Eventos enviados pelo servidor usando o Spring WebFlux e o Kafka reativo

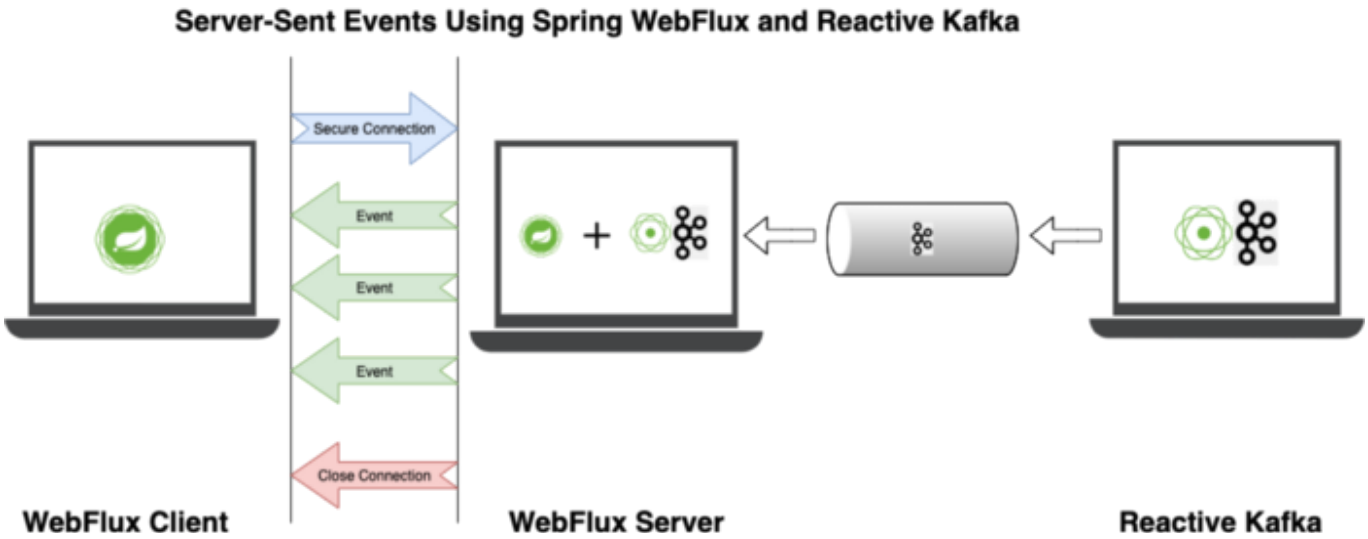
Gagan SolurVenkatesh Segue
27 de abril · 6 min de leitura

Hoje, vamos dar uma olhada no fluxo de dados de um servidor WebFlux REST reativo baseado em Kafka para um cliente Webflux de uma maneira sem bloqueio.

A arquitetura projetada abaixo pode ser usada para:

- Envie dados para aplicativos externos ou internos quase em tempo real.
- Envie dados para os arquivos e copie-os com segurança para os serviços em nuvem.
- Envie os mesmos dados para vários clientes a partir de um tópico Kafka.

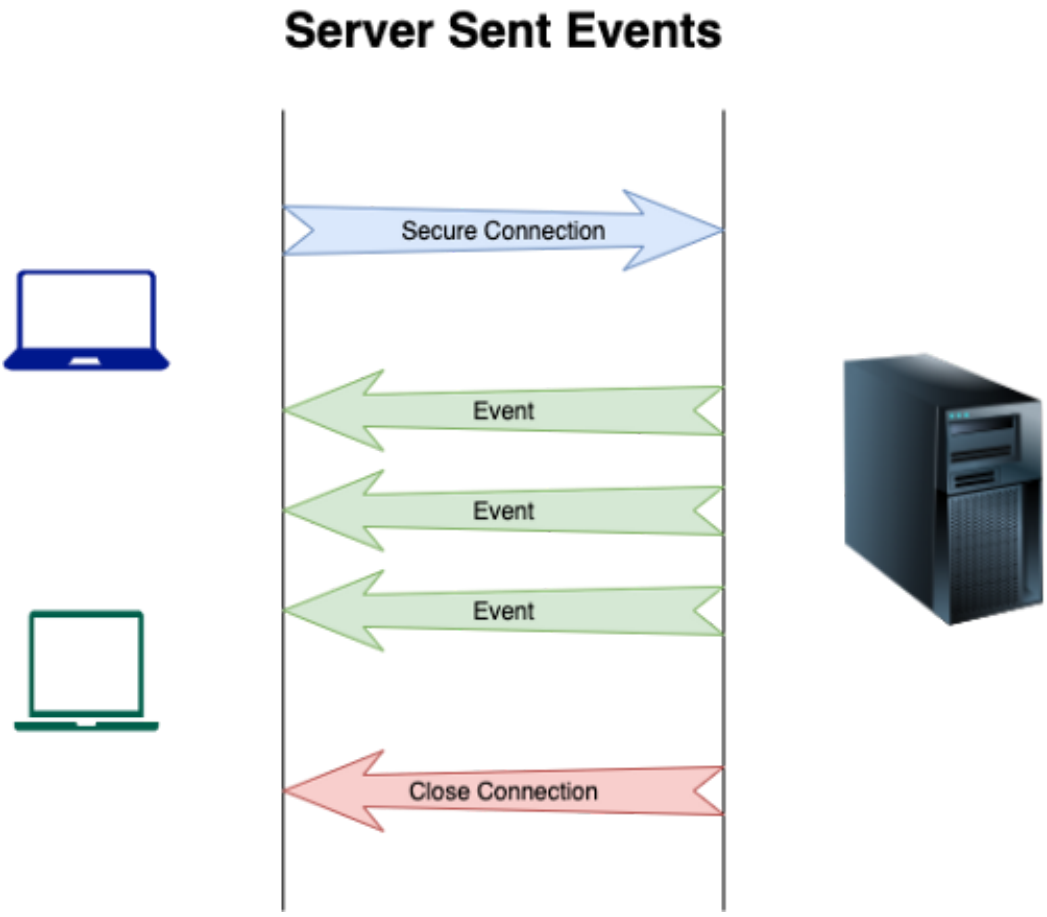
Vamos começar!



