Spring Boot Microservices Tutorial - Part 7

In Part 7 of this Spring Boot Microservices Tutorial series, we will integrate Kafka into our project and learn how to build Event-Driven Microservices with Spring Boot and Kafka.

What are Event Driven Microservices?

Event-driven microservices architecture is a way of building applications, where the systems communicate by publishing and consuming events, these events are available whenever other consumers need to read them at any time.

in simple way:

Event-Driven Microservices refer to a software architecture where **events** are used as the main form of communication between independent services. In this approach, services publish and consume events to exchange information and trigger actions.

Apache Kafka is a distributed messaging and streaming platform used frequently in the industry to implement Event-Driven Architecture.

Kafka, RabbitMQ → both are messaging system only.

Installing Apache Kafka through Docker

We will use Docker to install Apache Kafka together with Zookeeper. We will also use a Kafka UI to see the topics and messages in our Kafka Cluster using the Kafka UI project. Here is how the Docker compose file looks like in the **order-service docker-compose.yaml** file:

```yaml
version: '4'
services:
 mysql:
    image: mysql:8.3.0
    container_name: mysql
    environment:
      MYSQL_ROOT_PASSWORD: mysql
    ports:
      - "3306:3306"
    volumes:
      - ./mysql:/var/lib/mysql
      - ./docker/mysql/init.sql:/docker-entrypoint-initdb.d/init.sql
  zookeeper:
    image: confluentinc/cp-zookeeper:7.5.0
    hostname: zookeeper
    container_name: zookeeper
    ports:
      - "2181:2181"
    environment:
      ZOOKEEPER_CLIENT_PORT: 2181
      ZOOKEEPER_TICK_TIME: 2000

  broker:
    image: confluentinc/cp-kafka:7.5.0
    container_name: broker
    ports:
      - "9092:9092"
      - "29092:29092"
    depends_on:
      - zookeeper
    environment:
      KAFKA_BROKER_ID: 1
      KAFKA_ZOOKEEPER_CONNECT: 'zookeeper:2181'
      KAFKA_LISTENER_SECURITY_PROTOCOL_MAP: PLAINTEXT:PLAINTEXT,PLAINTEXT_HOST:PLAINTEXT
      KAFKA_ADVERTISED_LISTENERS:
PLAINTEXT://broker:29092,PLAINTEXT_HOST://localhost:9092
      KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1

  schema-registry:
    image: confluentinc/cp-schema-registry:7.5.0
    hostname: schema-registry
    container_name: schema-registry
    depends_on:
      - broker
    ports:
      - "8085:8081"
    environment:
      SCHEMA_REGISTRY_HOST_NAME: schema-registry
      SCHEMA_REGISTRY_KAFKASTORE_BOOTSTRAP_SERVERS: 'broker:29092'
      SCHEMA_REGISTRY_LISTENERS: http://schema-registry:8081
  kafka-ui:
    container_name: kafka-ui
    image: provectuslabs/kafka-ui:latest
    ports:
      - "8086:8080"
    depends_on:
      - broker
      - schema-registry
    environment:
      KAFKA_CLUSTERS_NAME: local
      KAFKA_CLUSTERS_BOOTSTRAPSERVERS: broker:29092
      KAFKA_CLUSTERS_SCHEMAREGISTRY: http://schema-registry:8081
      DYNAMIC_CONFIG_ENABLED: 'true'
```

The main services we use are

- **cp-zookeeper** which is a Zookeeper cluster that is used to orchestrate multiple Kafka clusters.
- **cp-kafka** which is the Kafka server itself
- **cp-schema-registry** is the service we used to define the schema of the messages that are sent between producers and consumers
- Lastly, we have **kafka-ui** which provides a nice UI to allow us to view the Kafka topics that are created and also helps us to view the messages from and send messages to the Kafka topic.

After updating the docker-compose file, just run **docker compose up -d** to start all the services.

