# How to Implement Server-Sent Events (SSE) in Spring Boot

8 min read · Apr 20, 2024

Alexander Obregon Follow

Listen Share

Image Source

## Introduction

When it comes to real-time web applications, keeping users informed with live updates is more important than ever. From financial dashboards displaying live stock market data to social media platforms updating with new posts, real-time data delivery is a fundamental requirement. Server-Sent Events (SSE) provide a streamlined way to push updates from a server to connected clients. In this article, we'll explore how to implement SSE in Spring Boot, using practical examples such as a live news feed or a real-time data dashboard.

## Understanding Server-Sent Events (SSE)

Server-Sent Events (SSE) offer a strong mechanism for enhancing web applications with real-time communication capabilities. These events are used to push data from the server to the client over a single, long-lived HTTP connection. This method provides a standardized way to send text data to the client which can be easily consumed by a web browser or any SSE-compatible client.

### What are Server-Sent Events?

SSE is a standard describing how servers can initiate data transmission towards browser clients once an initial client connection has been established. This protocol is particularly well-suited for scenarios where a server needs to send real-time updates to the client, such as a live news feed, a sports game result updates, or a financial dashboard that requires frequent data refresh without the overhead of re-establishing connections. Some of the benefits of SSE:

- **Simplicity and Efficiency:** SSE is simpler than WebSockets because it does not require the client and server to handle and maintain a two-way communication channel. SSE is designed for one-way communication from server to client, which simplifies the architecture and reduces both coding overhead and server load.
- **Built on HTTP:** Unlike WebSockets, which require special protocols and handshake mechanisms, SSE works entirely over HTTP. This means that it seamlessly integrates with existing web infrastructure such as HTTP/2, proxies, and firewalls.
- **Automatic Reconnection:** One of the most useful features of SSE is its built-in ability to automatically reconnect if the connection gets dropped. This makes sure that the client always attempts to maintain an active connection to the server, reducing the complexity of connection management on the client-side.
- **Event Identification:** SSE allows defining custom event types which can help in handling different types of messages differently on the client side. This adds a layer of flexibility, allowing developers to architect their applications with clear separation and handling for various message types.

### How Does SSE Work?

When implementing SSE, the server sets up a standard HTTP response, changing the “Content-Type” to “text/event-stream”. This tells the client that it’s establishing a stream where incoming data will be formatted as events. These events are plain text and are separated by a double newline. Each event sent by the server can have an optional event name, an optional ID, and data:

```
event: messageEvent
id: 12345
data: Here is some text data
```

The client, usually a web browser, listens to this stream through a JavaScript interface — typically an `EventSource` object. The `EventSource` API is designed to handle incoming messages by dispatching events to the browser.

### Practical Example

Consider a real-time traffic monitoring application. The server can push updates about traffic conditions, like congestion and accidents, as they occur. The client’s `EventSource` object will listen to these events and can update the UI in real-time, providing end-users with immediate updates without requiring the client to constantly fetch new data.

### Limitations of SSE

- **Browser Support:** While most modern browsers support SSE, Internet Explorer and older versions of other browsers do not. This may require fallback solutions for compatibility with all user bases.

- **One-way Communication:** SSE only supports one-way communication from the server to the client. If bi-directional communication is needed, then technologies like WebSockets might be more appropriate.
- **Handling Binary Data:** SSE is optimized for text data. While it's possible to send binary data through SSE, it requires encoding the binary data into a string format such as base64, which can increase the data size and processing overhead.

By integrating SSE into web applications, developers can significantly enhance the user experience by providing live, real-time data updates efficiently and with minimal infrastructure changes. This technology is especially useful for applications where users benefit from immediate notifications and updates, such as news feeds, live sports scores, or real-time monitoring systems.

## Setting Up Spring Boot for SSE

Implementing Server-Sent Events in a Spring Boot application involves several straightforward steps. Here you will be guided through the initial setup, including project creation, adding dependencies, and configuring a basic SSE controller.

