)** 904545 (2)	**1809705 (1)** 1809180 (2)	**1809202 (2)** 1809149 (1)	**987437 (2)** 987077 (1)	**867035 (1)** 866815 (2)	**867193 (1)** 866784 (2)	**866817 (1)** 866717 (2)	**865937 (1)** 865273 (2)
requests_error	**0 (3)** 0 (1)	**0 (3)** 0 (1)	**0 (3)** 0 (1)	**0 (3)** 0 (1)	**0 (1)** 0 (2)	**0 (1)** 0 (2)	**0 (1)** 0 (2)	**0 (1)** 0 (2)	**0 (1)** 0 (2)	**0 (1)** 0 (2)	**0 (1)** 0 (2)
requests_per_second_p50	**5 (3)** 5 (1)	**1000 (3)** 1000 (1)	**5000 (3)** 5000 (1)	**5000 (3)** 5000 (1)	**10001 (1)** 10000 (2)	**10000 (1)** 10000 (2)	**4893 (1)** 4817 (2)	**4811 (2)** 4750 (1)	**4817 (2)** 4802 (1)	**4837 (2)** 4807 (1)	**4797 (1)** 4793 (2)
requests_per_second_p90	**5.40 (3)** 5.40 (1)	**1001 (3)** 1001 (1)	**5003 (3)** 5003 (1)	**5004 (2)** 5003 (3)	**10010 (2)** 10009 (1)	**10046 (2)** 10010 (1)	**9487 (1)** 9478 (2)	**8632 (1)** 8623 (2)	**8590 (2)** 8538 (1)	**8617 (1)** 8559 (2)	**8613 (1)** 8498 (2)
requests_per_second_max	**6 (3)** 6 (1)	**1007 (3)** 1002 (1)	**5105 (2)** 5032 (3)	**5032 (3)** 5018 (1)	**10538 (1)** 10131 (2)	**10915 (2)** 10082 (1)	**9697 (2)** 9640 (1)	**9642 (1)** 9525 (2)	**9726 (1)** 9552 (2)	**9661 (2)** 9632 (1)	**10235 (1)** 9980 (2)
latency_millis_min	**109 (2)** 110 (3)	**0.08 (3)** 0.10 (1)	**100 (3)** 100 (1)	**100 (3)** 100 (1)	**100 (1)** 100 (2)	**100 (1)** 101 (2)	**100 (1)** 100 (2)	**100 (1)** 100 (2)	**100 (1)** 100 (2)	**100 (1)** 101 (2)	**101 (1)** 101 (2)
latency_millis_p50	**112 (3)** 114 (2)	**0.46 (3)** 0.92 (1)	**101 (3)** 101 (1)	**101 (3)** 101 (1)	**101 (1)** 101 (2)	**101 (1)** 101 (2)	**101 (1)** 101 (2)	**101 (1)** 101 (2)	**101 (1)** 101 (2)	**101 (1)** 102 (2)	**102 (1)** 103 (2)
latency_millis_p90	**115 (3)** 124 (2)	**0.72 (3)** 1.21 (1)	**101 (3)** 101 (1)	**101 (3)** 102 (1)	**101 (1)** 101 (2)	**101 (1)** 102 (2)	**102 (1)** 102 (2)	**102 (1)** 102 (2)	**102 (1)** 104 (2)	**104 (1)** 105 (2)	**106 (1)** 109 (2)
latency_millis_p99	**197 (1)** 228 (2)	**1.37 (3)** 1.55 (1)	**102 (3)** 103 (1)	**103 (3)** 104 (1)	**104 (2)** 105 (1)	**108 (1)** 130 (2)	**108 (2)** 109 (1)	**107 (2)** 112 (1)	**112 (1)** 115 (2)	**112 (1)** 121 (2)	**138 (1)** 200 (2)
latency_millis_max	**224 (1)** 265 (2)	**17 (3)** 32.9 (1)	**140 (3)** 149 (1)	**127 (3)** 140 (2)	**166 (2)** 305 (1)	**185 (1)** 302 (2)	**330 (2)** 341 (1)	**332 (1)** 412 (2)	**299 (2)** 342 (1)	**386 (1)** 400 (2)	**414 (1)** 501 (2)
