simple real-time scenario using Apache Kafka with two Spring Boot microservices. This example demonstrates a producer service that sends messages to a Kafka topic and a consumer service that listens to that topic.

**Scenario**

Imagine a scenario where a user submits an order, and you want to send an order confirmation via email. The producer service handles order submissions, and the consumer service sends email notifications.

**Microservices Setup**

**1. Producer Service (Order Service)**

**Dependencies (pom.xml)**

xml

Copy code

<dependency>

<groupId>org.springframework.kafka</groupId>

<artifactId>spring-kafka</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

**application.properties**

properties

Copy code

spring.kafka.bootstrap-servers=localhost:9092

spring.kafka.producer.key-serializer=org.apache.kafka.common.serialization.StringSerializer

spring.kafka.producer.value-serializer=org.apache.kafka.common.serialization.StringSerializer

kafka.topic.orders=order-topic

**OrderController.java**

java

Copy code

@RestController

@RequestMapping("/orders")

public class OrderController {

private final KafkaTemplate<String, String> kafkaTemplate;

@Value("${kafka.topic.orders}")

private String orderTopic;

public OrderController(KafkaTemplate<String, String> kafkaTemplate) {

this.kafkaTemplate = kafkaTemplate;

}

@PostMapping

public ResponseEntity<String> createOrder(@RequestBody String order) {

kafkaTemplate.send(orderTopic, order);

return ResponseEntity.ok("Order submitted successfully");

}

}

**2. Consumer Service (Email Service)**

**Dependencies (pom.xml)**

xml

Copy code

<dependency>

<groupId>org.springframework.kafka</groupId>

<artifactId>spring-kafka</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter</artifactId>

</dependency>

**application.properties**

properties

Copy code

spring.kafka.bootstrap-servers=localhost:9092

spring.kafka.consumer.group-id=email-service-group

spring.kafka.consumer.key-deserializer=org.apache.kafka.common.serialization.StringDeserializer

spring.kafka.consumer.value-deserializer=org.apache.kafka.common.serialization.StringDeserializer

kafka.topic.orders=order-topic

**EmailListener.java**

java

Copy code

@Component

public class EmailListener {

@KafkaListener(topics = "${kafka.topic.orders}", groupId = "email-service-group")

public void listen(String order) {

// Logic to send email

System.out.println("Order received: " + order);

sendEmail(order);

}

private void sendEmail(String order) {

// Implement email sending logic

System.out.println("Email sent for order: " + order);

}

}

**Running the Example**

1. **Start Kafka**: Make sure you have Kafka running locally. You can use Docker to set it up easily.
2. **Run the Producer Service**: Start your Order Service.
3. **Run the Consumer Service**: Start your Email Service.

**Testing the Flow**

You can test the producer service by sending a POST request to the /orders endpoint with a sample order in the body (e.g., "Order123"). The consumer service should automatically receive the message and process it, simulating an email being sent.

**Summary**

This example showcases a basic setup of two microservices communicating through Kafka. The producer service handles order submissions, while the consumer service listens for those orders and processes them (e.g., sending an email). You can expand on this by implementing real email-sending logic or more complex business rules.