### Prerequisites

Before starting, you'll need the Java Development Kit (JDK) version 11 or newer installed on your machine. You will also require a build tool such as Maven or Gradle, and an Integrated Development Environment (IDE) like IntelliJ IDEA, Eclipse, or Visual Studio Code.

### Create a New Spring Boot Application

The easiest way to start a new Spring Boot project is via the Spring Initializr website:

1. Go to Spring Initializr.
2. Choose Java as the language and Maven or Gradle as the build tool.
3. Select Spring Boot version 2.x or later.
4. Add dependencies: 'Spring Web' for web application support and 'Spring Boot DevTools' for automatic restarts and live reloads during development.

Once the project is generated, download and unzip it, then open it in your chosen IDE.

### Add Dependencies for SSE

To enable SSE in your Spring Boot application, you only need the Spring Web dependency, which includes all necessary modules to handle HTTP connections and data streaming. Make sure this dependency is included in your project's build configuration file ( `pom.xml` for Maven or `build.gradle` for Gradle).

### Implement an SSE Controller in Spring Boot

To set up SSE, you need to create a controller that will manage the SSE connections and data transmission. Here's how to implement a basic SSE controller:

1. **Define a Controller Class:** Create a new Java class in your project and annotate it with `@RestController` . This specifies that the class will be used to handle web requests.
2. **Create an SSE Endpoint:** Define a method in the controller that returns an `SseEmitter` , which is part of the Spring framework and is used to handle SSE connections.
3. **Implement Data Sending Logic:** Inside this method, configure how data will be sent to the clients. You can use a service or a component to generate or retrieve data that needs to be pushed to clients.

Here is a simple example of an SSE controller:

```
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.servlet.mvc.method.annotation.SseEmitter;
import org.springframework.stereotype.Controller;

import java.io.IOException;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

@Controller
public class SSEController {

    private final ExecutorService executor = Executors.newCachedThreadPool();

    @GetMapping("/stream-sse")
    public SseEmitter streamSseEvents() {
        SseEmitter emitter = new SseEmitter(Long.MAX_VALUE); // Long-lived connection
        executor.execute(() -> {
            try {
                for (int i = 0; i < 10; i++) {
                    Thread.sleep(1000); // simulate delay
                    emitter.send("SSE MVC - " + System.currentTimeMillis());
                }
                emitter.complete();
            } catch (IOException | InterruptedException e) {
                emitter.completeWithError(e);
            }
        });
        return emitter;
    }
}
```

**Testing the SSE Controller**

To verify that your SSE setup works, you can use a browser or tools like curl to connect to the SSE endpoint (`http://localhost:8080/stream-sse`). The server should stream events back to the client, displaying timestamps at one-second intervals.

By following these steps, you have successfully set up a basic SSE architecture in your Spring Boot application, allowing the server to push real-time events to connected clients. This setup can be extended with more complex logic for fetching and streaming data according to your application's requirements.

## Building a Real-Time News Feed with SSE

Now that we have our Spring Boot setup ready to handle Server-Sent Events, let's implement a real-time news feed feature. This feature will allow us to stream news updates to clients in real-time, using SSE. We'll walk through creating the model for the news items, setting up a service to manage news events, and extending the SSE controller to handle these news updates.

### Define a News Item Model

First, we need to define a simple data model that represents a news item. This model will be used to send news data to clients.

```java
import java.time.LocalDateTime;

public class NewsItem {
    private String title;
    private String content;
    private LocalDateTime dateTime;

    public NewsItem(String title, String content, LocalDateTime dateTime) {
        this.title = title;
        this.content = content;
        this.dateTime = dateTime;
    }

    public String getTitle() {
        return title;
    }

    public String getContent() {
        return content;
    }

    public LocalDateTime getDateTime() {
        return dateTime;
    }

    @Override
    public String toString() {
        return "NewsItem{" +
                "title='" + title + '\'' +
                ", content='" + content + '\'' +
                ", dateTime=" + dateTime +
                '}';
    }
}
```