cpu_use_percent_avg	**1.81 (2)** 1.93 (1)	**5.86 (3)** 8.54 (2)	**20.3 (3)** 27.7 (2)	**21.3 (3)** 36.5 (1)	**36.3 (2)** 48.5 (1)	**51.1 (2)** 54 (1)	**39.7 (1)** 40.6 (2)	**36.4 (1)** 37.3 (2)	**38.7 (1)** 39.8 (2)	**47.6 (2)** 48.1 (1)	**54.9 (2)** 55 (1)
cpu_use_percent_max	**2.16 (2)** 2.21 (1)	**9.77 (3)** 11.2 (2)	**30.2 (3)** 34 (2)	**37.7 (3)** 43.6 (1)	**51.4 (2)** 55.5 (1)	**59 (1)** 63.8 (2)	**58.9 (2)** 61.5 (1)	**58.4 (1)** 58.4 (2)	**59.9 (1)** 60.1 (2)	**65.8 (1)** 77.6 (2)	**86 (1)** 90.4 (2)
ram_use_percent_avg	**4.13 (3)** 5.59 (1)	**7.89 (3)** 8.50 (1)	**10.9 (3)** 11.2 (1)	**11.9 (3)** 12 (1)	**14.7 (1)** 14.9 (2)	**16 (1)** 16 (2)	**18.1 (1)** 18.2 (2)	**18.1 (1)** 18.2 (2)	**20 (1)** 20 (2)	**20.4 (2)** 20.7 (1)	**20.8 (2)** 21.1 (1)
ram_use_percent_max	**4.14 (3)** 5.60 (1)	**8.05 (3)** 8.57 (1)	**11.2 (3)** 11.4 (1)	**12.2 (3)** 12.2 (1)	**15.1 (1)** 15.3 (2)	**16.3 (1)** 16.4 (2)	**20 (1)** 20 (2)	**20.3 (1)** 20.5 (2)	**21.7 (1)** 21.8 (2)	**22.4 (1)** 22.4 (2)	**22.8 (1)** 22.8 (2)
heap_use_percent_avg	**4.54 (3)** 4.68 (2)	**31.2 (2)** 31.5 (1)	**32.7 (2)** 33.4 (1)	**32.6 (1)** 33.4 (2)	**36 (1)** 36.5 (2)	**52.3 (2)** 59.3 (1)	**34.2 (1)** 35.2 (2)	**32.2 (1)** 32.3 (2)	**32.1 (1)** 33.4 (2)	**34 (1)** 34.8 (2)	**35.4 (1)** 37.9 (2)
heap_use_percent_max	**8.67 (3)** 8.96 (1)	**59.7 (1)** 59.9 (3)	**63.4 (1)** 63.4 (2)	**63.1 (2)** 63.4 (1)	**65.5 (1)** 65.6 (2)	**79.3 (2)** 88 (1)	**65.9 (2)** 66.2 (1)	**66.2 (1)** 66.2 (2)	**63.6 (1)** 64.1 (2)	**68.4 (1)** 68.7 (2)	**68.9 (2)** 70 (1)
garbage_collection_count	**92 (2)** 95 (3)	**3238 (3)** 3254 (1)	**4791 (2)** 6551 (3)	**6619 (2)** 7609 (3)	**10456 (2)** 12475 (1)	**19274 (2)** 25450 (1)	**6251 (2)** 7058 (1)	**5724 (2)** 6910 (1)	**6547 (2)** 7446 (1)	**10091 (2)** 11724 (1)	**13293 (2)** 16418 (1)
garbage_collection_time_millis	**126 (1)** 152 (3)	**14286 (3)** 18704 (1)	**51824 (3)** 56483 (2)	**52855 (3)** 67193 (2)	**59216 (2)** 81478 (1)	**140029 (2)** 181971 (1)	**70350 (2)** 84751 (1)	**60639 (2)** 85545 (1)	**69197 (2)** 91891 (1)	**93911 (2)** 116807 (1)	**116403 (2)** 152213 (1)
platform_threads_avg	**32 (1)** 35 (3)	**50 (1)** 53 (2)	**50 (1)** 54 (3)	**50 (1)** 54 (3)	**50 (1)** 56 (2)	**50 (1)** 55 (2)	**50 (1)** 55 (2)	**50 (1)** 55 (2)	**50 (1)** 66 (2)	**50 (1)** 63 (2)	**50 (1)** 64 (2)
