MY FAVORITE SPRING BOOT 3 FEATURES

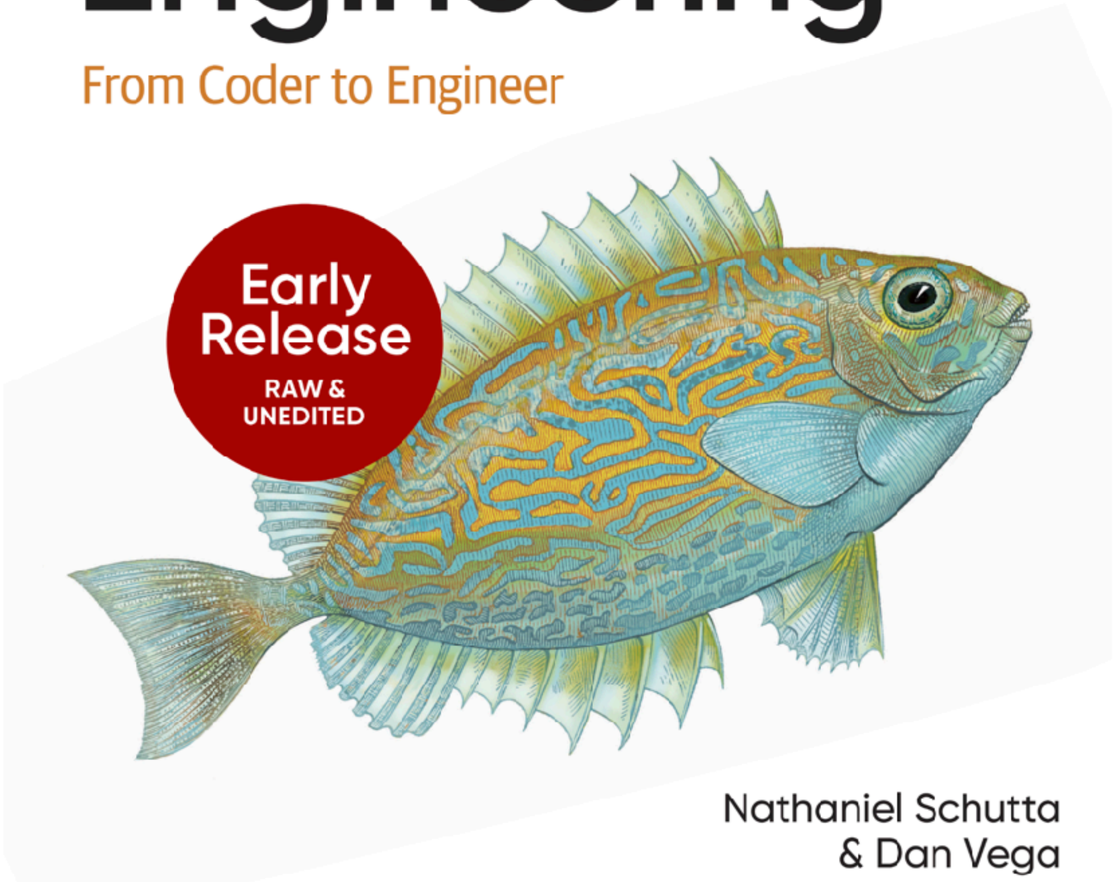
And a look ahead to Spring Boot 4

Dan Vega - Spring Developer Advocate @Broadcom

ABOUT ME

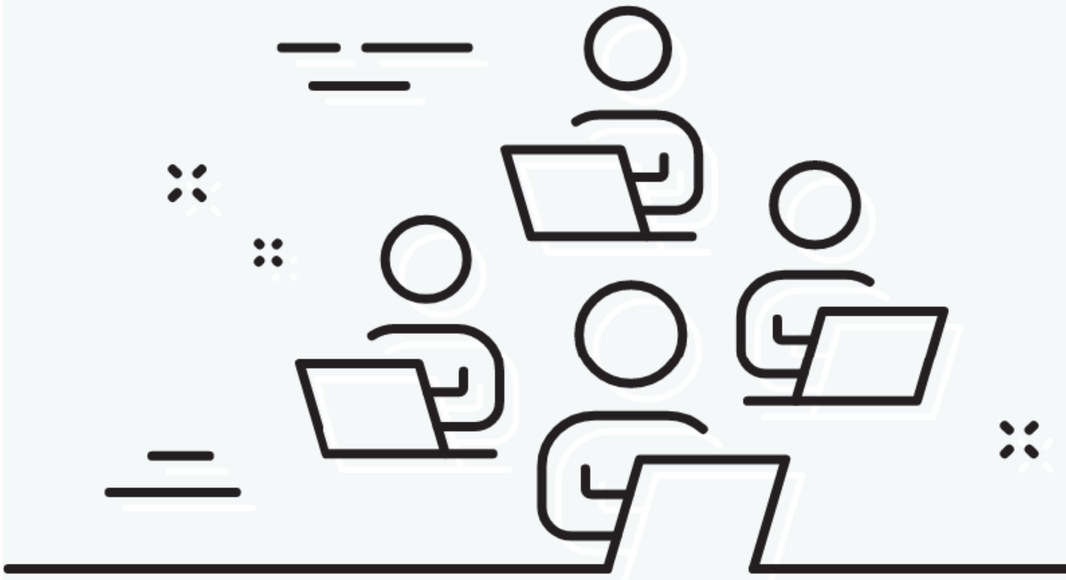
Learn more at danvega.dev

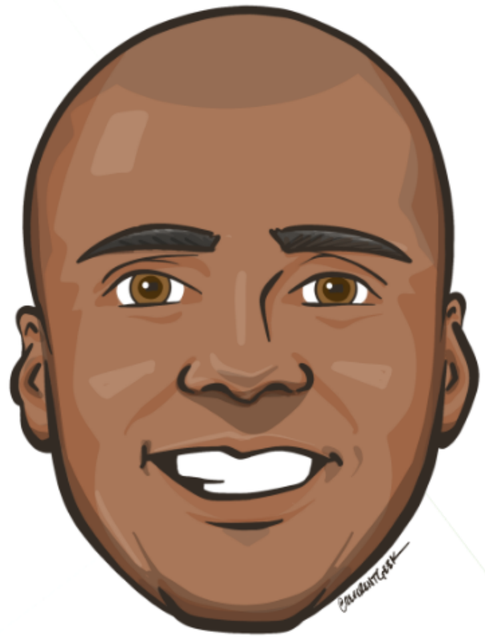
- 👨‍👩‍👧‍👦 Husband & Father
- 🏠 Cleveland
- ☕ Java Champion
- 👨‍💻 Software Development 23 Years
- 🍃 Spring Developer Advocate
- 📖 Author (Soon to be)



