

Virtual Threads, 2 years later

Adam Warski, June 2025

warski.org



You can build Virtual Thread-based
applications today

But, you have to be mindful in the process.

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

PART I

What's a virtual thread?

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](#)

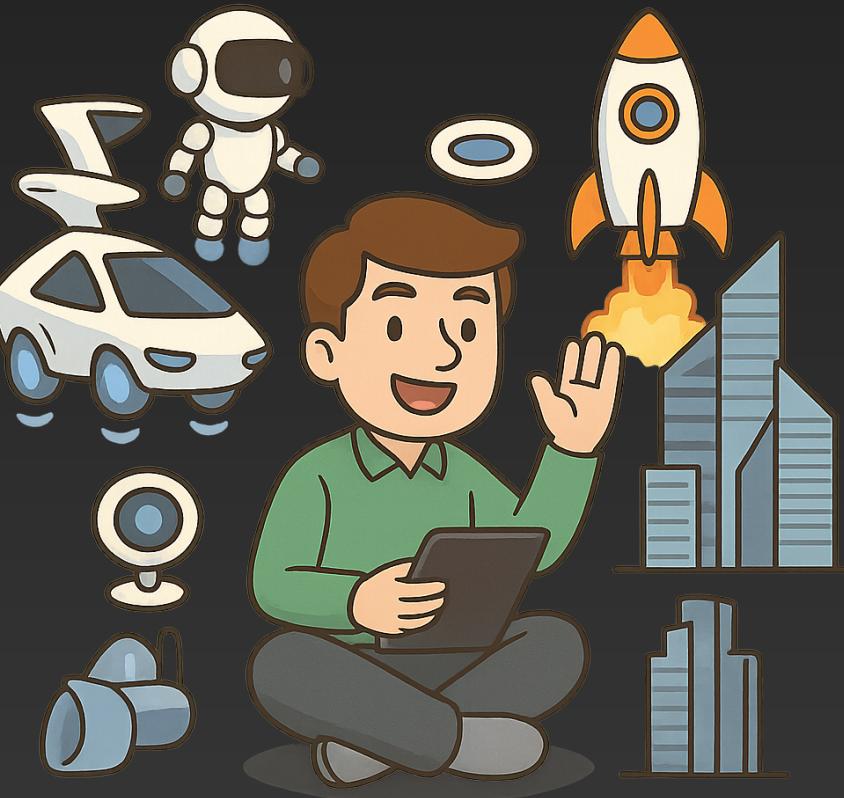
```
timed( () -> {
    var threads = new Thread[10_000_000];
    var results = ConcurrentHashMap.newKeySet();
    for (int i=0; i<threads.length; i++) {
        threads[i] = Thread
            .ofVirtual()
            .start(() -> results.add(0));
    }

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

    return null;
} );
```