1. Eventos Enviados pelo Servidor Usando Spring WebFlux e Kafka Reativo para Cliente Único

Antes de executarmos um aplicativo de amostra para demonstrar eventos enviados pelo servidor (SSE) usando o Spring WebFlux e o Kafka reativo, vamos entender os conceitos fundamentais:

O que são eventos enviados pelo servidor?
Eventos enviados pelo servidor (SSE) é uma tecnologia Server Push que permite que um cliente receba atualizações automáticas do servidor por meio da conexão HTTP.



2. Eventos Enviados pelo Servidor

O SSE pode ser usado para:

- Substitua a pesquisa longa (que cria uma nova conexão para cada pull), mantendo uma conexão única e mantendo um fluxo contínuo de eventos passando por ela.

- Ative aplicativos que usam comunicação de dados unidirecional (por exemplo: sites de comércio eletrônico, atualizações de preços de ações ao vivo).

O que é o Spring WebFlux?
A estrutura *Spring WebFlux* é uma pilha da Web reativa totalmente assíncrona e sem bloqueio que permite o gerenciamento de um grande número de conexões simultâneas. O WebFlux suporta a pressão de retorno do *Reactive Streams* e é executado em servidores como o Netty. Ele nos permite dimensionar verticalmente os serviços para lidar com a maior carga no mesmo hardware.

O que é Kafka Reativo?
O Kafka reativo é uma API reativa para o Kafka, baseada no projeto Reactor e na API do produtor / consumidor do Kafka. Ele permite que os dados sejam publicados e consumidos do Kafka usando API funcional com contrapressão sem bloqueio e baixos custos indiretos, o que permite que o Kafka reativo seja integrado a outros sistemas de reatores e forneça um pipeline reativo de ponta a ponta.

Nota: para obter uma compreensão completa do Webflux e do Kafka Reativo, certifique-se de entender a terminologia.

De acordo com a arquitetura mostrada na imagem-1, construiremos um servidor WebFlux usando a estrutura Spring WebFlux e o Kafka reativo, expondo uma API REST para os clientes fazerem solicitações HTTP seguras.

Depois que uma conexão segura é estabelecida entre o cliente e o servidor de fluxo da web, ela consome mensagens dos tópicos do Kafka e envia os dados de forma assíncrona sem fechar a conexão com o cliente, a menos que seja necessário.

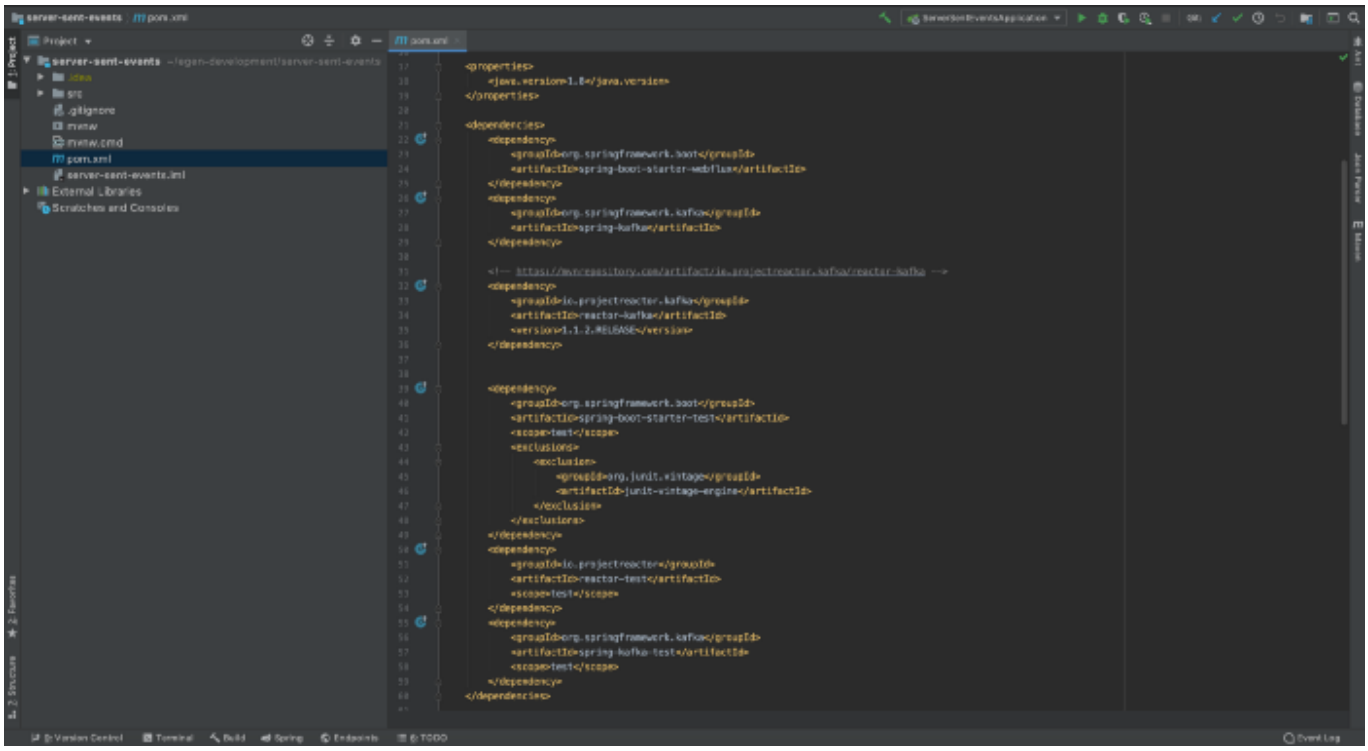
Em vez de criar um produtor reativo de Kafka, aproveitaremos o exemplo do produtor existente no repositório do reator. Além disso, em vez de criar um cliente de fluxo da Web, testaremos a resposta SSE do servidor usando um comando curl no terminal.