### Create a News Service

Next, we create a service that will handle the generation and distribution of news items. This service could in a real-world application be connected to a database or an external API. For simplicity, we will simulate news generation.

```java
import org.springframework.stereotype.Service;

import java.time.LocalDateTime;
import java.util.ArrayList;
import java.util.List;

@Service
public class NewsService {
    private final List<NewsItem> newsItems = new ArrayList<>();

    // Method to simulate news updates
    public void addNewsItem(String title, String content) {
        NewsItem newsItem = new NewsItem(title, content, LocalDateTime.now());
        newsItems.add(newsItem);
        // Notify subscribers
        notifySubscribers(newsItem);
    }

    // Placeholder method to simulate subscriber notifications
    private void notifySubscribers(NewsItem newsItem) {
        // This should interact with the SSE controller to send updates
    }

    public List<NewsItem> getNewsItems() {
        return newsItems;
    }
}
```

### Extend the SSE Controller to Handle News Updates

We will now modify the SSE controller to handle connections from clients and push updates when new news items are available.

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.servlet.mvc.method.annotation.SseEmitter;

import java.util.Collections;
import java.util.List;
import java.util.concurrent.CopyOnWriteArrayList;

@RestController
public class NewsController {

    private final List<SseEmitter> emitters = new CopyOnWriteArrayList<>();
    private final NewsService newsService;

    @Autowired
    public NewsController(NewsService newsService) {
        this.newsService = newsService;
    }

    @GetMapping("/news")
    public SseEmitter subscribeToNews() {
        SseEmitter emitter = new SseEmitter(Long.MAX_VALUE);
        emitters.add(emitter);

        emitter.onCompletion(() -> emitters.remove(emitter));
        emitter.onTimeout(() -> emitters.remove(emitter));
        emitter.onError((e) -> emitters.remove(emitter));

        // Send existing news on subscription
        newsService.getNewsItems().forEach(newsItem -> {
            try {
                emitter.send(SseEmitter.event().name("NEWS").data(newsItem.toString()));
            } catch (Exception e) {
                emitter.completeWithError(e);
            }
        });