spring

OFFICE HOURS





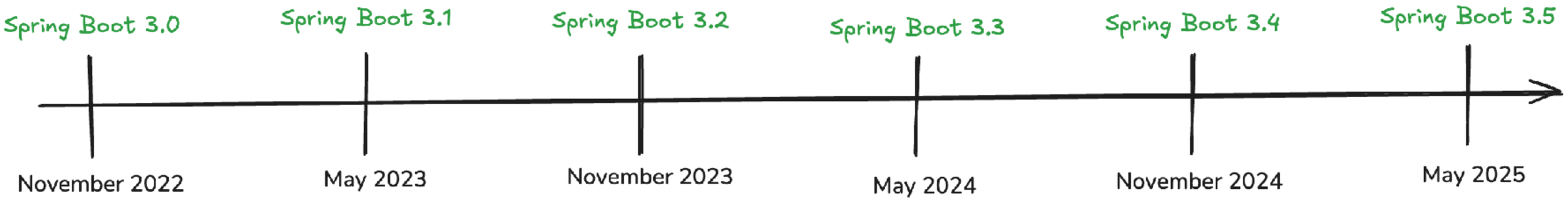
https://www.springofficehours.io

AGENDA

What are we going to cover

- Spring Boot 3.x & Framework 6.x
- Spring Boot 4.x & Framework 7.x
- Roadmap

SPRING FRAMEWORK 6.X & SPRING BOOT 3.X


Spring Boot 3.0
November 2022
Spring Boot 3.1
May 2023
Spring Boot 3.2
November 2023
Spring Boot 3.3
May 2024
Spring Boot 3.4
November 2024
Spring Boot 3.5
May 2025

spring

CURRENT: SPRING FRAMEWORK 6.X & SPRING BOOT 3.X

Spring Boot 3.0 (November 2022)

- JDK 17+
- Jakarta EE 9/10
- Ahead-of-Time (AOT) Compilation (GraalVM)
- Observability
- HTTP Interface Clients

AHEAD-OF-TIME (AOT)

“Spring’s support for AOT optimizations is meant to inspect an Application Context at build time and apply decisions and discovery logic that usually happens at runtime. Doing so allows building an application startup arrangement that is more straightforward and focused on a fixed set of features based mainly on the classpath and the Environment.”

`https://docs.spring.io/spring-framework/reference/core/aot.html`

AOT COMPILATION

What is it?

Reducing startup time and **memory footprint** in production

Optional for optimized JVM deployment

Precondition for GraalVM native executables

Runtime hints for reflection, resources, serialization, proxies

AOT is a tradeoff; extra build setup and less flexibility at runtime

WHY COMPILE TO NATIVE?

Instant Startup

Milliseconds for native instead of seconds for the JVM

No Warmup

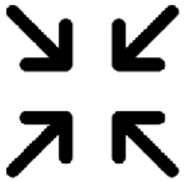
Performance benefits available immediately

Low Resource Usage

Lower Memory consumption no JIT compilation

Reduced Attack Surface

Closed world of dependencies with explicit reflection



Compact Packaging

Smaller containers and easier to deploy

Lower Compute Costs

Reduce your infrastructure costs

spring

USE CASES FOR NATIVE IMAGES

JVM Likely Better	Assess	Reccomended
• High Traffic Websites	• Microservices	• Low Memory & CPU
• Huge Memory and CPU Needs	• Container Images	• Function as a Service
• Frequent Deployments	• Kubernetes	• Scale to Zero
• Big Monolithic App	• Backoffice Applications	• CLI's

spring

OBSERVABILITY

WHAT IS OBSERVABILITY?

Observability is the ability to observe the internal state of a running system from the outside. It consists of the three pillars **logging**, **metrics** and **traces**.

Direct Observability instrumentation with **Micrometer Observation** in several parts of the Spring Framework.

RestTemplate, **WebClient, RestClient, ChatClient** are instrumented to produce HTTP client request observations.

Spring Cloud Sleuth no longer needed for **Distributed Tracing**


{API}
</>
{...}

OBSERVATION API

You might want to emit log messages

You might want to start / stop timers

Increment counter

You want to start / stop span

Signal error

If you want to do metrics and tracing you need to instrument your application twice

That is a lot of boiler plate

```java
@SpringBootApplication
public class Application {

    private static final Logger log = LoggerFactory.getLogger(Application.class);

    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }

    @Bean
    CommandLineRunner commandLineRunner(ObservationRegistry registry) {
        return args -> {
            // Create an Observation and observe your code!
            Observation.createNotStarted( name: "user.name", registry)
                    .contextualName( s: "getting-user-name")
                    // let's assume that you can have 3 user types
                    .lowCardinalityKeyValue("userType", "userType1")
                    // let's assume that this is an arbitrary number
                    .highCardinalityKeyValue("userId", "1234")
                    // this is a shortcut for starting an observation, opening a scope,
                    // running user's code, closing the scope and stopping the observation
                    .observe(() -> log.info("Hello"));
        };
    }
}
```

```java
@SpringBootApplication
public class Application {

    private static final Logger log = LoggerFactory.getLogger(Application.class);

    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }

    @Bean
    @Observed(name = "command-line-runner")
    CommandLineRunner commandLineRunner(ObservationRegistry registry) {
        return args -> {
            log.info("Hello");
        };
    }
}
```

DEMO TIME

CURRENT: SPRING FRAMEWORK 6.X & SPRING BOOT 3.X

Spring Boot 3.1 (May 2023)