Pré-requisitos:

- Java v1.8 +
- Apache Kafka + entendimentos básicos

- Intellij ou Eclipse ou Sprint Tool Suite
- Ferramenta Kafka Conduktor

Vamos criar um aplicativo de inicialização Spring usando as seguintes dependências mostradas abaixo.

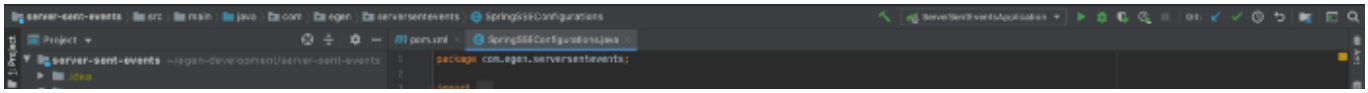


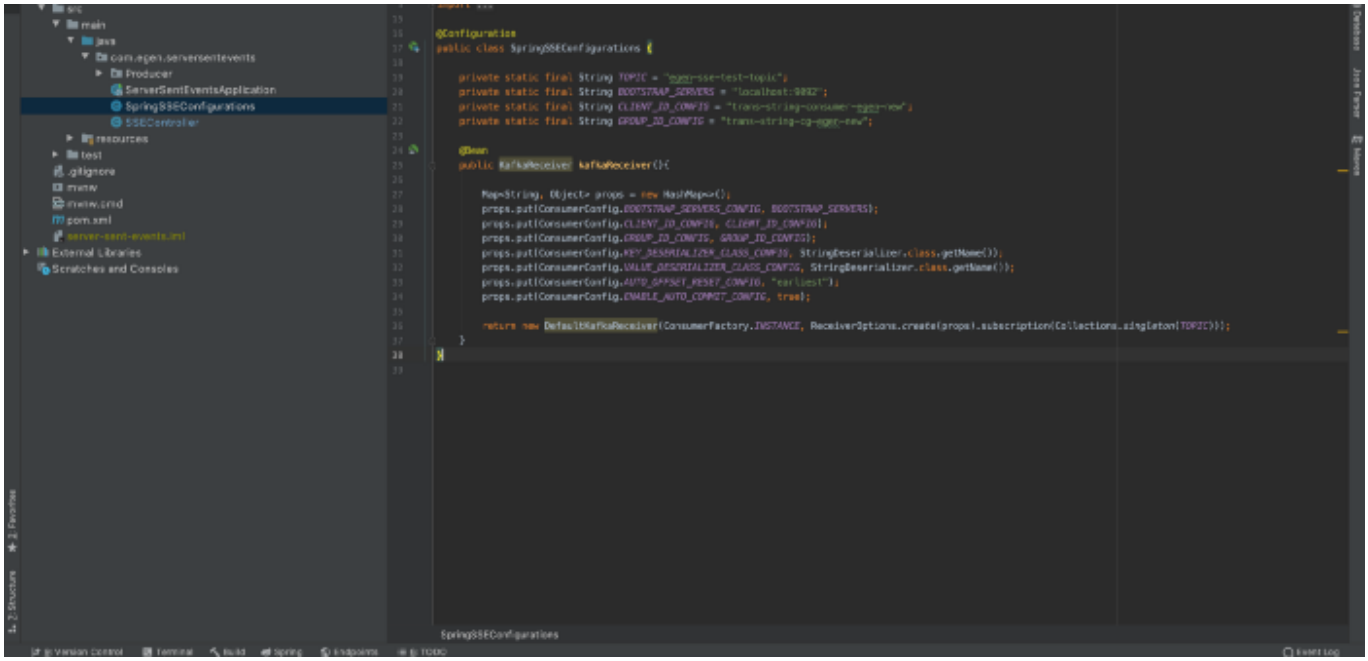
3. pom.xml

Vamos criar uma configuração do receptor Kafka , que é um consumidor, como mostrado abaixo. Ele é configurado com *GROUP_ID_CONFIG* genérico, já que estamos trabalhando no tratamento de um único cliente por enquanto, habilitando *confirmações automáticas* e sempre lendo as mensagens *mais antigas,* mas também podemos atualizá-las para as *mais recentes*.