After adding docker-compose.yml file we can create Notification project with below dependencies in pom.xml file.

```xml
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-mail</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.kafka</groupId>
        <artifactId>spring-kafka</artifactId>
    </dependency>
    <dependency>
        <groupId>org.projectlombok</groupId>
        <artifactId>lombok</artifactId>
        <optional>true</optional>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>org.springframework.kafka</groupId>
        <artifactId>spring-kafka-test</artifactId>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>org.testcontainers</groupId>
        <artifactId>kafka</artifactId>
        <scope>test</scope>
    </dependency>
</dependencies>
```

After adding dependencies, we have to create OrderPlacedEvent in Notification Service. same class was present in order service. make sure both OrderPlacedEvent class should be in same package name.

```java
@Data
@NoArgsConstructor
public class OrderPlacedEvent {
    private String orderNumber;
    private String email;
```

Application.properties (Notification service)

```properties
spring.application.name=Notification_Service
server.port=8082

#Kafka Consumer Properties

spring.kafka.bootstrap-servers=localhost:9092
spring.kafka.consumer.group-id=notificationService
spring.kafka.consumer.key-
deserializer=org.apache.kafka.common.serialization.StringDeserializer
spring.kafka.consumer.value-
deserializer=org.springframework.kafka.support.serializer.JsonDeserializer
spring.kafka.consumer.properties.spring.json.type.mapping=event:com.lakshmiTech.microserv
ices.order.event.OrderPlacedEvent
spring.kafka.consumer.properties.spring.json.trusted.packages=com.lakshmiTech.microservic
es.order.event
```

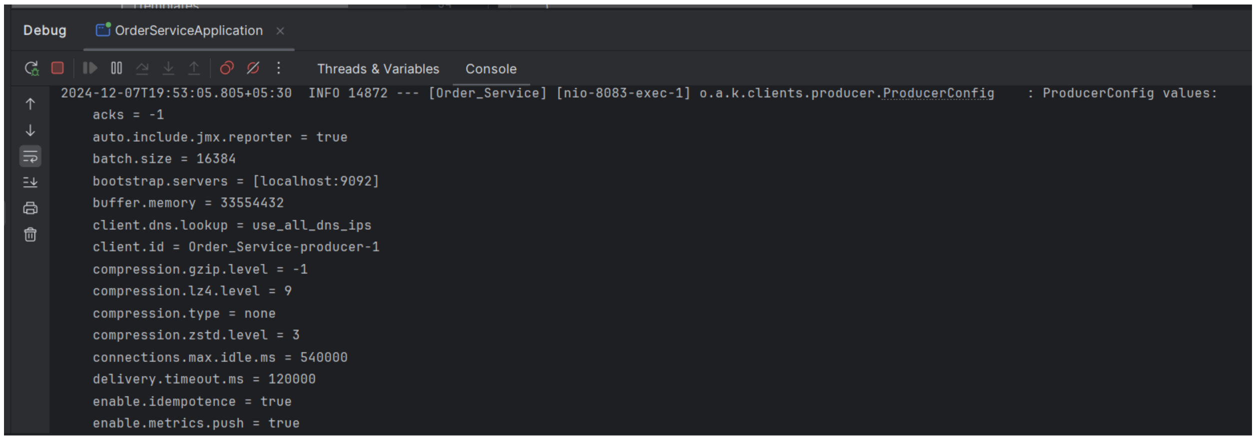
```
#Mail properties

spring.mail.host=sandbox.smtp.mailtrap.io
spring.mail.port=2525
spring.mail.username=ca4235e37c8dcb
spring.mail.password=83f7bb366a0d5a
spring.mail.protocol=smtp
mail.smtp.auth=true
mail.smtp.starttls.enable=true
mail.smtp.starttls.required=true
```

and login to mailtrap.com from google. setFrom mail we have to give which mail connected to mailtrap. and setTo mail we can give anything