- Docker Compose
- Testcontainers
- Spring Authorization Server

DOCKER COMPOSE

SOME REALLY COOL PROJECT

PostgreSQL 16.0 on port 5432

Kafka 3.4.1 on port 9092

Redis 6.0.20 on port 6379

Elasticsearch 8.8.2 in a 3 node cluster configuration on port 8201

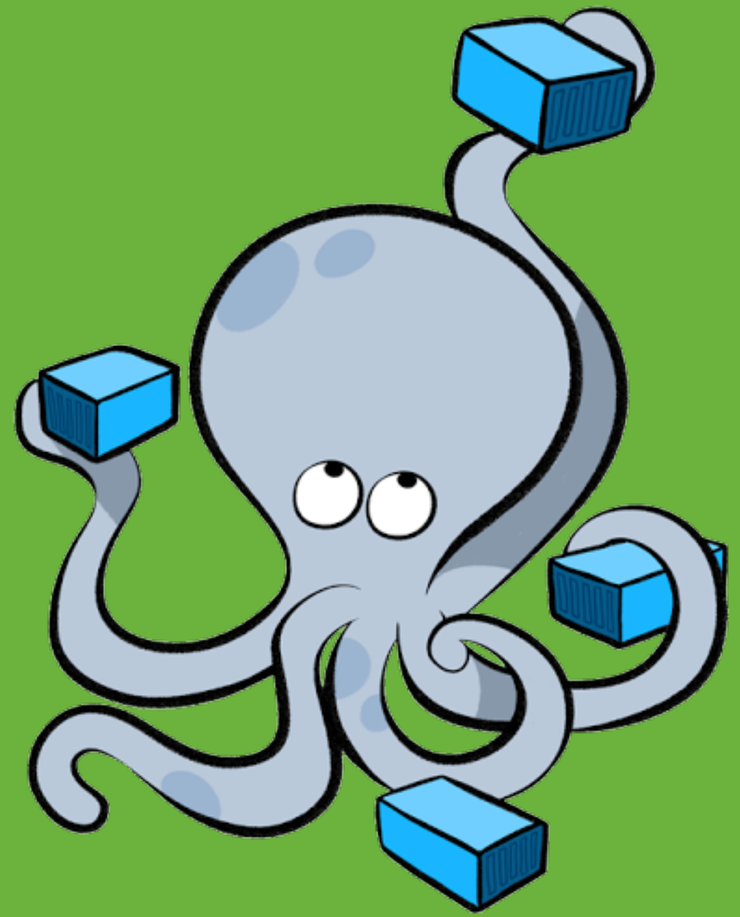
MongoDB 6.0.8 in a 3 node cluster configuration on port 27017

Microservices

- The customer-service running on port 8081
- The product-service running on port 8082
- The order-service running on port 8083