Se habilitarmos vários clientes, cada cliente poderá receber mensagens do mesmo tópico com base no último deslocamento confirmado.

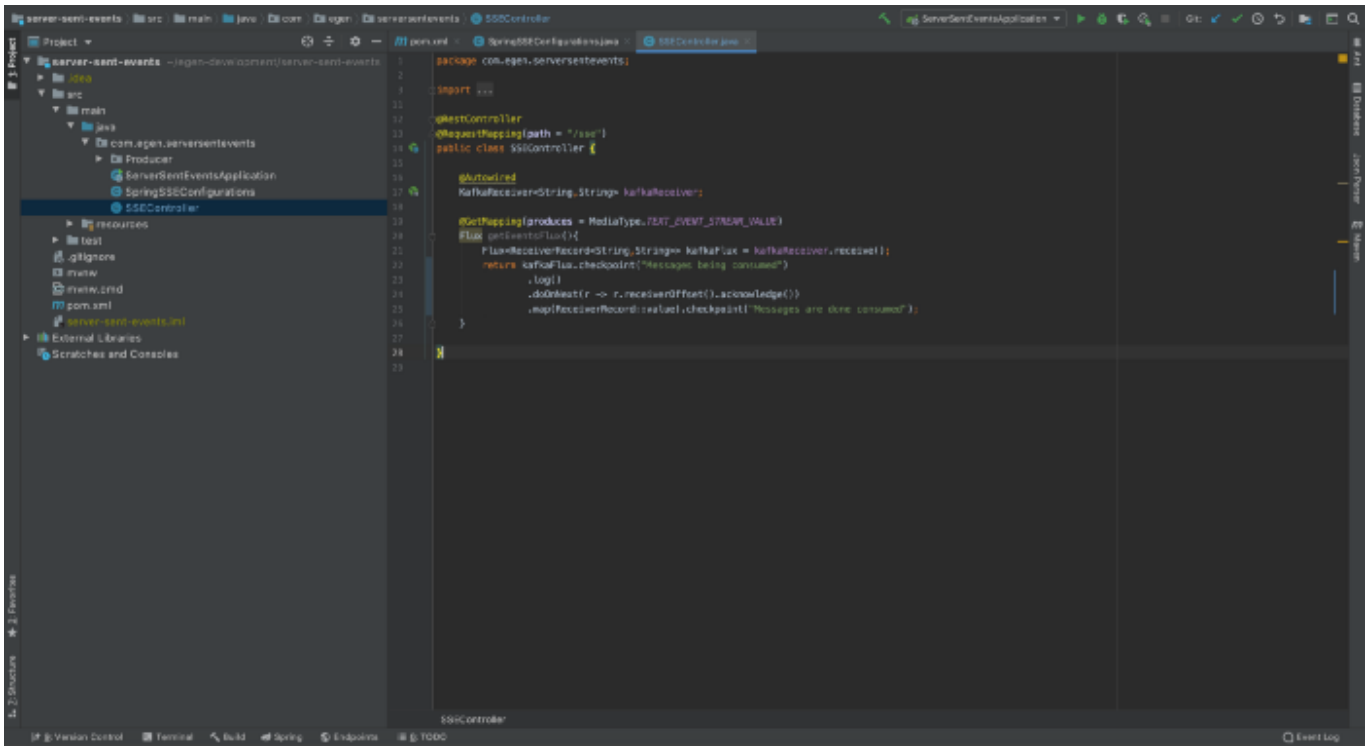
Para manter as coisas simples, lidaremos com desserializadores de string que podem ser estendidos para esquemas JSON / AVRO genéricos.





