**APACHE KAFKA R**EAL TIME CASE STUDY

A simple real-time scenario using Apache Kafka with Spring Boot, involving two microservices: a **Producer** service that sends messages and a **Consumer** service that processes those messages.

**Scenario: User Registration Notification**

* **Microservice 1 (User Service)**: When a new user registers, it sends a message to a Kafka topic.
* **Microservice 2 (Notification Service)**: It listens to the Kafka topic and sends a notification (e.g., email or SMS) to the registered user.

**Scenario Overview**

In this example, the **Producer** service will generate user registration events and publish them to a Kafka topic. The **Consumer** service will listen to this topic and process the events (e.g., logging or storing user information).

**Setup**

1. **Dependencies**: Add the following dependencies to your pom.xml for both microservices:

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>

1. **Application Properties**: Configure application.properties for both services.

**Producer (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.user-registration=user-registration-topic

**Consumer (application.properties)**:

properties

Copy code

spring.kafka.bootstrap-servers=localhost:9092

spring.kafka.consumer.group-id=user-registration-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.user-registration=user-registration-topic

**Producer Service**

1. **Create a Spring Boot Application**:

java

Copy code

@SpringBootApplication

public class UserRegistrationProducerApplication {

public static void main(String[] args) {

SpringApplication.run(UserRegistrationProducerApplication.class, args);

}

}

1. **Kafka Producer Service**:

java

Copy code

@Service

public class UserRegistrationProducer {

private final KafkaTemplate<String, String> kafkaTemplate;

private final String topic;

@Autowired

public UserRegistrationProducer(KafkaTemplate<String, String> kafkaTemplate,

@Value("${kafka.topic.user-registration}") String topic) {

this.kafkaTemplate = kafkaTemplate;

this.topic = topic;

}

public void sendUserRegistration(String message) {

kafkaTemplate.send(topic, message);

}

}

1. **REST Controller**:

java

Copy code

@RestController

@RequestMapping("/api/users")

public class UserController {

private final UserRegistrationProducer producer;

@Autowired

public UserController(UserRegistrationProducer producer) {

this.producer = producer;

}

@PostMapping("/register")

public ResponseEntity<String> registerUser(@RequestBody String user) {

producer.sendUserRegistration(user);

return ResponseEntity.ok("User registration event sent.");

}

}

**Consumer Service**

1. **Create a Spring Boot Application**:

java

Copy code

@SpringBootApplication

public class UserRegistrationConsumerApplication {

public static void main(String[] args) {

SpringApplication.run(UserRegistrationConsumerApplication.class, args);

}

}

1. **Kafka Consumer Service**:

java

Copy code

@Service

public class UserRegistrationConsumer {

@KafkaListener(topics = "${kafka.topic.user-registration}", groupId = "user-registration-group")

public void listen(String message) {

System.out.println("Received user registration: " + message);

// Process the message (e.g., store user info in database)

}

}

**Running the Application**

1. **Start Kafka**: Ensure that Kafka is running on localhost:9092.
2. **Run Both Microservices**: Start the Producer and Consumer applications.
3. **Testing**:
   * Use a tool like Postman or cURL to send a POST request to the Producer:

bash

Copy code

POST http://localhost:8080/api/users/register

Body: {"username": "john\_doe", "email": "john@example.com"}

* + The Consumer should log the received message.

**Conclusion**

This example demonstrates a simple real-time communication pattern using Apache Kafka in a microservices architecture. The Producer service sends user registration events, and the Consumer service processes those events. You can extend this further by adding error handling, message formats (JSON), and more complex processing logic