FLOWX.AI

PLATFORM DEEP DIVE / Integrations

Contents

PLATFORM DEEP DIVE / Integrations / Creating a Kafka consumer

TIP

This guide focuses on creating a

The fallback content to display on prerendering

consumer using Spring Boot.

Here are some tips, including the required configurations and code samples, to help you implement a Kafka consumer in Java.

Required dependencies

Ensure that you have the following dependencies in your project:

```
<dependency>
    <groupId>org.springframework.kafka</groupId>
    <artifactId>spring-kafka</artifactId>
</dependency>

<dependency>
    <groupId>io.strimzi</groupId>
    <artifactId>kafka-oauth-client</artifactId>
    <version>0.6.1</version>
</dependency>

<dependency>
    <groupId>org.apache.kafka</groupId>
    <artifactId>kafka-clients</artifactId>
    <version>2.5.1</version>
</dependency>

<dependency>
    <groupId>io.opentracing.contrib</groupId>
    <artifactId>opentracing-kafka-client</artifactId>
    <version>0.1.13</version>
</dependency>
```

Configuration

Ensure that you have the following configuration in your `application.yml` or `application.properties` file:

```
spring.kafka:
      bootstrap-servers: URL_OF_THE_KAFKA_SERVER
      consumer:
        group-id: ADD_CONSUMER_NAME
        auto-offset-reset: earliest
        key-deserializer:
org.apache.kafka.common.serialization.StringDeserializer
        value-deserializer:
org.apache.kafka.common.serialization.StringDeserializer
        properties:
          interceptor:
            classes:
io.opentracing.contrib.kafka.TracingConsumerInterceptor
          security.protocol: "SASL_PLAINTEXT"
          sasl.mechanism: "OAUTHBEARER"
          sasl.jaas.config:
"org.apache.kafka.common.security.oauthbearer.OAuthBearerLoginM
required ;"
          sasl.login.callback.handler.class:
io.strimzi.kafka.oauth.client.JaasClientOauthLoginCallbackHand

kafka:
  consumerThreads: 1
  authorizationExceptionRetryInterval: 10
  ADD_NEEDED_TOPIC_NAMES_HERE
```

Code sample for a Kafka Listener

Here's an example of a Kafka listener method:

```
@KafkaListener(topics = "TOPIC_NAME_HERE")
public void listen(ConsumerRecord<String, String> record)
throws JsonProcessingException {

  SomeDTO request = objectMapper.readValue(record.value(),
SomeDTO.class);

  // process received DTO
}
```

Make sure to replace "TOPIC_NAME_HERE" with the actual name of the Kafka topic you want to consume from. Additionally, ensure that you have the necessary serialization and deserialization logic based on your specific use case.

Was this page helpful?

PLATFORM DEEP DIVE / Integrations / Creating a Kafka producer

> **TIP**
>
> This guide focuses on creating a
>
> The fallback content to display on prerendering
> producer using Spring Boot.

Here are some tips, including the required configurations and code samples, to help you implement a Kafka producer in Java.

Required dependencies

Ensure that you have the following dependencies in your project:

```
<dependency>
    <groupId>org.springframework.kafka</groupId>
    <artifactId>spring-kafka</artifactId>
</dependency>

<dependency>
    <groupId>io.strimzi</groupId>
    <artifactId>kafka-oauth-client</artifactId>
    <version>0.6.1</version>
</dependency>

<dependency>
    <groupId>org.apache.kafka</groupId>
    <artifactId>kafka-clients</artifactId>
    <version>2.5.1</version>
</dependency>

<dependency>
    <groupId>io.opentracing.contrib</groupId>
    <artifactId>opentracing-kafka-client</artifactId>
    <version>0.1.13</version>
</dependency>
```