git
RUN

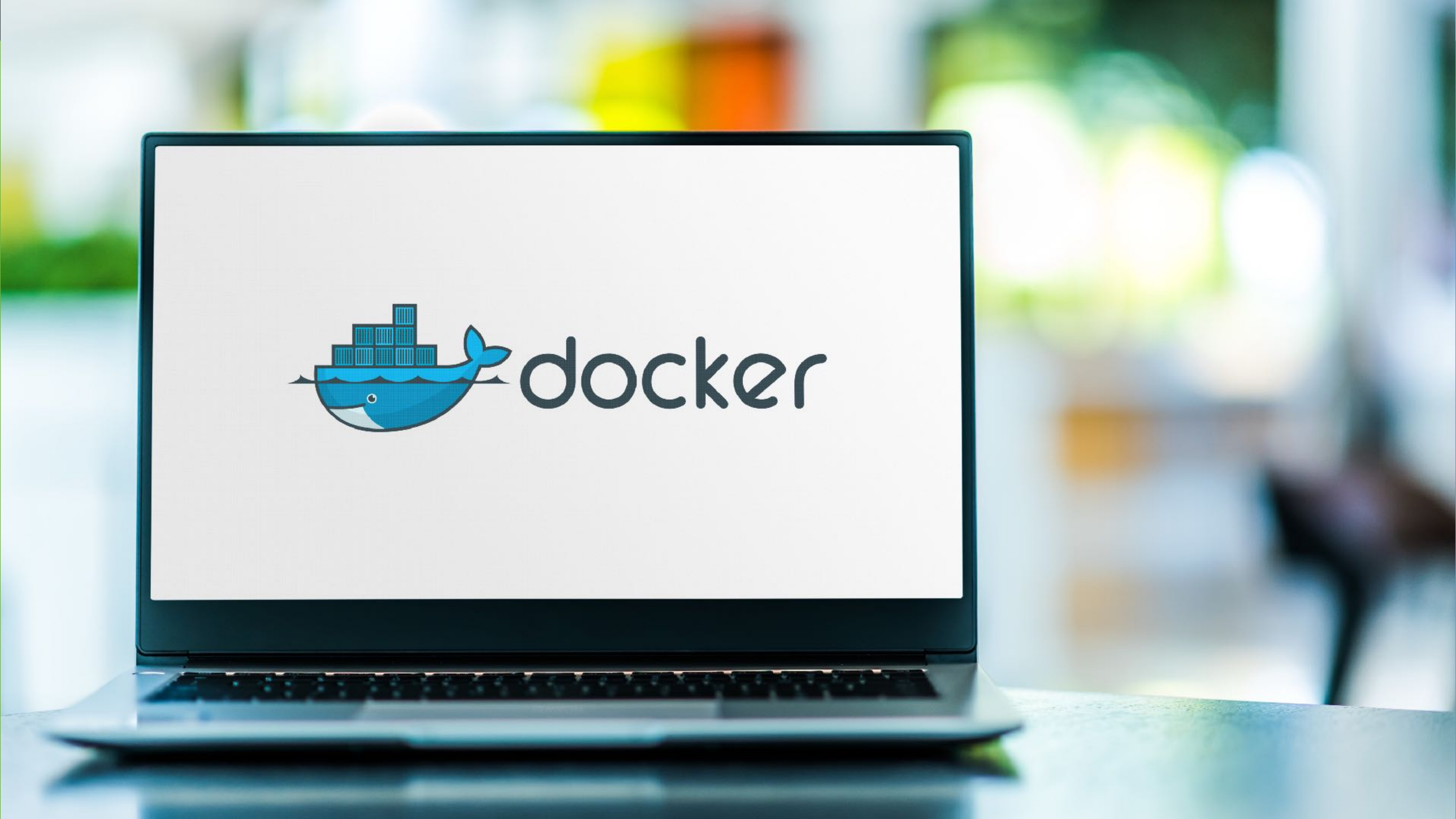
DEVELOPER EXPERIENCE

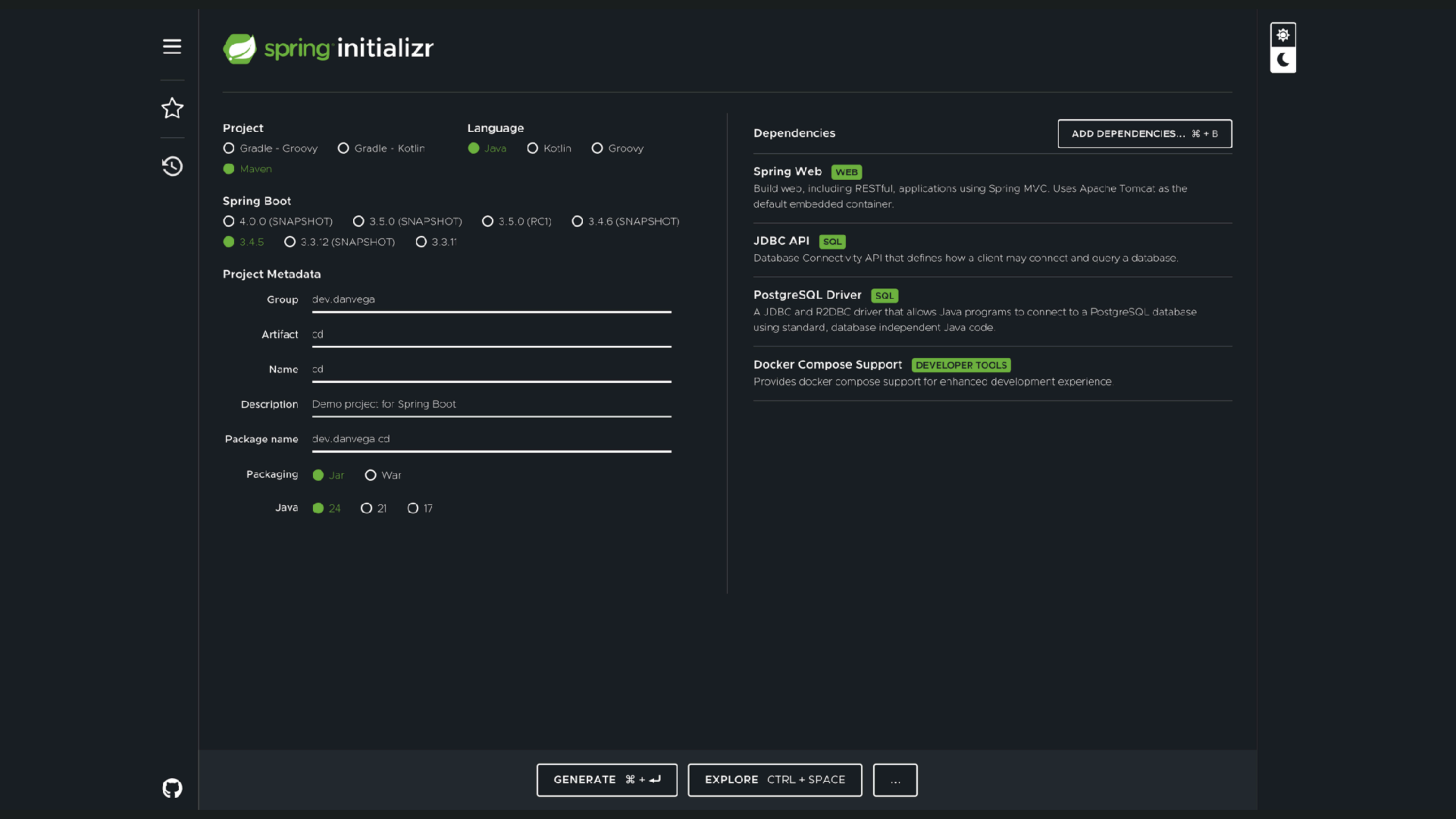


Docker Compose

Testcontainers

Connection Details


docker


spring initializr
Project
Gradle - Groovy
Gradle - Kotlin
Maven
Language
Java
Kotlin
Groovy
Spring Boot
4.0.0 (SNAPSHOT)
3.5.0 (SNAPSHOT)
3.5.0 (RC1)
3.4.6 (SNAPSHOT)
3.4.5
3.3.12 (SNAPSHOT)
3.3.11
Project Metadata
Group
dev.danvega
Artifact
cd
Name
cd
Description
Demo project for Spring Boot
Package name
dev.danvega.cd
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.
JDBC API
SQL
Database Connectivity API that defines how a client may connect and query a database.
PostgreSQL Driver
SQL
A JDBC and R2DBC driver that allows Java programs to connect to a PostgreSQL database using standard, database independent Java code.
Docker Compose Support
DEVELOPER TOOLS
Provides docker compose support for enhanced development experience.
GENERATE ⌘ + ↵
EXPLORE CTRL + SPACE
...




cd.zip

- .gitattributes
- .gitignore
- .mvn
- HELP.md
- compose.yaml
- mvnw
- mvnw.cmd
- pom.xml
- src

DOWNLOAD COPY

```yaml
services:
  postgres:
    image: 'postgres:latest'
    environment:
      - 'POSTGRES_DB=mydatabase'
      - 'POSTGRES_PASSWORD=secret'
      - 'POSTGRES_USER=myuser'
    ports:
      - '5432'

```

DOWNLOAD ⌘ + ↵ CLOSE ESC

```
  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::                (v3.2.0-RC1)

2023-11-01T11:19:57.979-04:00  INFO 76999 --- [           main] dev.danvega.cd.CdApplication             : Starting CdApplication using Java 21 with PID 76999 (/Users/vega/Downloads/cd/target/classes
2023-11-01T11:19:57.980-04:00  INFO 76999 --- [           main] dev.danvega.cd.CdApplication             : No active profile set, falling back to 1 default profile: "default"
2023-11-01T11:19:58.009-04:00  INFO 76999 --- [           main] .s.b.d.c.l.DockerComposeLifecycleManager : Using Docker Compose file '/Users/vega/Downloads/cd/compose.yaml'
2023-11-01T11:20:15.179-04:00  INFO 76999 --- [utReader-stderr] o.s.boot.docker.compose.core.DockerCli    :  Network cd_default  Creating
2023-11-01T11:20:15.204-04:00  INFO 76999 --- [utReader-stderr] o.s.boot.docker.compose.core.DockerCli    :  Network cd_default  Created
2023-11-01T11:20:15.204-04:00  INFO 76999 --- [utReader-stderr] o.s.boot.docker.compose.core.DockerCli    :  Container cd-postgres-1  Creating
2023-11-01T11:20:15.231-04:00  INFO 76999 --- [utReader-stderr] o.s.boot.docker.compose.core.DockerCli    :  Container cd-postgres-1  Created
2023-11-01T11:20:15.233-04:00  INFO 76999 --- [utReader-stderr] o.s.boot.docker.compose.core.DockerCli    :  Container cd-postgres-1  Starting
2023-11-01T11:20:15.409-04:00  INFO 76999 --- [utReader-stderr] o.s.boot.docker.compose.core.DockerCli    :  Container cd-postgres-1  Started
2023-11-01T11:20:15.409-04:00  INFO 76999 --- [utReader-stderr] o.s.boot.docker.compose.core.DockerCli    :  Container cd-postgres-1  Waiting
2023-11-01T11:20:15.916-04:00  INFO 76999 --- [utReader-stderr] o.s.boot.docker.compose.core.DockerCli    :  Container cd-postgres-1  Healthy
2023-11-01T11:20:17.521-04:00  INFO 76999 --- [           main] .s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data JDBC repositories in DEFAULT mode.
2023-11-01T11:20:17.528-04:00  INFO 76999 --- [           main] .s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository scanning in 5 ms. Found 0 JDBC repository interfaces.
2023-11-01T11:20:17.708-04:00  INFO 76999 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port 8080 (http)
2023-11-01T11:20:17.712-04:00  INFO 76999 --- [           main] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
2023-11-01T11:20:17.712-04:00  INFO 76999 --- [           main] o.apache.catalina.core.StandardEngine    : Starting Servlet engine: [Apache Tomcat/10.1.15]
2023-11-01T11:20:17.732-04:00  INFO 76999 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
2023-11-01T11:20:17.733-04:00  INFO 76999 --- [           main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 442 ms
2023-11-01T11:20:17.923-04:00  INFO 76999 --- [           main] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Starting...
2023-11-01T11:20:18.017-04:00  INFO 76999 --- [           main] com.zaxxer.hikari.pool.HikariPool        : HikariPool-1 - Added connection org.postgresql.jdbc.PgConnection@1213ffbc
2023-11-01T11:20:18.017-04:00  INFO 76999 --- [           main] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Start completed.
2023-11-01T11:20:18.093-04:00  INFO 76999 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port 8080 (http) with context path ''
2023-11-01T11:20:18.099-04:00  INFO 76999 --- [           main] dev.danvega.cd.CdApplication             : Started CdApplication in 20.271 seconds (process running for 20.598)
```

WHAT DOES THIS MEAN FOR ME?