        return emitter;
    }
    public void dispatchNewsItem(NewsItem newsItem) {
        List<SseEmitter> deadEmitters = new ArrayList<>();
        emitters.forEach(emitter -> {
            try {
                emitter.send(SseEmitter.event().name("NEWS").data(newsItem.toString()));
            } catch (Exception e) {
                deadEmitters.add(emitter);
            }
        });
        emitters.removeAll(deadEmitters);
    }
}
```

### Testing the News Feed

To test the real-time news feed:

1. Run the Spring Boot application.
2. Connect to the SSE endpoint ( `http://localhost:8080/news` ) using a web browser or a tool like curl.
3. Use a method in your `NewsService` to generate new news items.

As news items are added, they should be pushed to all connected clients in real-time. This setup effectively demonstrates how SSE can be utilized to build a live news feed in a Spring Boot application. With this in place, you can scale or modify the service to cater to more complex scenarios or integrate with real-world data sources.

## Conclusion

Implementing Server-Sent Events (SSE) in Spring Boot provides a powerful solution for enhancing web applications with real-time capabilities. This technology enables efficient and effective communication from the server to the client, ideal for applications that require live data updates such as news feeds, financial dashboards, and social media platforms. By following the steps outlined in this article, you can seamlessly integrate SSE into your Spring Boot applications, ensuring that your users receive timely and continuous updates without the need for manual refreshes.