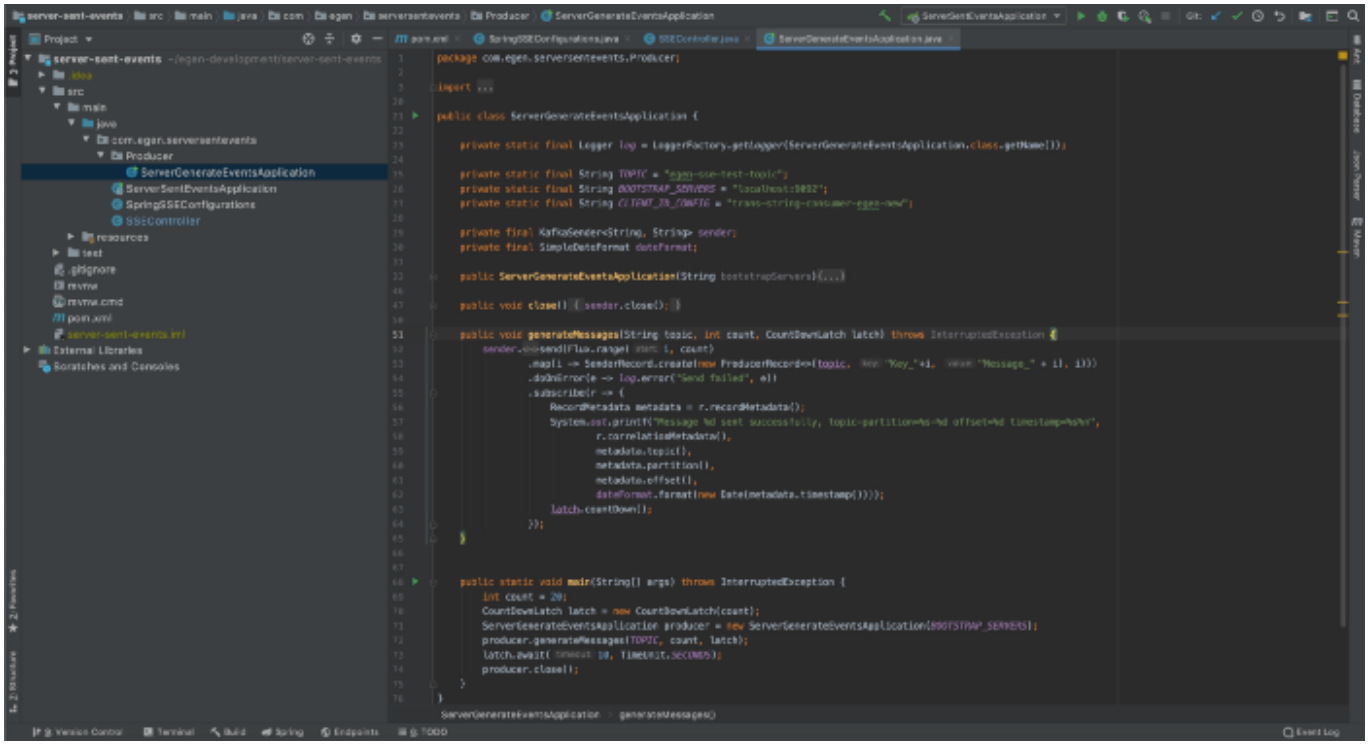
4. Configurações do receptor Kafka

Depois que as configurações estiverem prontas, criaremos um controlador REST que consome mensagens dos tópicos do Kafka e envia respostas como o fluxo de dados. Use MediaType.TEXT_EVENT_STREAM_VALUE como o tipo de conteúdo. Isso informa ao cliente que uma conexão será estabelecida e o fluxo está aberto para o envio de eventos do servidor para o cliente.



5. Controlador REST SSE

Antes de testarmos este aplicativo, aqui está um exemplo de produtor que estamos aproveitando do repositório do reator com o StringSerializer:



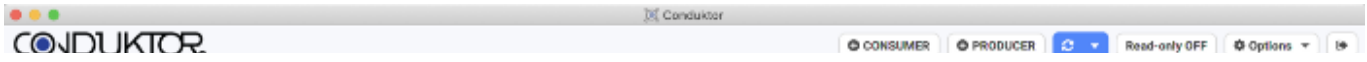
6. Kafka Sender

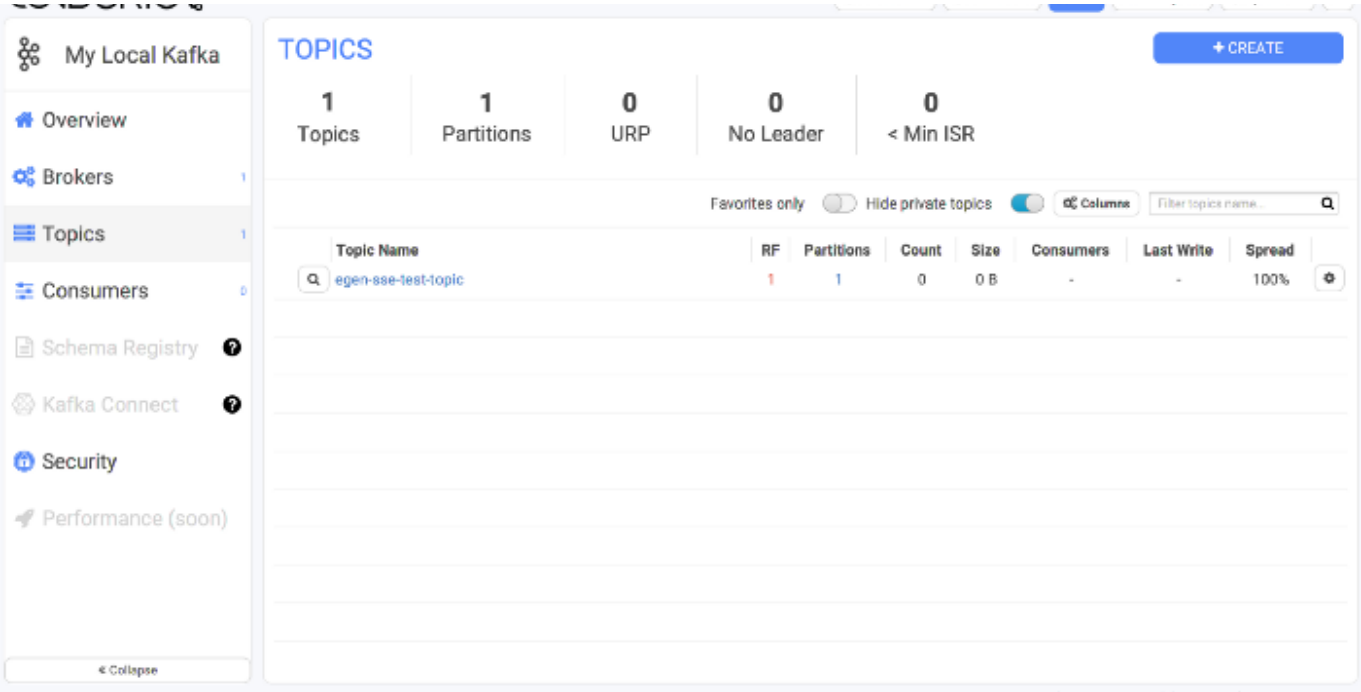
Agora, inicie o servidor Kafka no host local: 9092 e crie um tópico conforme mostrado nas configurações acima.