You don't have to remember to run **docker compose up** before starting the application.

If the services are already running, it will detect that, too, and will use them.

The images started by Docker Compose are automatically detected and used to create **ConnectionDetails** beans pointing to the services.

That means you don’t have to put properties in your configuration, you don’t have to remember how to construct PostgreSQL JDBC URLs, and so on.

compose.yaml

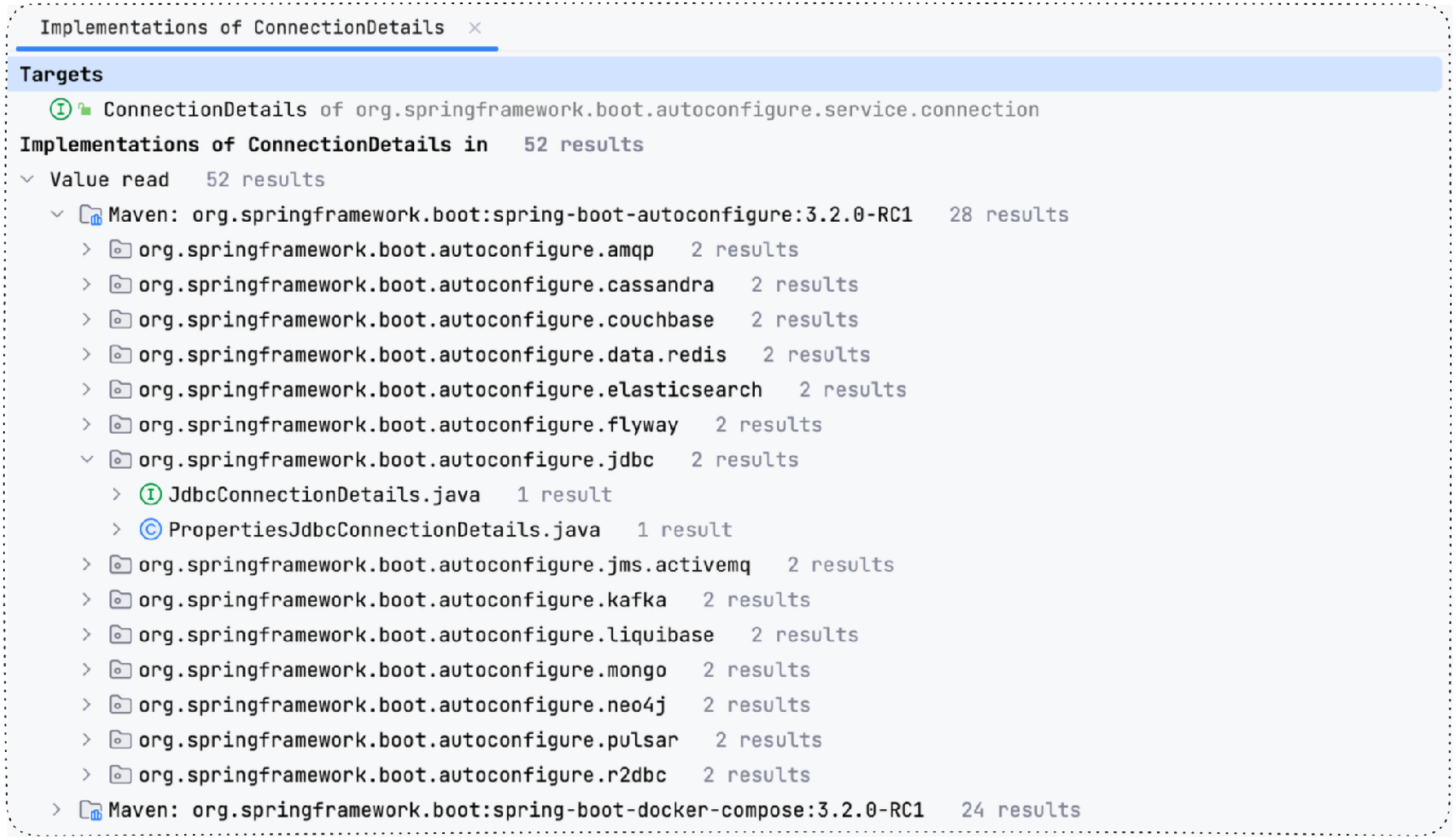
```yaml
services:
  postgres:
    image: 'postgres:latest'
    environment:
      - 'POSTGRES_DB=mydatabase'
      - 'POSTGRES_PASSWORD=secret'
      - 'POSTGRES_USER=myuser'
    ports:
      - '5432'
```

application.properties

```properties
spring.datasource.url=jdbc:postgresql://localhost:32771/mydatabase
spring.datasource.username=myuser
spring.datasource.password=secret
```

ConnectionDetails.java

```
1   /.../

16

17  package org.springframework.boot.autoconfigure.service.connection;

18

19  import org.springframework.boot.origin.OriginProvider;

20

        Base interface for types that provide the details required to
        establish a connection to a remote service.

        Implementation classes can also implement OriginProvider in order to
        provide origin information.

        Since: 3.1.0

        Author: Moritz Halbritter, Andy Wilkinson, Phillip Webb

33  public interface ConnectionDetails {

34

35  }
```


Implementations of ConnectionDetails
Targets
ConnectionDetails of org.springframework.boot.autoconfigure.service.connection
Implementations of ConnectionDetails in 52 results
Value read 52 results
Maven: org.springframework.boot:spring-boot-autoconfigure:3.2.0-RC1 28 results
org.springframework.boot.autoconfigure.amqp 2 results
org.springframework.boot.autoconfigure.cassandra 2 results
org.springframework.boot.autoconfigure.couchbase 2 results
org.springframework.boot.autoconfigure.data.redis 2 results
org.springframework.boot.autoconfigure.elasticsearch 2 results
org.springframework.boot.autoconfigure.flyway 2 results
org.springframework.boot.autoconfigure.jdbc 2 results
JdbcConnectionDetails.java 1 result
PropertiesJdbcConnectionDetails.java 1 result
org.springframework.boot.autoconfigure.jms.activemq 2 results
org.springframework.boot.autoconfigure.kafka 2 results
org.springframework.boot.autoconfigure.liquibase 2 results
org.springframework.boot.autoconfigure.mongo 2 results
org.springframework.boot.autoconfigure.neo4j 2 results
org.springframework.boot.autoconfigure.pulsar 2 results
org.springframework.boot.autoconfigure.r2dbc 2 results
Maven: org.springframework.boot:spring-boot-docker-compose:3.2.0-RC1 24 results

TESTCONTAINERS

TESTCONTAINERS:

Test dependencies as code

No more need for mocks or complicated environment configurations. Define your test dependencies as code, then simply run your tests and containers will be created and then deleted.