With its simplicity and integration with existing web standards, SSE represents an accessible yet strong choice for developers looking to deliver dynamic content and interactive user experiences. Whether you're building a complex real-time analytics platform or a simple live notification system, SSE in Spring Boot offers a straightforward path to achieving your real-time data transmission goals.

1. *Spring Initializr*
2. *Spring Framework Documentation*
3. *Oracle JDK Download*

**Thank you for reading! If you find this helpful, please consider highlighting, clapping, responding or connecting with me on Twitter/X as it's very appreciated and helps keep content like this free!**

Spring Boot icon by Icons8

Spring Boot   Java   Programming   Technology   Server Sent Events

Follow

## Written by Alexander Obregon

27K followers · 14 following

I post daily about programming topics and share what I learn as I go. For recaps, exclusive content, and to support me: https://alexanderobregon.substack.com

## Responses (2)

Write a response

What are your thoughts?

Highlight   Share

BabyIT
Apr 22, 2024

Thanks for your article!

21   Reply

Sam SamuraiDev
Jul 24, 2024

hi, what about integration testing of these endpoints?

Reply

## More from Alexander Obregon

Alexander Obregon

### How Spring Boot Auto-Configuration Works

Spring Boot's auto-configuration feature is one of its standout functionalities, allowing developers to build applications with minimal...

Nov 18, 2024 188 4

Alexander Obregon

**Enhancing Logging with @Log and @Slf4j in Spring Boot Applications**

Spring Boot has become a popular choice for developing enterprise-grade applications due to its ease of use, powerful features, and strong...

Sep 22, 2023 367 5

Alexander Obregon

**Using Spring Boot with Flyway to Manage Database Migrations**

Managing changes to a database schema over time is one of the hardest parts of working on any backend system. Flyway solves this by...

