Here’s how to set up **messaging between services** and integrate specific APIs for your **Notification** and **Order Services**.

**1. Messaging Between Services**

**Use RabbitMQ (or Kafka) for Asynchronous Communication**

**Why Messaging?**

* Decouples services for scalability and fault tolerance.
* Enables event-driven architecture (e.g., "Order Placed" triggers a notification).

**Step 1: Install and Configure RabbitMQ**

* Install RabbitMQ on your local machine or use a managed service like AWS MQ or CloudAMQP.
* Start RabbitMQ:

bash

Copy code

rabbitmq-server

* Access the RabbitMQ Management UI at http://localhost:15672 (default credentials: guest/guest).

**Step 2: Add RabbitMQ to Spring Boot**

1. Add dependencies in pom.xml:

xml

Copy code

<dependency>

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

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

</dependency>

1. Configure RabbitMQ in application.yml:

yaml

Copy code

spring:

rabbitmq:

host: localhost

port: 5672

username: guest

password: guest

1. Define a queue and exchange in both **Order Service** and **Notification Service**.

Example: Order Service sends messages to a queue named order.notifications.

**Step 3: Order Service - Publish Messages**

1. Create a configuration class:

java

Copy code

@Configuration

public class RabbitMQConfig {

public static final String EXCHANGE = "order-exchange";

public static final String ROUTING\_KEY = "order.notifications";

@Bean

public TopicExchange topicExchange() {

return new TopicExchange(EXCHANGE);

}

@Bean

public Queue queue() {

return new Queue("order.notifications");

}

@Bean

public Binding binding(Queue queue, TopicExchange exchange) {

return BindingBuilder.bind(queue).to(exchange).with(ROUTING\_KEY);

}

}

Create a service to send messages:

java

Copy code

@Service

public class OrderEventPublisher {

private final RabbitTemplate rabbitTemplate;

@Autowired

public OrderEventPublisher(RabbitTemplate rabbitTemplate) {

this.rabbitTemplate = rabbitTemplate;

}

public void publishOrderCreatedEvent(OrderEvent event) {

rabbitTemplate.convertAndSend("order-exchange", "order.notifications", event);

}

}

Define the event model:

java

Copy code

public class OrderEvent {

private String orderId;

private String message;

// Getters and setters

}

Use the service in your order creation logic:

java

Copy code

OrderEvent event = new OrderEvent(orderId, "Order Created");

orderEventPublisher.publishOrderCreatedEvent(event);

**Step 4: Notification Service - Consume Messages**

1. Configure RabbitMQ in the Notification Service (similar to the Order Service).
2. Create a listener for messages:

java

Copy code

@Service

public class NotificationListener {

@RabbitListener(queues = "order.notifications")

public void handleOrderEvent(OrderEvent event) {

System.out.println("Received message: " + event.getMessage());

// Send email or SMS notification

}

}

**2. Notification Service - Send Email/SMS**

**Email Notifications with Spring Mail**

1. Add dependency in pom.xml:

xml

Copy code

<dependency>

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

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

</dependency>

1. Configure email in application.yml:

yaml

Copy code

spring:

mail:

host: smtp.gmail.com

port: 587

username: your-email@gmail.com

password: your-password

properties.mail.smtp:

auth: true

starttls.enable: true

1. Create a service to send emails:

java

Copy code

@Service

public class EmailService {

@Autowired

private JavaMailSender mailSender;

public void sendEmail(String to, String subject, String body) {

SimpleMailMessage message = new SimpleMailMessage();

message.setTo(to);

message.setSubject(subject);

message.setText(body);

mailSender.send(message);

}

}

1. Use the service in your listener:

java

Copy code

@Autowired

private EmailService emailService;

public void handleOrderEvent(OrderEvent event) {

emailService.sendEmail("customer@example.com", "Order Update", event.getMessage());

}

**SMS Notifications with Twilio**

1. Add dependency in pom.xml:

xml

Copy code

<dependency>

<groupId>com.twilio</groupId>

<artifactId>twilio</artifactId>

<version>8.29.1</version>

</dependency>

1. Configure Twilio in your application:

yaml

Copy code

twilio:

account-sid: YOUR\_ACCOUNT\_SID

auth-token: YOUR\_AUTH\_TOKEN

phone-number: YOUR\_TWILIO\_PHONE\_NUMBER

1. Create a service to send SMS:

java

Copy code

@Service

public class SmsService {

@Value("${twilio.account-sid}")

private String accountSid;

@Value("${twilio.auth-token}")

private String authToken;

@Value("${twilio.phone-number}")

private String twilioPhoneNumber;

@PostConstruct

public void init() {

Twilio.init(accountSid, authToken);

}

public void sendSms(String to, String message) {

Message.creator(new PhoneNumber(to), new PhoneNumber(twilioPhoneNumber), message).create();

}

}

1. Use the service in your listener:

java

Copy code

@Autowired

private SmsService smsService;

public void handleOrderEvent(OrderEvent event) {

smsService.sendSms("+1234567890", event.getMessage());

}

**3. Order Service - Payment Gateway**

**Integrate Stripe**

1. Add Stripe dependency:

xml

Copy code

<dependency>

<groupId>com.stripe</groupId>

<artifactId>stripe-java</artifactId>

<version>22.11.0</version>

</dependency>

1. Initialize Stripe:

java

Copy code

@Service

public class PaymentService {

@Value("${stripe.api-key}")

private String apiKey;

@PostConstruct

public void init() {

Stripe.apiKey = apiKey;

}

public PaymentIntent createPaymentIntent(int amount, String currency) throws StripeException {

Map<String, Object> params = new HashMap<>();

params.put("amount", amount);

params.put("currency", currency);

return PaymentIntent.create(params);

}

}

1. Use the service during order creation:

java

Copy code

PaymentIntent paymentIntent = paymentService.createPaymentIntent(5000, "usd");

System.out.println("Payment Intent Created: " + paymentIntent.getId());