With support for many languages and testing frameworks, all you need is Docker.

```java
@SpringBootTest
class UserRepositoryTest {

    @Container
    static PostgreSQLContainer<?> postgres = new PostgreSQLContainer<>("postgres:13")
            .withDatabaseName("testdb")
            .withUsername("test")
            .withPassword("test");

    @DynamicPropertySource
    static void configureProperties(DynamicPropertyRegistry registry) {
        registry.add("spring.datasource.url", postgres::getJdbcUrl);
        registry.add("spring.datasource.username", postgres::getUsername);
        registry.add("spring.datasource.password", postgres::getPassword);
    }

    @Test
    void shouldSaveUser() {
        // test code
    }
}
```

```java
@SpringBootTest
@Testcontainers
class UserRepositoryTest {

    @Container
    @ServiceConnection
    static PostgreSQLContainer<?> postgres = new PostgreSQLContainer<>("postgres:13");

    @Test
    void shouldSaveUser() {
        // test code - that's it!
    }
}
```

Remember that Docker Compose

Feature we talked about earlier?

That is really nice but...

```java
@TestConfiguration(proxyBeanMethods = false)
public class TestcontainersConfiguration {

    @Bean
    @ServiceConnection
    PostgreSQLContainer<?> postgresContainer() {
        return new PostgreSQLContainer<>("postgres:15-alpine");
    }

    @Bean
    @ServiceConnection
    RedisContainer redisContainer() {
        return new RedisContainer("redis:7-alpine");
    }
}
```

```java
@SpringBootApplication
public class MyApplication {

    public static void main(String[] args) {
        SpringApplication.from(MyApplication::main)
            .with(TestcontainersConfiguration.class)
            .run(args);
    }
}
```

DEMO TIME

CURRENT: SPRING FRAMEWORK 6.X & SPRING BOOT 3.X

Spring Boot 3.2 (November 2023)

- JDK 21 (LTS) Support
- Virtual Threads
- New Rest Client
- New JDBC Client
- SSL Bundle Reloading

REST CLIENT

HISTORY OF CLIENT ABSTRACTIONS

- Rest Template
- Web Client
- Graphql Client
- Declarative Http Client
  - Spring Cloud Open Feign
  - Http Interfaces
- Rest Client

REST CLIENT

Perform HTTP Requests & Return Responses

Exposes a Fluent & Synchronous API

Builds on everything we learned from RestTemplate & WebClient

Http Interfaces using the Rest Client in an imperative application

Underlying HTTP Client Libraries

- JDK HttpClient (default)
- Apache HttpClient

```java
private final RestClient restClient;

public PostService() {
    restClient = RestClient.builder()
            .baseUrl("https://jsonplaceholder.typicode.com")
            .defaultHeader( header: "X-POWERED-BY", ...values: "Dan's YouTube Channel")
            .build();
}

List<Post> findAll() {
    return restClient.
}
```

```
get()                                   RequestHeadersUriSpec<?>
put()                                   RequestBodyUriSpec
post()                                  RequestBodyUriSpec
delete()                                RequestHeadersUriSpec<?>
var                                     T name = expr
head()                                  RequestHeadersUriSpec<?>
method(HttpMethod method)               RequestBodyUriSpec
mutate()                                Builder
options()                               RequestHeadersUriSpec<?>
patch()                                 RequestBodyUriSpec
equals(Object obj)                      boolean
hashCode()                              int
Press ^. to choose the selected (or first) suggestion and insert a dot afterwards  Next Tip
```

```
List<Post> findAll() {
    return restClient.get() RequestHeadersUriSpec<capture of ?>
            .uri( uri: "/posts") capture of ?
            .header( headerName: "X-Powered-By", ...headerValues: "Dan's YouTube Channel")
            .retrieve() ResponseSpec
            .body(new ParameterizedTypeReference<>() {});
}

Post findById(int id) {
    return restClient.get() RequestHeadersUriSpec<capture of ?>
            .uri( uri: "/posts/{id}", id) capture of ?
            .retrieve() ResponseSpec
            .body(Post.class);
}
```

```
Post create(Post post) {
    return restClient.post() RequestBodyUriSpec
            .uri( uri: "/posts") RequestBodySpec
            .contentType(MediaType.APPLICATION_JSON)
            .body(post)
            .retrieve() ResponseSpec
            .body(Post.class);
}

void delete(Integer id) {
    restClient.delete() RequestHeadersUriSpec<capture of ?>
            .uri( uri: "/posts/{id}", id) capture of ?
            .retrieve() ResponseSpec
            .toBodilessEntity();
}
```

JDBC CLIENT

HISTORY OF DATABASE ABSTRACTIONS

- **J**ava **D**ata**B**ase **C**onnection (JDBC)
- JDBC Template
- Spring Data
  - Spring Data JPA
  - Spring Data JDBC
- JDBC Client

```java
private static final Logger log = LoggerFactory.getLogger(JdbcTemplatePostService.class);
private final JdbcTemplate jdbcTemplate;

public JdbcTemplatePostService(JdbcTemplate jdbcTemplate) { this.jdbcTemplate = jdbcTemplate; }

RowMapper<Post> rowMapper = (rs, rowNum) -> new Post(
        rs.getString( columnLabel: "id"),
        rs.getString( columnLabel: "title"),
        rs.getString( columnLabel: "slug"),
        rs.getDate( columnLabel: "date").toLocalDate(),
        rs.getInt( columnLabel: "time_to_read"),
        rs.getString( columnLabel: "tags")
);

@Override
public List<Post> findAll() {
    var sql = "SELECT id,title,slug,date,time_to_read,tags FROM post";
    return jdbcTemplate.query(sql, rowMapper);
}
```

```java
@Override
public Optional<Post> findById(String id) {
    var sql = "SELECT id,title,slug,date,time_to_read,tags FROM post WHERE id = ?";
    Post post = null;
    try {
        post = jdbcTemplate.queryForObject(sql,rowMapper,id);
    } catch (DataAccessException ex) {
        log.info("Post not found: " + id);
    }

    return Optional.ofNullable(post);
}

@Override
public void create(Post post) {
    String sql = "INSERT INTO post(id,title,slug,date,time_to_read,tags) values(?,?,?,?,?,?)";
    int insert = jdbcTemplate.update(sql,post.id(),post.title(),post.slug(),post.date(),post.timeToRead(),post.tags());
    if(insert == 1) {
        log.info("New Post Created: " + post.title());
    }
}
```

```
List<Post> findAll() {
    return jdbcClient.sql("SELECT * FROM post") StatementSpec
            .query(Post.class) MappedQuerySpec<Post>
            .list();
}

public Optional<Post> findById(String id) {
    return jdbcClient.sql("SELECT id,user_id,title,body FROM post WHERE id = :id") StatementSpec
            .param( name: "id", id)
            .query(Post.class) MappedQuerySpec<Post>
            .optional();
}

public void create(Post post) {
    int update = jdbcClient.sql("INSERT INTO post (id, user_id, title, body) VALUES (?, ?, ?, ?)")
            .params(List.of(post.id(), post.userId(), post.title(), post.body()))
            .update();
    log.info("Inserted {} rows", update);
}
```