Jul 1 20 1

Alexander Obregon

**Building Real-time Applications with Python and WebSockets**

Real-time applications have become much more common over the past decade, enabling instant communication and live updates that keep users…

Jun 2, 2024 72 4

See all from Alexander Obregon

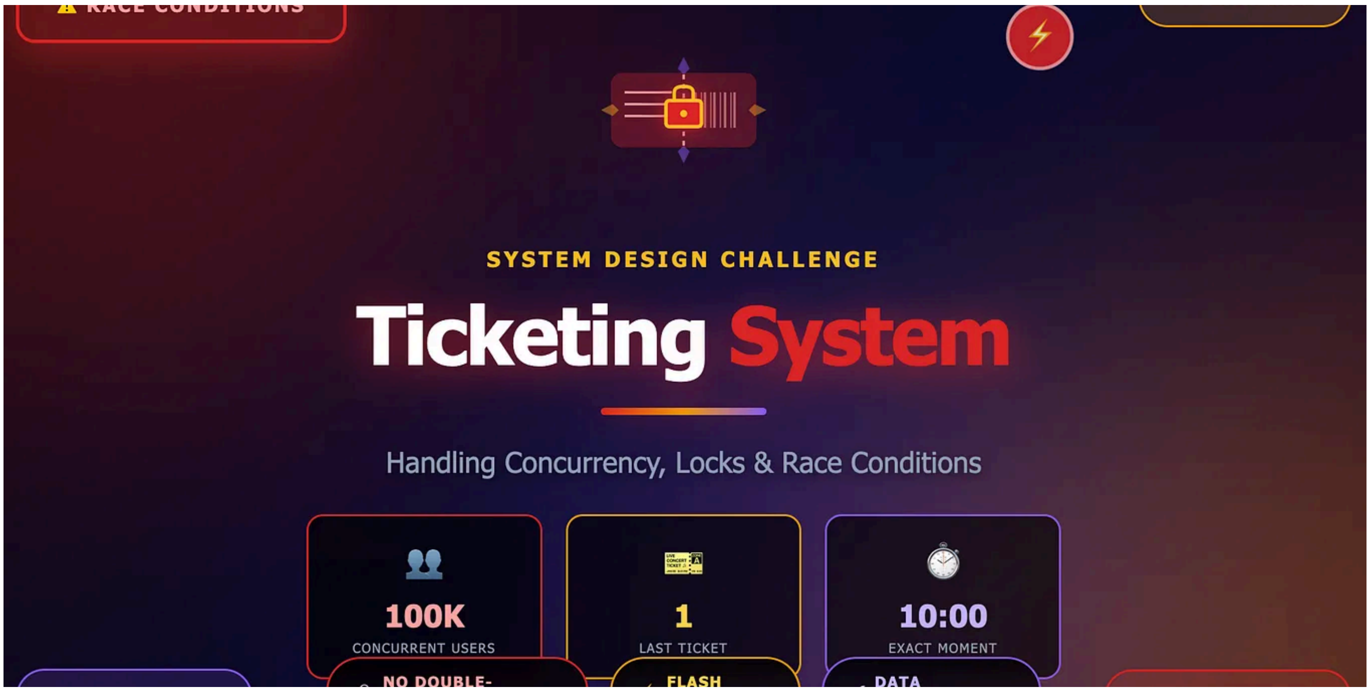
## Recommended from Medium

Nitesh Thakur

## Polling vs Long Polling vs SSE vs WebSockets — The Deep Dive Behind Real-Time Cricket Scores

When India plays Pakistan, millions of cricket fans open apps like Cricbuzz or ESPNcricinfo to follow every ball. Updates need to be...

Sep 23



Arvind Kumar

**Building a Ticketing System: Concurrency, Locks, and Race Conditions**

What happens when 100,000 fans try to book the same concert ticket at exactly 10:00 AM? Let's design a ticketing system that prevents...

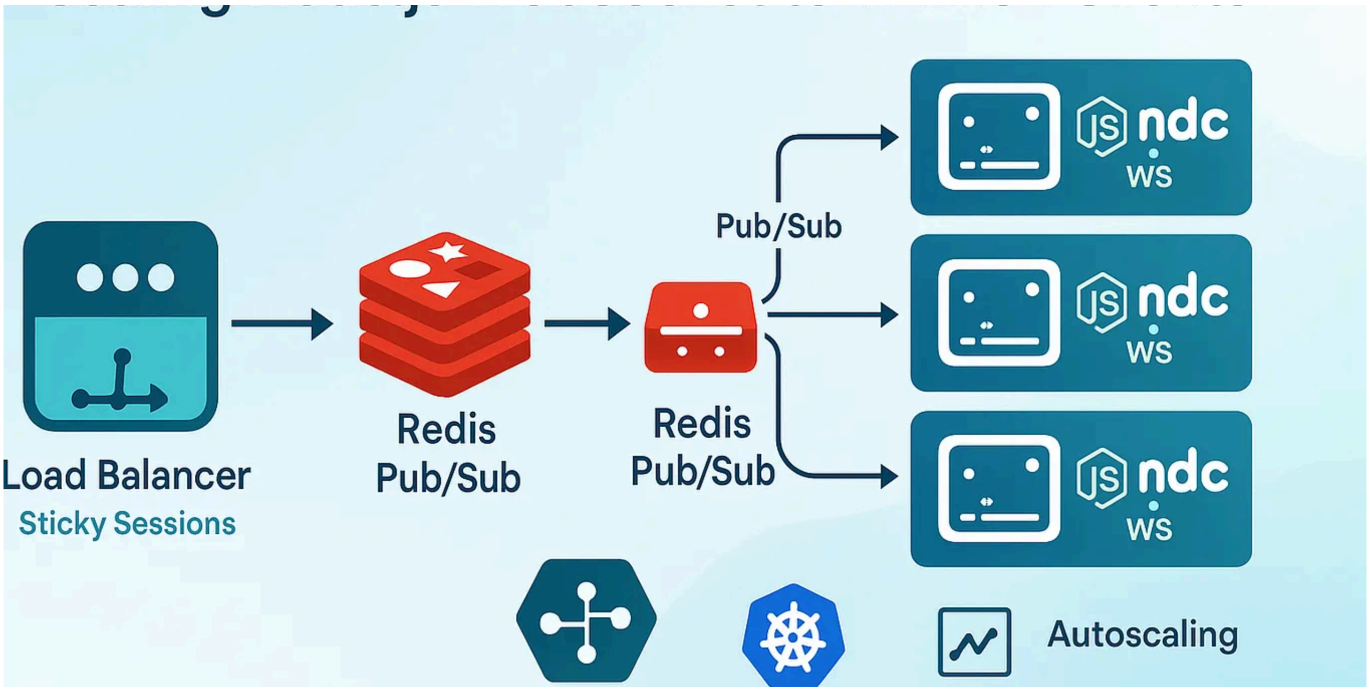
Oct 30 1.2K 11

Renan Schmitt

**How to Prevent Concurrent Job Execution in Spring Boot with Redis**

Have you ever deployed multiple instances of your Spring Boot service—only to realize that all of them are running the same scheduled job...

Dec 1 3

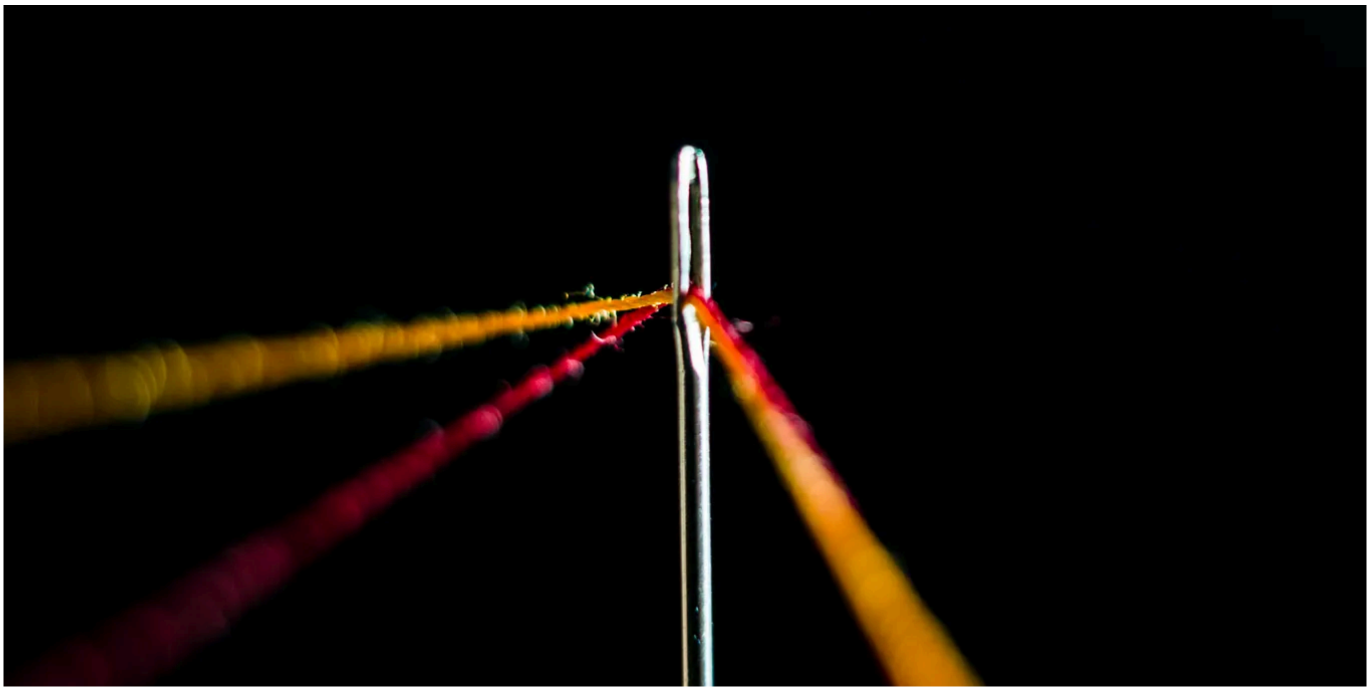


Hash Block

**Scaling Node.js to 1 Million Concurrent WebSocket Clients with Horizontal Sharding**

How I Built a High-Throughput Real-Time Infrastructure Using Native ws, Redis Pub/Sub, and Smart Sharding

Jul 30 9



In Javarevisited by Ashish Choudhary

**How Java Virtual Threads Broke Netflix**

TL;DR: Thread pinning

Oct 1 459 2

JS In JavaScript in Plain English by Riya Sharma

**TLS/SSL Setup For Spring Boot Made Simple**

A Step-by-Step Guide

Sep 22 7 1

See more recommendations