Os seguintes comandos seriam úteis.

```
$ bin / zookeeper-server-start.sh config / zookeeper.properties

$ bin / kafka-server-start.sh config / server.properties

$ bin / kafka-topics.sh - create - bootstrap-server localhost: 9092 -
fator de replicação 1 - partições 1 - topic <topic_name>

$ bin / kafka-topics.sh - list - bootstrap-server localhost: 9092
```

A imagem abaixo mostra o tópico criado com a partição 1 e sem que nenhuma mensagem (Contagem = 0) seja produzida no Kafka Sender.

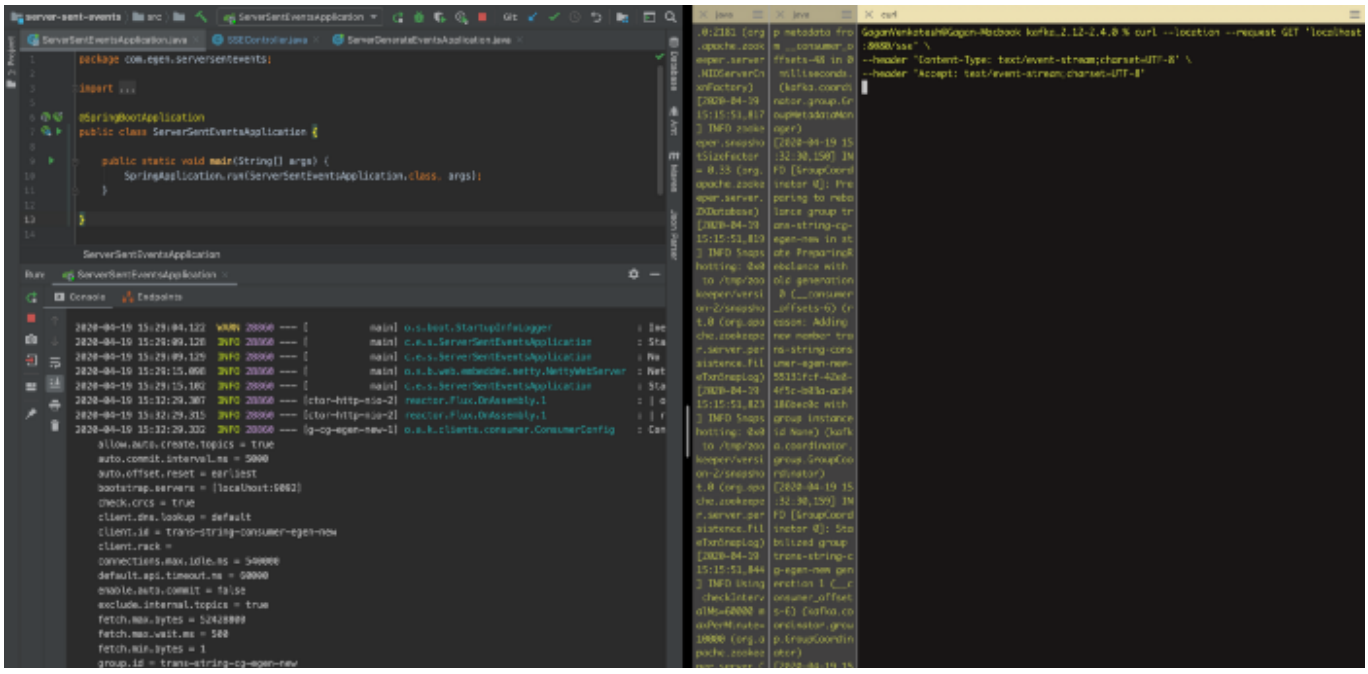


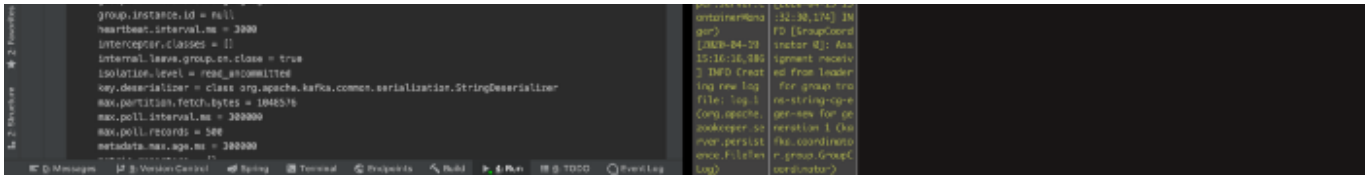


7. Condutor

Em seguida, vamos executar o aplicativo Spring-boot no localhost: 8080 e fazer uma solicitação de amostra para este servidor usando um comando curl no terminal, que mantém a conexão ativa.