VIRTUAL THREADS

WHAT IS A THREAD IN JAVA

- Java is Multithreaded by default
- *`java.lang.Thread`*
  - Abstraction over operating system threads
  - Every thread is essentially a wrapper over an OS thread
- OS/JVM thread = platform thread
- Platform threads are expensive
- Take a lot of memory / take time to create & destroy
- Traditional threads are stored on the JVM stack

Project Loom is to intended to explore, incubate and deliver Java VM features and APIs built on top of them for the purpose of supporting **easy-to-use**, **high-throughput lightweight concurrency** and new programming models on the Java platform.

PROJECT LOOM

Project Loom includes the following JEPs.

- JEP 425: Virtual Threads went final in JDK 21
- JEP 428: Structured Concurrency (Preview) preview continued in JDK 25.
- JEP 429: Scoped Values (Preview) Finalized in JDK 25.

JDK 24: Virtual Threads without Pinning (JEP 491)

Problem (JDK 23)

In JDK 21, when a virtual thread entered a synchronized block and performed blocking I/O, it became pinned to its carrier thread—tying up platform threads and hurting scalability.Optional for optimized JVM deployment

Solution (JDK 24):

Virtual threads now release their carrier even within synchronized blocks, allowing the carrier to serve other threads while blocked which dramatically boosts throughput.

Performance Gains:

- Pure Java benchmark (5,000 threads each with unique locks and sleep in sync block):
  JDK 21 → 31.8 s, JDK 24 → 0.45 s (≈70× faster).

- Spring Boot high-concurrency simulation:
  ≈800 requests/sec → ≈4,264 requests/sec (~5.3× increase).

spring

VIRTUAL THREADS & SPRING APPLICATIONS

EMBRACING VIRTUAL THREADS IN SPRING APPLICATIONS

Web Applications.

- Servlet callback based on Input / Output Stream
- Tomcat / Jetty Executor setup for dispatching to request handlers
- “Loom-ready” database drivers, scalable connection pool setup
- Ideally no code changes in the main application codebase
- Virtual threads supported in GraalVM Native Image as well

- Significant scalability benefits for database driven web applications

spring

EMBRACING VIRTUAL THREADS IN SPRING APPLICATIONS

Messaging / Scheduling

- Spring managed task executors with virtual thread options
  - JMS Message Listener containers
  - @Scheduled handler methods
- Many of those listeners and handlers methods trigger I/O operations

EMBRACING VIRTUAL THREADS IN SPRING APPLICATIONS

Where it doesn't make sense

- Purely CPU bound handler methods
- Reactive based web applications

ENABLING VIRTUAL THREADS IN SPRING

- spring.threads.virtual.enabled=true
- This will cause Spring Boot's Auto-configuration to request virtual threads in all the libraries that support it
  - Spring Framework's MVC
  - Embedded Web Server
  - Task Executor
  - Client Libraries (Rabbit, Kafka, Pulsar, Redis, etc...)

DEMO TIME

CURRENT: SPRING FRAMEWORK 6.X & SPRING BOOT 3.X

Spring Boot 3.3 (May 2024)

- CDS Support
- Observability Enhancements
- SBOM Actuator Endpoint
- Service Connections
- Base64 Resources

DEMO TIME

CURRENT: SPRING FRAMEWORK 6.X & SPRING BOOT 3.X

Spring Boot 3.4 (November 2024)

- Structured Logging
- @Fallback Beans & Enhanced Bean Management
- AssertJ Support for MockMvc
- Expanded Virtual Thread Support
- Building Images: Out-of-the-box support for ARM

DEMO TIME

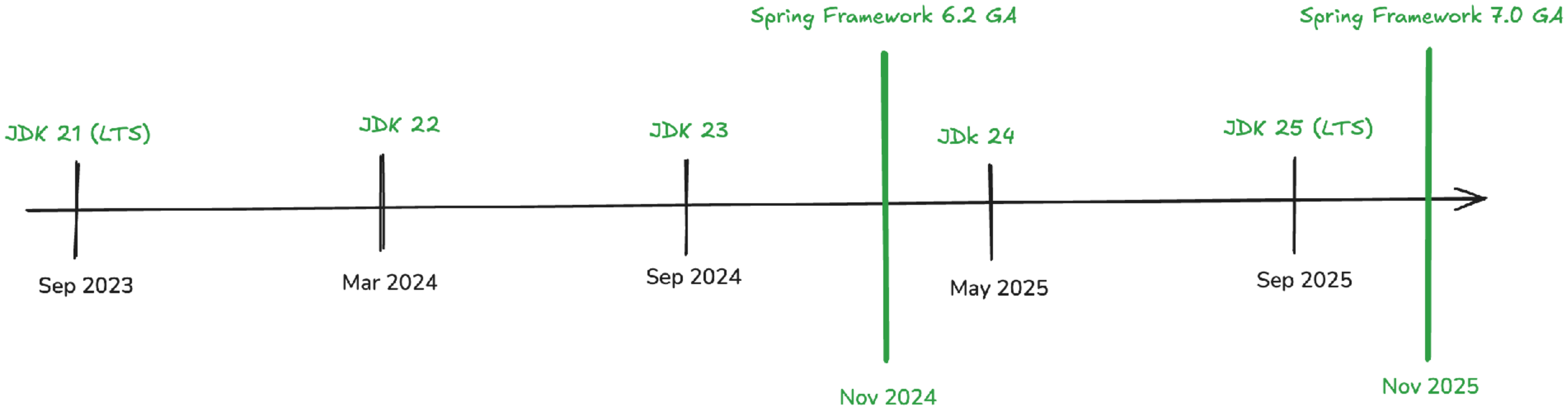
CURRENT: SPRING FRAMEWORK 6.X & SPRING BOOT 3.X

Spring Boot 3.5 (May 2025)