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

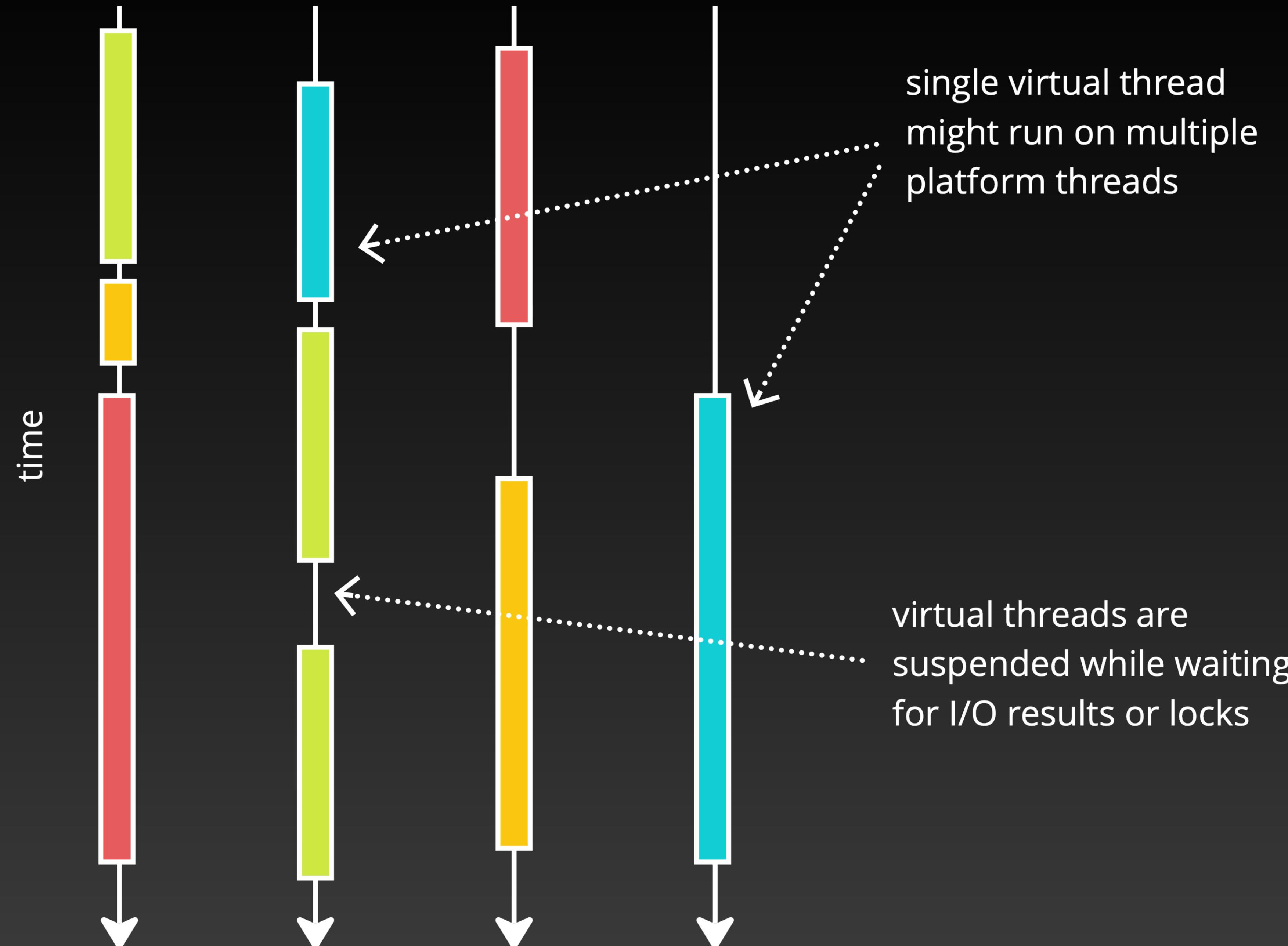
Goals of virtual threads

- maintain thread utilization
- retain simplicity of synchronous code

By:

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

4 cores, 4 platform threads



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);  
}
```

PART II

Evolution of VT & associated JEPs

JDK 21 (LTS, September 2023)

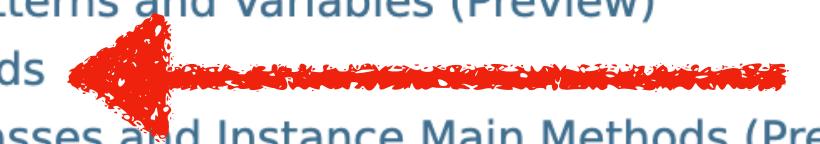
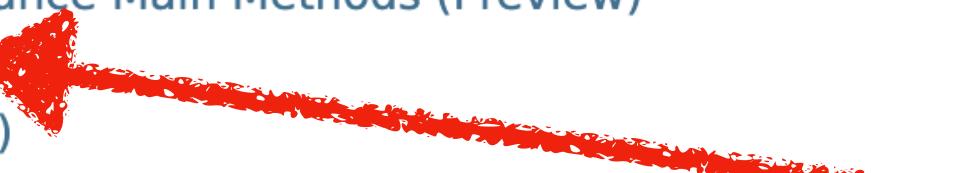
JDK 21

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 \(Preview\)](#) 

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 `sun.misc.Unsafe`
- 499: Structured Concurrency (Fourth Preview)

Java 25 (LTS, September 2025)

JDK 25

This release will be the Reference Implementation of version 25 of the Java SE Platform, as specified by [JSR 400](#) in the Java Community Process.

Schedule

2025/06/05	Rampdown Phase One (branch from main line)
2025/07/17	Rampdown Phase Two
2025/08/07	Initial Release Candidate
2025/08/21	Final Release Candidate
2025/09/16	General Availability

Status

The [main line branch](#) is open for bug fixes, small enhancements, and JEPs as proposed and tracked via the [JEP Process](#).

Features

JEPs proposed to target JDK 25

		review ends
470: PEM Encodings of Cryptographic Objects (Preview)		2025/05/27
518: JFR Cooperative Sampling		2025/05/26
520: JFR Method Timing & Tracing		2025/05/26
521: Generational Shenandoah		2025/05/27

JEPs targeted to JDK 25, so far

502: Stable Values (Preview)
503: Remove the 32-bit x86 Port
505: Structured Concurrency (Fifth Preview)
506: Scoped Values



Pinning

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

- **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](#)

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
- Thread locals: often used to cache expensive objects
- Inherited

Structured concurrency