NotificationService.java

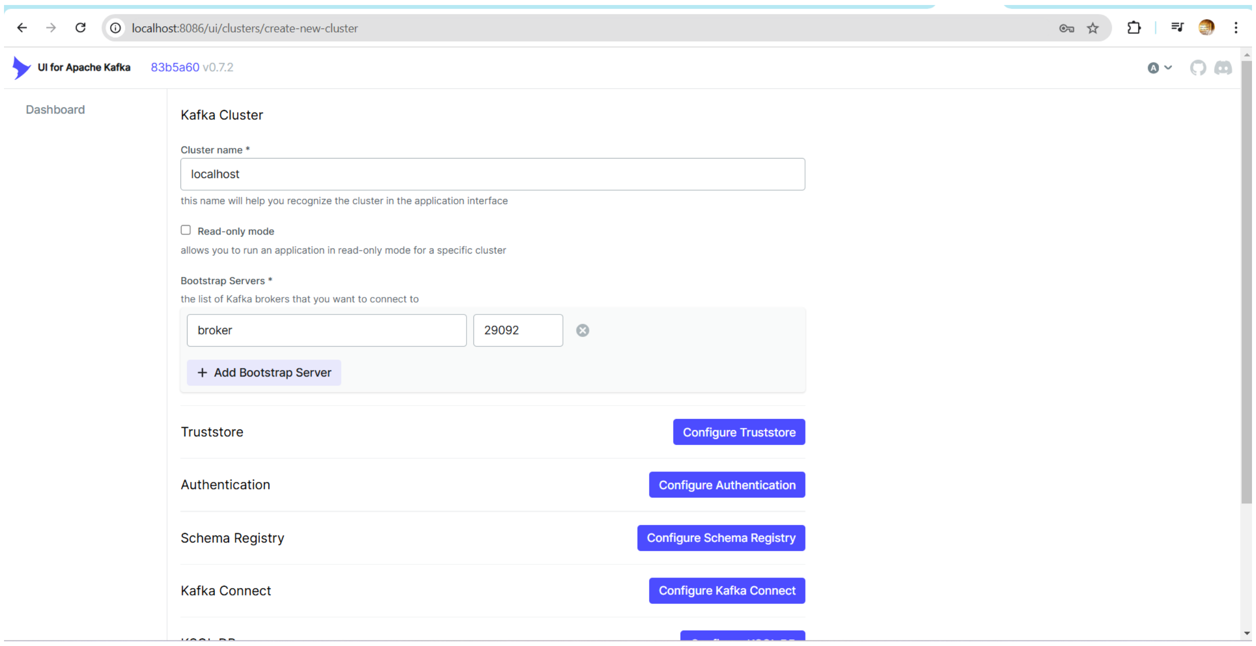
```java
package com.lakshmiTech.microservices.service;

import com.lakshmiTech.microservices.order.event.OrderPlacedEvent;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.kafka.annotation.KafkaListener;
import org.springframework.mail.MailException;
import org.springframework.mail.javamail.JavaMailSender;
import org.springframework.mail.javamail.MimeMessageHelper;
import org.springframework.mail.javamail.MimeMessagePreparator;
import org.springframework.stereotype.Service;



@Service
public class NotificationService {

    private static final Logger log = LoggerFactory.getLogger(NotificationService.class);

    private final JavaMailSender javaMailSender;

    public NotificationService(JavaMailSender javaMailSender) {
        this.javaMailSender = javaMailSender;
    }

    @KafkaListener(topics = "order-placed")
    public void listen(OrderPlacedEvent orderPlacedEvent){
        log.info("Got message from order-placed topic {} ", orderPlacedEvent);

        //send email to the customer
        MimeMessagePreparator messagePreparator = mimeMessage -> {
            MimeMessageHelper messageHelper = new MimeMessageHelper(mimeMessage, true);
            messageHelper.setFrom("ayyamlakshmiperumal420@gmail.com");
            messageHelper.setTo(orderPlacedEvent.getEmail());
            messageHelper.setSubject(String.format("Your Order with OrderNumber %s is
placed successfully", orderPlacedEvent.getOrderNumber()));
            messageHelper.setText(String.format("""
                            Hi

                            Your order with order number %s is now placed successfully.

                            Best Regards
                            Spring Shop
                            """,
                    orderPlacedEvent.getOrderNumber()));
        };
        try {
            javaMailSender.send(messagePreparator);
            log.info("Order Notifcation email sent!!");
        } catch (MailException e) {
            log.error("Exception occurred when sending mail", e);
            throw new RuntimeException("Exception occurred when sending mail to
springshop@email.com", e);
        }
    }
}
```