- SSL Bundle Metrics
- Load Properties from Environment Variables
  - `APP_CONFIG="app.message=Hello from ENV properties!"`
  - `SPRING_CONFIG_IMPORT=optional:env:APP_CONFIG`
- Trigger Quartz Jobs from the Actuator
- JSON output for ECS structured logging updated to nested format

DEMO TIME

SPRING FRAMEWORK 7 & SPRING BOOT 4


Spring Framework 6.2 GA
Spring Framework 7.0 GA
JDK 21 (LTS)
JDK 22
JDK 23
JDk 24
JDK 25 (LTS)
Sep 2023
Mar 2024
Sep 2024
Nov 2024
May 2025
Sep 2025
Nov 2025

Project

- ○ Gradle - Groovy
- ○ Gradle - Kotlin
- ● Maven

Language

- ● Java
- ○ Kotlin
- ○ Groovy

Spring Boot

- ○ 4.0.0 (SNAPSHOT)
- ● 4.0.0 (M1)
- ○ 3.5.5 (SNAPSHOT)
- ○ 3.5.4
- ○ 3.4.9 (SNAPSHOT)
- ○ 3.4.8

Project Metadata

Group	dev.danvega
Artifact	boot4
Name	boot4
Description	A First Look at Spring Boot 4
Package name	dev.danvega.boot4
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.

GENERATE ⌘ + ↵ EXPLORE CTRL + SPACE ...

INTRODUCING SPRING FRAMEWORK 7.X

A new generation of Spring for 2026 and beyond 🚀

- JDK 17+ (JDK 25 LTS Support)
- Jakarta EE 11
- Standard JSpecify Nullability
- Kotlin 2.x
- AOT on GraalVM and Leyden
- Virtual Threads on JDK 25

spring

JSPECIFY

Standard nullability annotations

- Type-use annotations even for nested generic type declarations
- @NullMarked packages with @Nullable parameters / fields / etc.
- Tooling ecosystem: eg. NullAway

spring

KOTLIN 2.X

Embracing the latests generation of Kotlin

- Deeper integration with Java nullability through JSpecify
- New K2 compiler for all platforms
- Faster and lighter reflection

BEAN REGISTRARS

Encapsulated programmatic bean registration with AOT support

```java
// 1. Create a BeanRegistrar implementation
public class EmailServiceRegistrar implements BeanRegistrar {
    @Override
    public void register(BeanRegistry registry, Environment env) {
        // Register a bean with constructor arguments
        registry.registerBean("emailService", EmailService.class,
            bean -> {
                bean.addConstructorArgValue(env.getProperty("email.host"));
                bean.addConstructorArgValue(env.getProperty("email.port"));
            });
    }
}

// 2. Import it in your configuration
@Configuration
@Import(EmailServiceRegistrar.class)
public class AppConfig {
    // Your other @Bean definitions
}

// 3. That's it! The EmailService bean is now available for injection
```

spring

```java
// 1. Create a BeanRegistrar implementation
public class EmailServiceRegistrar implements BeanRegistrar {
    @Override
    public void register(BeanRegistry registry, Environment env) {
        // Register a bean with constructor arguments
        registry.registerBean("emailService", EmailService.class,
            bean -> {
                bean.addConstructorArgValue(env.getProperty("email.host"));
                bean.addConstructorArgValue(env.getProperty("email.port"));
            });
    }
}

// 2. Import it in your configuration
@Configuration
@Import(EmailServiceRegistrar.class)
public class AppConfig {
    // Your other @Bean definitions
}

// 3. That's it! The EmailService bean is now available for injection
```

API VERSIONING

First-class support for web endpoint versioning in MVC/WebFlux

- Version source: request header, request param, path variable
- Configurable API versioning strategy for the entire application

```
@RequestMapping(path = "/myEndpoint", version = "2.0")
public ResponseEntity<...> myEndpointV2() {
    ...
}
```

spring

UNIFIED JMS CLIENT

A modern alternative to JmsTemplate / JmsMessagingTemplate

- For *`jakarta.jms.Message`* and *`org.springframework.messaging.Message`*
- Customizable operations in fluent style: QoS settings etc.
- Unified exception translation to MessageException
- Unified message conversation capabilities

- Along the lines of the JdbcClient & RestClient

HTTP INTERFACE CLIENT CONFIGURATION

```java
import java.util.List;

@HttpExchange(url = "https://jsonplaceholder.typicode.com")
public interface TodoClient {

    @GetExchange("/todos")
    List<Todo> getAllTodos();

    @GetExchange("/todos/{id}")
    Todo getTodoById(@PathVariable Integer id);

    @PostExchange("/todos")
    Todo createTodo(@RequestBody Todo todo);

    @PutExchange("/todos/{id}")
    Todo updateTodo(@PathVariable Integer id, @RequestBody Todo todo);

    @DeleteExchange("/todos/{id}")
    void deleteTodo(@PathVariable Integer id);
}
```

```java
@Configuration
@ImportHttpServices(TodoClient.class)
public class HttpClientConfig {

}
```

spring

```java
@Configuration(proxyBeanMethods = false)
@ImportHttpServices(group = "weather", types = {FreeWeather.class, CommercialWeather.class})
@ImportHttpServices(group = "user", types = {UserServiceInternal.class, UserServiceOfficial.class})
static class HttpServicesConfiguration extends AbstractHttpServiceRegistrar {

    @Bean
    public RestClientHttpServiceGroupConfigurer groupConfigurer() {
        return groups -> groups.filterByName("weather", "user")
                .configureClient((group, builder) -> builder.defaultHeader("User-Agent", "My-Application"));
    }

}
```

SPRING FRAMEWORK 7.0 RELEASE NOTES

https://github.com/spring-projects/spring-framework/wiki/Spring-Framework-7.0-Release-Notes

SPRING BOOT 4.0 RELEASE NOTES

https://github.com/spring-projects/spring-boot/wiki/Spring-Boot-4.0-Release-Notes

ROADMAP

SPRING FRAMEWORK 6.2 GA : AVAILABLE NOW!

Final 6.x feature branch with long-term support

- Foundation for Spring Boot 3.4 and 3.5
- Production-ready on JDK 17 LTS and JDK 21 LTS
- Regular stream of 6.2 maintenance releases

Includes deep core container revision

- Faster autowiring, fallback beans, background initialization
- Revised generic type matching algorithm for partial generics

spring

SPRING FRAMEWORK 7.0 GA : NOVEMBER 2025

A new generation for 2026 and beyond

- Foundation for **Spring Boot 4.0**
- Compatible with **JDK 17+**, optimized for **JDK 25 LTS**
- Jakarta EE 11, JSpecify, Kotlin 2.x, Leyden and more

Empowering modern application architectures

- Programmatic registration of beans (including AOT)
- API versioning for web endpoints, unified JMS Client

spring

THANK YOU!

danvega@gmail.com

https://www.danvega.dev