```
Response handle() throws EE, IE {  
    try (var scope = new StructuredTaskScope.ShutdownOnFailure()) {  
        Supplier<String> user = scope.fork(() -> findUser());  
        Supplier<Integer> order = scope.fork(() -> fetchOrder());  
  
        scope.join()  
            .throwIfFailed();  
  
        return new Response(user.get(), order.get());  
    }  
}
```



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());  
    }  
}
```



[Jox library](#)

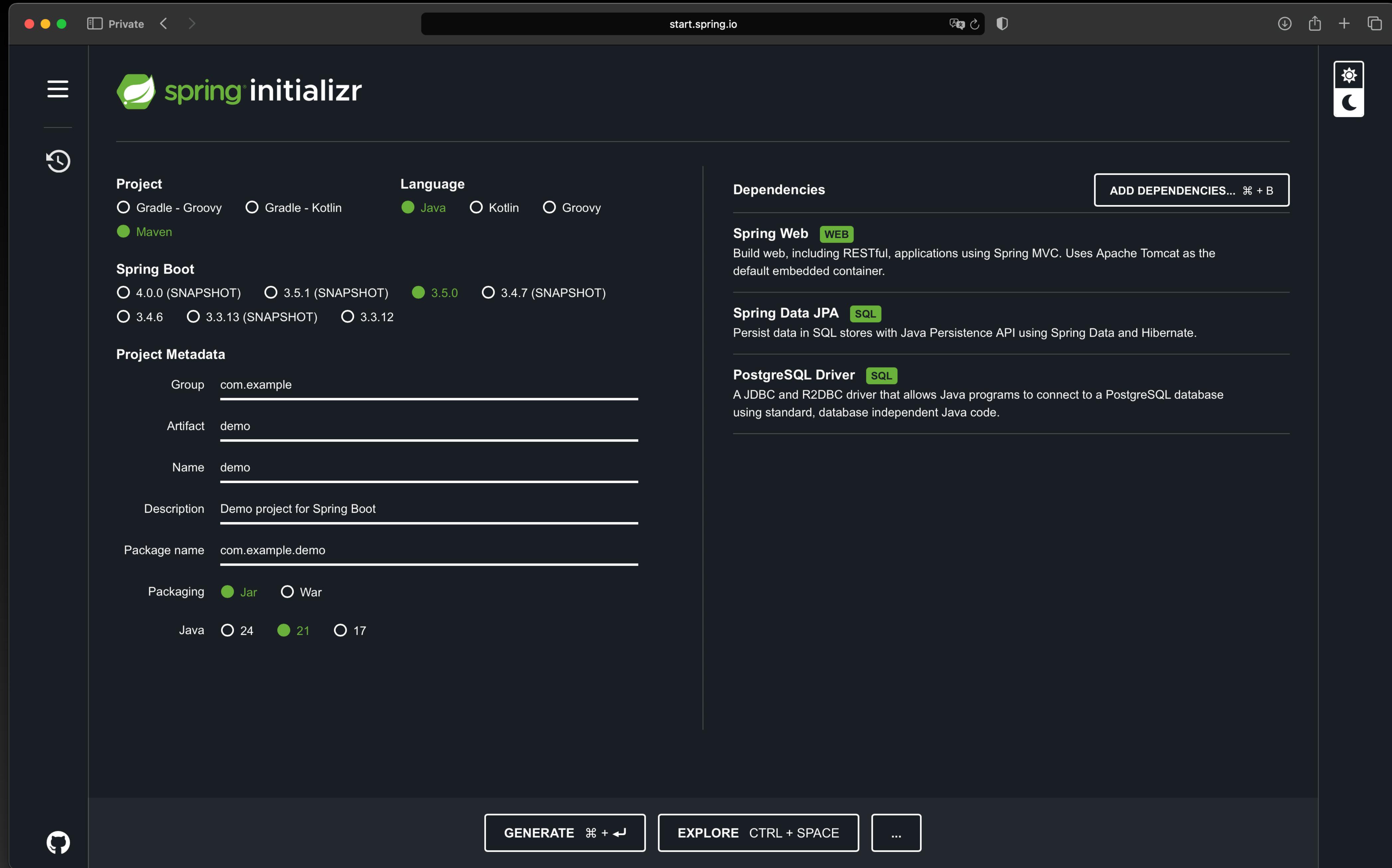


[Java 25 structured concurrency JEP](#)

```
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());  
    }) ;  
}
```