platform_threads_max	**36 (1)** 38 (3)	**51 (1)** 54 (2)	**51 (1)** 55 (3)	**52 (1)** 56 (3)	**52 (1)** 58 (2)	**52 (1)** 57 (2)	**52 (1)** 57 (2)	**52 (1)** 57 (2)	**52 (1)** 71 (2)	**52 (1)** 68 (2)	**53 (1)** 69 (2)
sockets_avg	**12 (3)** 12 (1)	**1004 (1)** 1004 (2)	**4982 (1)** 4982 (2)	**4982 (1)** 4982 (2)	**10009 (1)** 10009 (2)	**10047 (1)** 10009 (2)	**13640 (1)** 13636 (2)	**12245 (1)** 12244 (2)	**12178 (2)** 12177 (1)	**12379 (1)** 12173 (2)	**12367 (1)** 12180 (2)
sockets_max	**15 (3)** 15 (1)	**1018 (3)** 1010 (1)	**5010 (3)** 5010 (1)	**5011 (1)** 5010 (3)	**10010 (1)** 10010 (2)	**10549 (1)** 10010 (2)	**20010 (1)** 20010 (2)	**20010 (1)** 20010 (2)	**20010 (1)** 20010 (2)	**20010 (1)** 20010 (2)	**20010 (1)** 20010 (2)
network_kib_per_req_avg	**1.42 (1)** 1.42 (2)	**0.71 (1)** 0.72 (2)	**0.82 (1)** 0.82 (2)	**4.06 (1)** 4.37 (3)	**4.06 (1)** 4.55 (2)	**4.96 (1)** 5.43 (2)	**4.07 (1)** 4.55 (2)	**4.11 (1)** 4.59 (2)	**7.48 (1)** 7.98 (2)	**8.34 (1)** 8.78 (2)	**9.19 (1)** 9.65 (2)
network_kib_per_req_max	**2.25 (1)** 2.25 (2)	**0.72 (1)** 0.72 (2)	**0.82 (1)** 0.84 (2)	**4.11 (1)** 4.62 (2)	**4.22 (1)** 4.63 (2)	**5.04 (1)** 6.01 (2)	**6.44 (1)** 7.24 (2)	**6.29 (1)** 6.99 (2)	**10.2 (2)** 10.3 (1)	**12.2 (2)** 13.3 (1)	**12.1 (1)** 13.6 (2)
network_packets_per_req_avg	**14.1 (1)** 14.1 (2)	**6 (1)** 6 (2)	**7.99 (1)** 7.99 (2)	**7.99 (1)** 12 (3)	**7.96 (1)** 15.9 (2)	**15.9 (1)** 23.9 (2)	**8.41 (1)** 16.4 (2)	**8.61 (1)** 16.6 (2)	**9.53 (1)** 17.5 (2)	**16.1 (1)** 24.1 (2)	**23.6 (1)** 31.3 (2)
network_packets_per_req_max	**25.2 (1)** 25.2 (2)	**6.28 (1)** 6.44 (2)	**8.03 (1)** 8.15 (2)	**8.04 (1)** 16.1 (2)	**8.26 (1)** 16.2 (2)	**16.2 (1)** 26.4 (2)	**28.4 (1)** 34.3 (2)	**37.2 (1)** 44.2 (2)	**23.9 (1)** 26 (2)	**31.6 (1)** 39 (2)	**37 (1)** 49.3 (2)

Benchmark VTs vs WebFlux

You can build Virtual Thread-based applications today

But, you have to mindful in the process.

SOFTWAREMILL

Accelerating product development

VIRTUSLAB

Summary

- VT: widely supported in Java frameworks & libraries
- But, rarely the default
- Conscious decision to use
  - not "carefree"

Problems with VT

- `ThreadLocal`s
- Pinning
- Monopolization
- Thread inheritance

Summary cntd

- Often works OOTB
- Make sure your use-case is appropriate
- Reactive is still alive
  - Integrating blocking & non-blocking code is tricky
- Will Java 25 be a breakthrough?

```
println("Thank you!")
```

warski.org