After adding kafka properties in properties file and add above code in order service.java file and run application. then we can see below output.



```
2024-12-07T19:53:05.805+05:30  INFO 14872 --- [Order_Service] [nio-8083-exec-1] o.a.k.clients.producer.ProducerConfig    : ProducerConfig values: 
	acks = -1
	auto.include.jmx.reporter = true
	batch.size = 16384
	bootstrap.servers = [localhost:9092]
	buffer.memory = 33554432
	client.dns.lookup = use_all_dns_ips
	client.id = Order_Service-producer-1
	compression.gzip.level = -1
	compression.lz4.level = 9
	compression.type = none
	compression.zstd.level = 3
	connections.max.idle.ms = 540000
	delivery.timeout.ms = 120000
	enable.idempotence = true
	enable.metrics.push = true
```

After that go to google type http://localhost:8086 and u can see below page and create cluster with localhost name and broker.



localhost:8086/ui/clusters/create-new-cluster

UI for Apache Kafka 83b5a60 v0.7.2

Dashboard

Kafka Cluster

Cluster name *

localhost

this name will help you recognize the cluster in the application interface

Read-only mode

allows you to run an application in read-only mode for a specific cluster

Bootstrap Servers *

the list of Kafka brokers that you want to connect to

broker 29092

+ Add Bootstrap Server

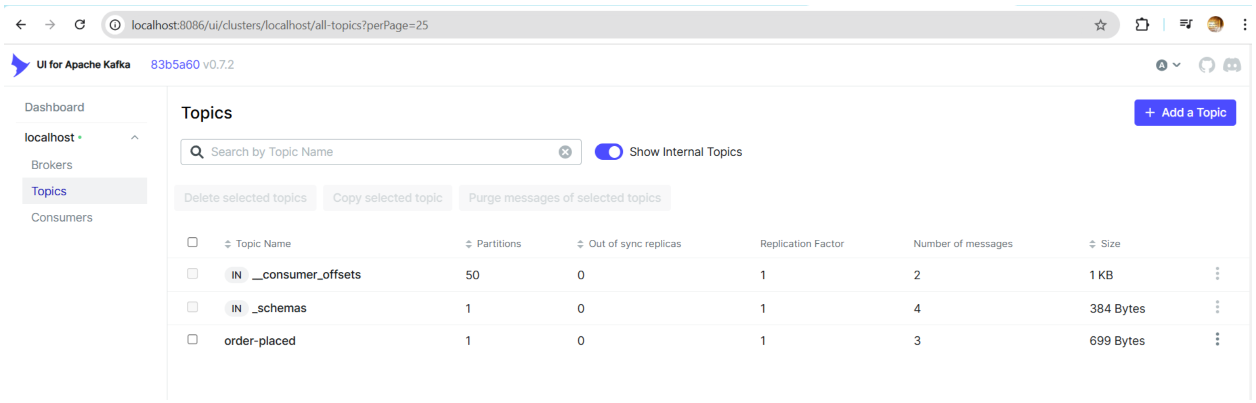
Truststore — Configure Truststore

Authentication — Configure Authentication

Schema Registry — Configure Schema Registry

Kafka Connect — Configure Kafka Connect

After that cluster will be crated and under topics u can see below.