Configuration

Ensure that you have the following configuration in your `application.yml` or `application.properties` file:

```
spring.kafka:
      bootstrap-servers: URL_OF_THE_KAFKA_SERVER
      producer:
        key-deserializer:
org.apache.kafka.common.serialization.StringSerializer
        value-serializer:
org.springframework.kafka.support.serializer.JsonSerializer
        properties:
          interceptor:
            classes:
io.opentracing.contrib.kafka.TracingProducerInterceptor
          security.protocol: "SASL_PLAINTEXT"
          sasl.mechanism: "OAUTHBEARER"
          sasl.jaas.config:
"org.apache.kafka.common.security.oauthbearer.OAuthBearerLoginM
required ;"
          sasl.login.callback.handler.class:
io.strimzi.kafka.oauth.client.JaasClientOauthLoginCallbackHandl

kafka:
  authorizationExceptionRetryInterval: 10
  ADD_NEEDED_TOPIC_NAMES_HERE # make sure to use the correct na
pattern for topics used to send data to the FLOWX Engine
```

Code sample for a Kafka producer

DANGER

Ensure that you have the necessary KafkaTemplate bean autowired in your producer class. The sendMessage method demonstrates how to send a message to a Kafka topic with the specified headers and payload. Make sure to include all the received Kafka headers in the response that is sent back to the

The fallback content to display on prerendering

.

```
private final KafkaTemplate<String, Object> kafkaTemplate;

public void sendMessage(String topic, Headers headers,
Object payload) {
  ProducerRecord<String, Object> producerRecord = new
ProducerRecord<>(topic, payload);
  // make sure to send all the received headers back to the
FlowX Engine
  headers.forEach(header ->
producerRecord.headers().add(header));
  kafkaTemplate.send(producerRecord);
}
```

Was this page helpful?

PLATFORM DEEP DIVE / Integrations / Jaeger setup for microservices

The scope of this document is to present some basic information on how to include Jaeger tracing into a Java based project.

Required dependencies

```
<dependency>
    <groupId>io.jaegertracing</groupId>
    <artifactId>jaeger-client</artifactId>
    <version>1.4.0</version>
</dependency>
<dependency>
    <groupId>io.opentracing.contrib</groupId>
    <artifactId>opentracing-kafka-client</artifactId>
    <version>0.1.13</version>
</dependency>
```

Needed configs

Add Kafka interceptors for Tracing

```
kafka:
    producer:
        properties:
            interceptor:
                classes:
io.opentracing.contrib.kafka.TracingProducerInterceptor

kafka:
    consumer:
        properties:
```

```
            interceptor:
                classes:
io.opentracing.contrib.kafka.TracingConsumerInterceptor
```

Extract Jaeger span context from received Kafka message

```
@KafkaListener(topics = "${TOPIC_NAME}")
public void listen(ConsumerRecord<String, String> record) {
    // some code
    SpanContext spanContext =
TracingKafkaUtils.extractSpanContext(record.headers(),
tracer);
    // some other code
}
```

TIP

Use this context to create child spans of it and log events from adapter:

```
Span span =
tracer.buildSpan(JAEGER_SPAN_NAME).asChildOf(spanContext).
```

Send span context with outgoing Kafka messages

```
ProducerRecord<String, Object> producerRecord = new
ProducerRecord<>(responseTopic, responseMessage);

TracingKafkaUtils.inject(span.context(),
producerRecord.headers(), tracer);
```

```
kafkaTemplate.send(producerRecord);
```

Was this page helpful?

PLATFORM DEEP DIVE / Integrations / Mock integrations

If you need to test the business process flow but haven't completed all integrations, you can still do so by utilizing the mock integrations server included in the platform.

Setup

To begin, configure the microservice's DB settings to use a Postgres DB. Then, deploy the mocked adapter microservice.

Adding a new integration

Setting up a mocked integration requires only one step: adding a mock Kafka request and response.

You have two options for accomplishing this:

1. Add the information directly to the DB.
2. Use the provided API.

For each Kafka message exchange between the engine and the integration, you need to create a separate entry.

POST MOCK_ADAPTER_URL/api/kafka-exchanges/

GET MOCK_ADAPTER_URL/api/kafka-exchanges/

Was this page helpful?