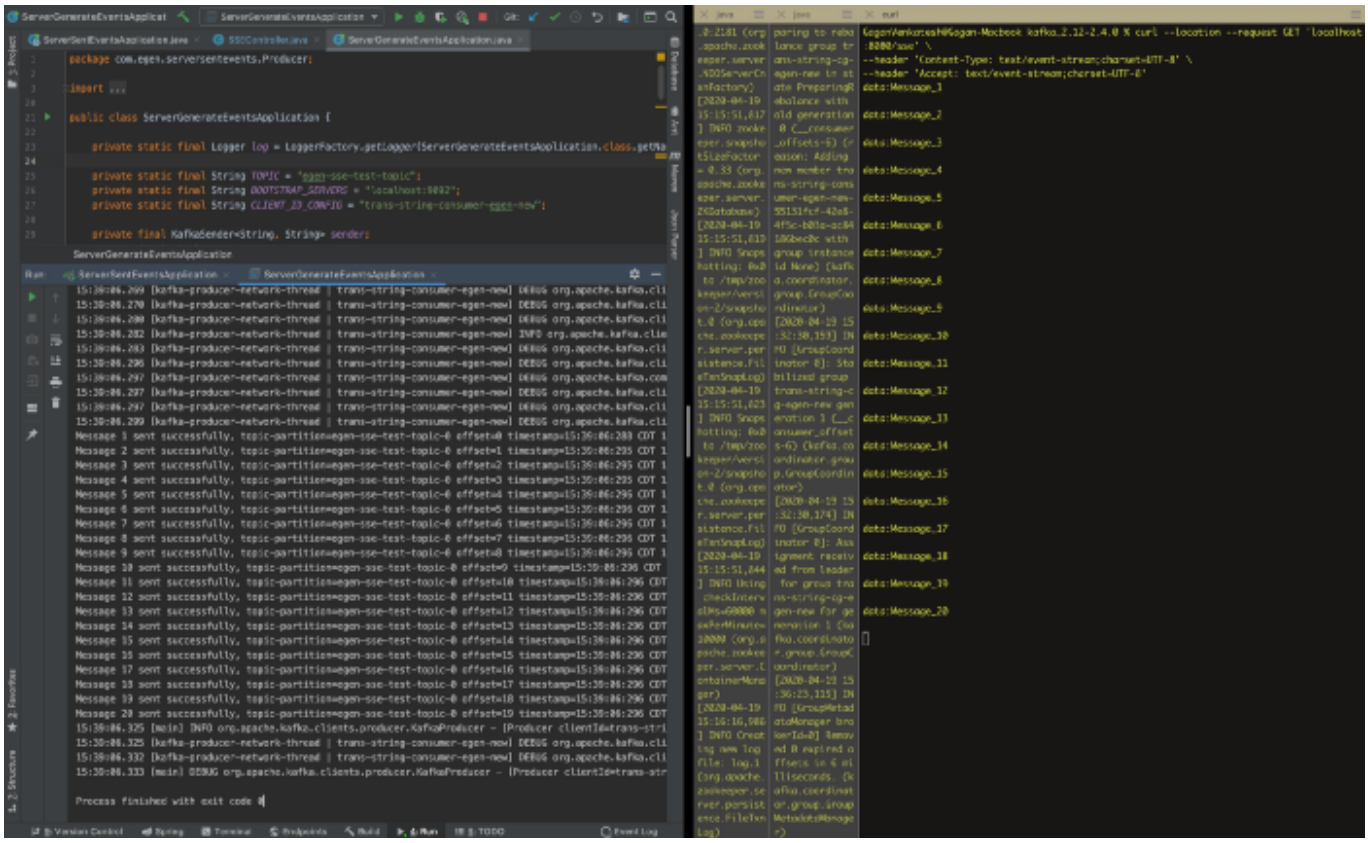
```
curl - location - request GET 'localhost: 8080 / sse' \
- header 'Tipo de conteúdo: text / event-stream; charset = UTF-8' \
- header 'Accept: text / event-stream; charset = UTF-8 "
```





8. O aplicativo de inicialização Spring e o consumidor Kafka estão registrados.

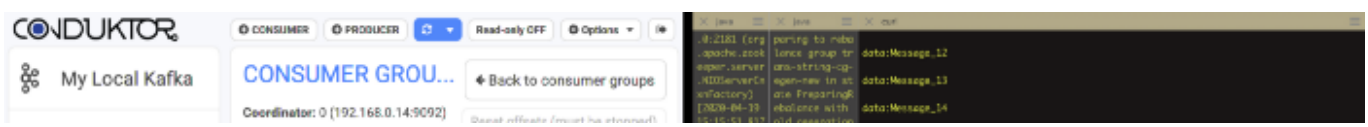
Depois que o comando curl é executado no terminal, um receptor Kafka é registrado (como mostrado no console acima). Agora, vamos colocar alguns dados no tópico Kafka executando o Kafka Sender e ver como os dados estão sendo recebidos no terminal, que atua como cliente aqui.