localhost:8086/ui/clusters/localhost/all-topics?perPage=25

UI for Apache Kafka 83b5a60 v0.7.2

Dashboard
localhost
Brokers
Topics
Consumers

Topics

+ Add a Topic

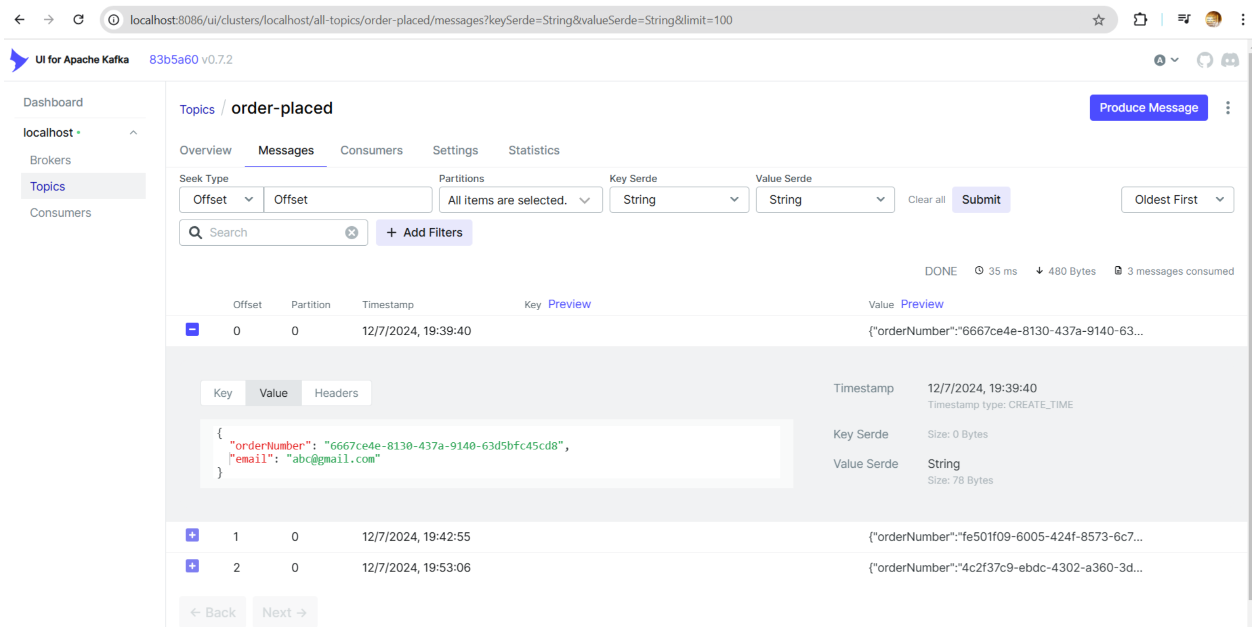
Search by Topic Name

Show Internal Topics

Delete selected topics | Copy selected topic | Purge messages of selected topics

Topic Name	Partitions	Out of sync replicas	Replication Factor	Number of messages	Size
IN __consumer_offsets	50	0	1	2	1 KB
IN _schemas	1	0	1	4	384 Bytes
order-placed	1	0	1	3	699 Bytes

and open messages under order-placed event, u can see below page



Seek Type: Offset | Offset
Partitions: All items are selected.
Key Serde: String
Value Serde: String
Oldest First

DONE 35 ms 480 Bytes 3 messages consumed

	Offset	Partition	Timestamp	Key Preview	Value Preview
−	0	0	12/7/2024, 19:39:40		{"orderNumber":"6667ce4e-8130-437a-9140-63...
+	1	0	12/7/2024, 19:42:55		{"orderNumber":"fe501f09-6005-424f-8573-6c7...
+	2	0	12/7/2024, 19:53:06		{"orderNumber":"4c2f37c9-ebdc-4302-a360-3d...

Key | Value | Headers

```
{
  "orderNumber": "6667ce4e-8130-437a-9140-63d5bfc45cd8",
  "email": "abc@gmail.com"
}
```

Timestamp: 12/7/2024, 19:39:40
Timestamp type: CREATE_TIME
Key Serde: Size: 0 Bytes
Value Serde: String
Size: 78 Bytes

After running the above code, we can see in the output that the message from the `order-service` was successfully received, processed, and an email was also sent.