PART III

Web frameworks



```
@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());
    }

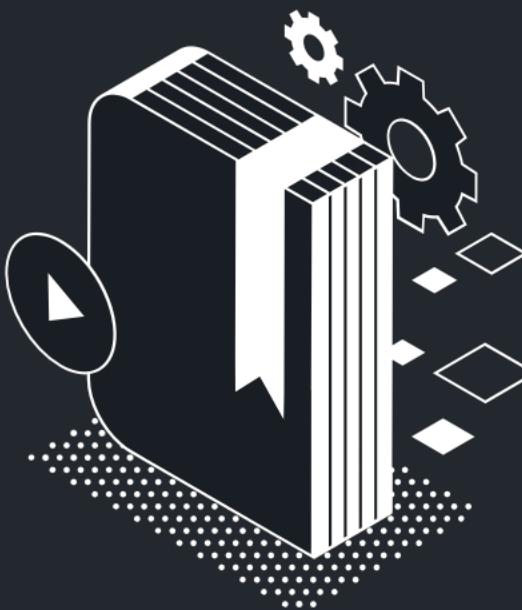
    @GetMapping("/test")
    public String test(@RequestParam(value = "id") Long id) {
        var result = customerRepository.findById(id);
        logThread();

        // ...
    }
}
```

```
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] [tomcat-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] [tomcat-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] [tomcat-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](#)

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

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, ThreadLocals
 - Java ecosystem needs to be come VT-friendly
 - VT useful for I/O-bound workloads only



[Quarkus VT guide](#)



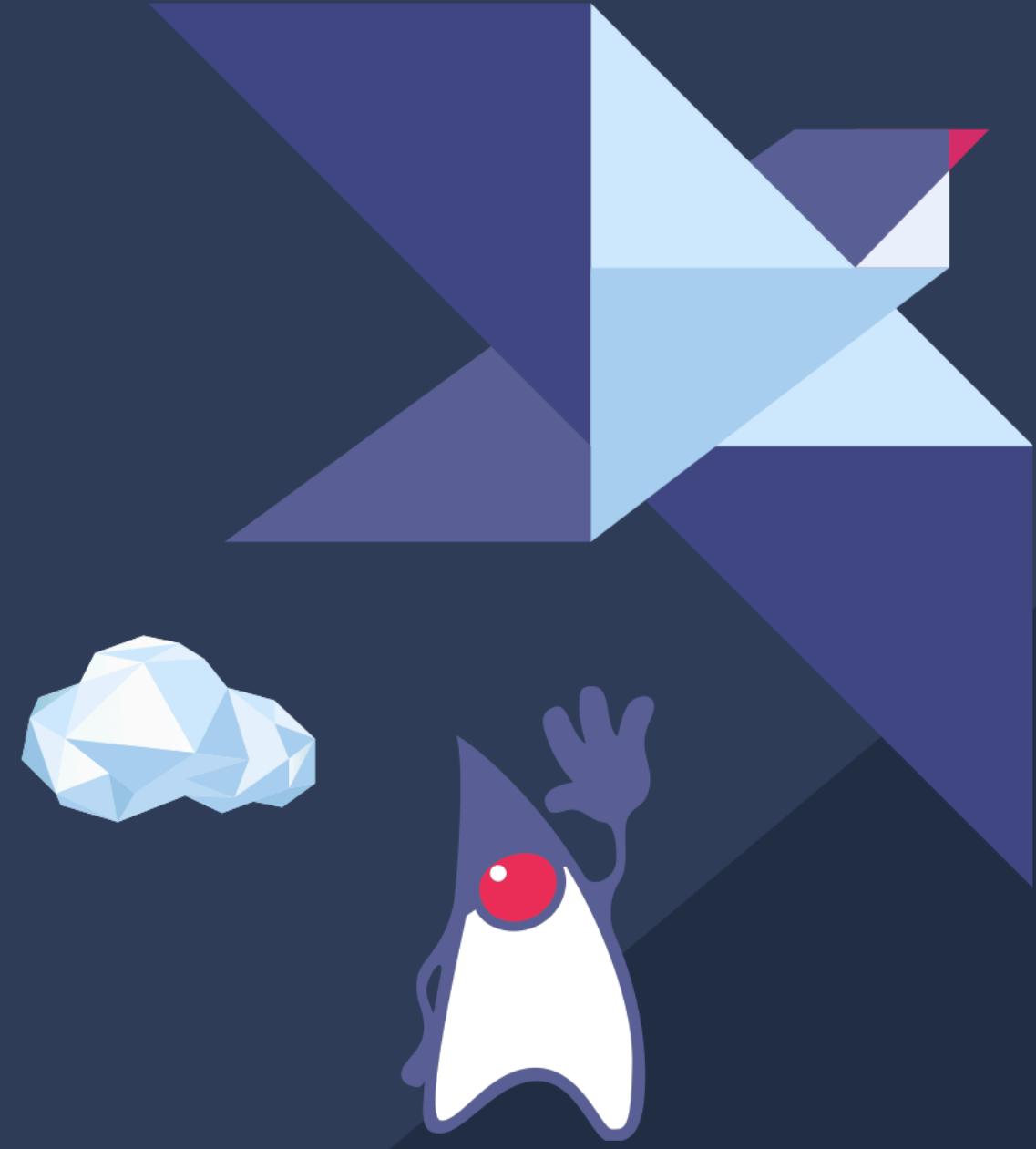
Works OOTB

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

[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 [[0x1645022f 0x3dd9164b] WebServer socket] INFO demo.Main -- Thread name: [0x1645022f 0x3dd9164b] WebServer socket, id: 37, isVirtual: true
16:12:02.596 [[0x1645022f 0x3dd9164b] WebServer socket] DEBUG io.helidon.webserver.http1.Http1LoggingConnectionListener.send -- [0x1645022f 0x3dd9164b] 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)

Apache Tomcat 10 Configuration Reference

Version 10.1.41, May 8 2025

The Executor (thread pool)

Table of Contents

- [Introduction](#)
- [Attributes](#)
 1. [Common Attributes](#)
 2. [Standard Implementation](#)
 3. [Virtual Thread Implementation](#)



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

- threading model-agnostic
- no priority
- subclasses of Thread cannot be virtual
- compatibility concerns
- ThreadLocals
- deeper integration possible in v12



PART IV

Libraries

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 EE, IE {
    try (var scope = new StructuredTaskScope.ShutdownOnFailure()) {
        Supplier<String> user = scope.fork(() -> findUser());
        Supplier<Integer> order = scope.fork(() -> fetchOrder());

        scope.join()
            .throwIfFailed();

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





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

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

        scope.join()
            .throwIfFailed();

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

Logging + MDC

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

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



[Inheritable MDC source](#)

Reactive

RxJava	Integration module available (<code>RxJavaFiberInterop</code>), doesn't seem used much Possibility to use VT-executor as a scheduler Await on Futures
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 Futures
Netty	Use a VT-based executor Await on Futures

Streaming

- Java streams
- Stream gatherers (JEP 485)
- Pull-type API
- Jox Flows
- Push-type API
- Reactive streams-compatible

```
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](#)

```
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();
```



[Jox Docs](#)

PART V

Summary



Accelerating product development



Summary

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

Problems with VT

- ThreadLocals
- 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