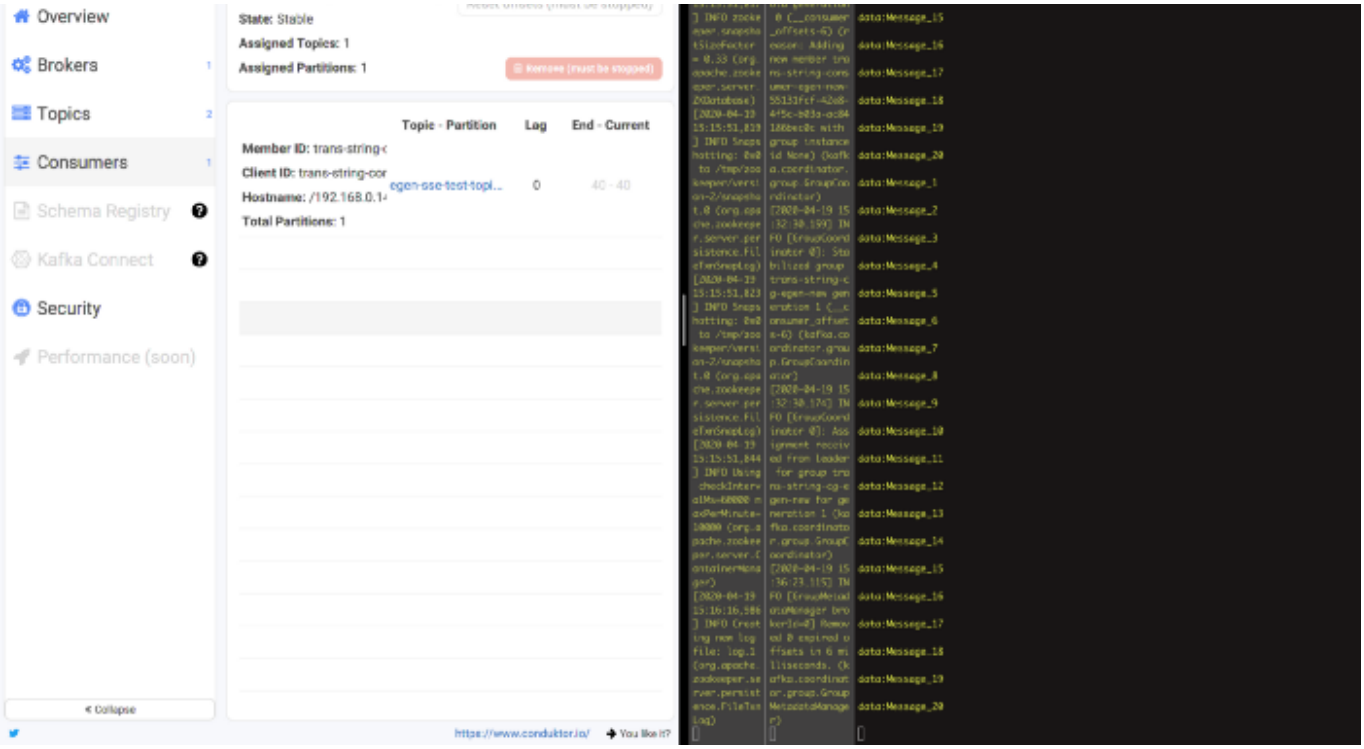


9. Eventos enviados pelo servidor recebidos

Como podemos ver na Imagem-9, uma vez que as mensagens são publicadas no tópico, os dados são enviados para o terminal - que é o Server-Sent Events (SSE).

A Imagem-10 mostra a ferramenta Conduktor rastreando o consumidor e exibindo as mensagens sendo consumidas pelo respectivo cliente.





10. Condutor

Podemos ver que 40 mensagens foram publicadas e todas as mensagens foram consumidas (Lag = 0, End-Current = 40-40) e enviadas ao cliente, conforme mostrado no terminal.

Esse projeto pode ser expandido para oferecer suporte a vários clientes, tornando GROUP_ID_CONFIG exclusivo e configurando as compensações para as mais recentes para cada cliente. Isso cria um novo grupo de consumidores para cada cliente, mantendo a conexão ativa e transmitindo os dados de forma assíncrona. Caso algum cliente perca a conexão com o servidor e consiga restabelecer uma conexão segura após a interrupção, o cliente retornará ao grupo de consumidores existente e receberá dados do deslocamento anterior confirmado.

Essa arquitetura também pode ser usada para criar um agendador em lote para consumir e transferir a mensagem para os arquivos e copiá-los com segurança nos serviços em nuvem, permitindo que o cliente acesse arquivos para processamento adicional.

Se você gostou deste passo-a-passo básico do SSE usando o Spring WebFlux e o Kafka Reativo, sinta-se à vontade para compartilhar e seguir nossa publicação!

> *Consulte o código aqui .*

Obrigado a Vishwa Teja Vangari e Meera Vyas .

Eventos Enviados pelo Servidor Spring Webflux Projeto Reator Kafka Dados