output:

```
2024-12-08T08:43:14.526+05:30  INFO 12644 --- [Notification_Service] [ntainer#0-0-C-1]
k.c.c.i.ConsumerRebalanceListenerInvoker : [Consumer clientId=consumer-
notificationService-1, groupId=notificationService] Adding newly assigned partitions:
order-placed-0
2024-12-08T08:43:14.573+05:30  INFO 12644 --- [Notification_Service] [ntainer#0-0-C-1]
o.a.k.c.c.internals.ConsumerUtils        : Setting offset for partition order-placed-0 to
the committed offset FetchPosition{offset=9, offsetEpoch=Optional.empty,
currentLeader=LeaderAndEpoch{leader=Optional[localhost:9092 (id: 1 rack: null)],
epoch=0}}
2024-12-08T08:43:14.573+05:30  INFO 12644 --- [Notification_Service] [ntainer#0-0-C-1]
o.s.k.l.KafkaMessageListenerContainer    : notificationService: partitions assigned:
[order-placed-0]
2024-12-08T08:43:17.455+05:30  INFO 12644 --- [Notification_Service] [ntainer#0-0-C-1]
c.l.m.service.NotificationService        : Got message from order-placed topic
OrderPlacedEvent(orderNumber=a1f801da-c96e-4152-a1c2-ed0596bc24da,
email=malalakshmia@gmail.com)
2024-12-08T08:43:21.286+05:30  INFO 12644 --- [Notification_Service] [ntainer#0-0-C-1]
c.l.m.service.NotificationService        : Order Notifcation email sent!!
```

As we can see in our application, the `OrderPlacedEvent` class is duplicated; it is present in both services (`order` and `notification`). While this is not an issue, we should remove the duplication of `OrderPlacedEvent` across the two services.

To achieve this, we can create a schema that automatically generates the class for us. For this purpose, we can use a Kafka registry.

Now we can see how to implement using AVSC file:

Spring Kafka:

We will be using the Spring Kafka project to implement Kafka functionality in our Spring Boot projects, for that add the below dependencies in the order-service project.

```xml
<dependency>
    <groupId>org.springframework.kafka</groupId>
    <artifactId>spring-kafka</artifactId>
</dependency>
<dependency>
<groupId>io.confluent</groupId>
<artifactId>kafka-avro-serializer</artifactId>
<version>7.6.0</version>
</dependency>
<dependency>
<groupId>io.confluent</groupId>
<artifactId>kafka-schema-registry-client</artifactId>
<version>7.6.0</version>
</dependency>
<dependency>
<groupId>org.apache.avro</groupId>
<artifactId>avro</artifactId>
<version>1.11.3</version>
</dependency>
<dependency>
<groupId>org.springframework.kafka</groupId>
<artifactId>spring-kafka-test</artifactId>
<scope>test</scope>
</dependency>
```

along with above dependencies we have to add below repository tag as well in pom.xml file. then dependencies will be downloaded.

```xml
<repositories>
    <repository>
        <id>confluent</id>
        <url>https://packages.confluent.io/maven/</url>
    </repository>
</repositories>
```

The above dependencies not only add the spring-kafka functionality but also bring in dependencies to work with **schema-registry**. We will define our schema in avro format, for that reason we need to also add the **avro** and **kafka-avro-serializer** dependencies.

After adding the above dependencies, now it's time to implement the logic to send an event to the kafka topic whenever there is an order placed in the **order-service**. We will first start by defining the avro-schema of the event we want to send. And we will define the schema in a .avsc file, avsc is the format to define the Avro schema, let's add the below file under **src/main/resources/avro** folder.

order-placed.avsc

```
{
  "type": "record",
  "name": "OrderPlacedEvent",
  "namespace": "com.techie.microservices.order.event",
  "fields": [
    { "name": "orderNumber", "type": "string" },
    { "name": "email", "type": "string" },
    { "name": "firstName", "type": "string" },
    { "name": "lastName", "type": "string" }
  ]
}
```

Here we have a few fields orderNumber, email, firstName, and lastName that are used to send notifications to the user whenever an order is placed successfully.

The idea is to generate the Java classes automatically using this schema, so if there is a change in the schema file, then those changes will be automatically applied during the build time.

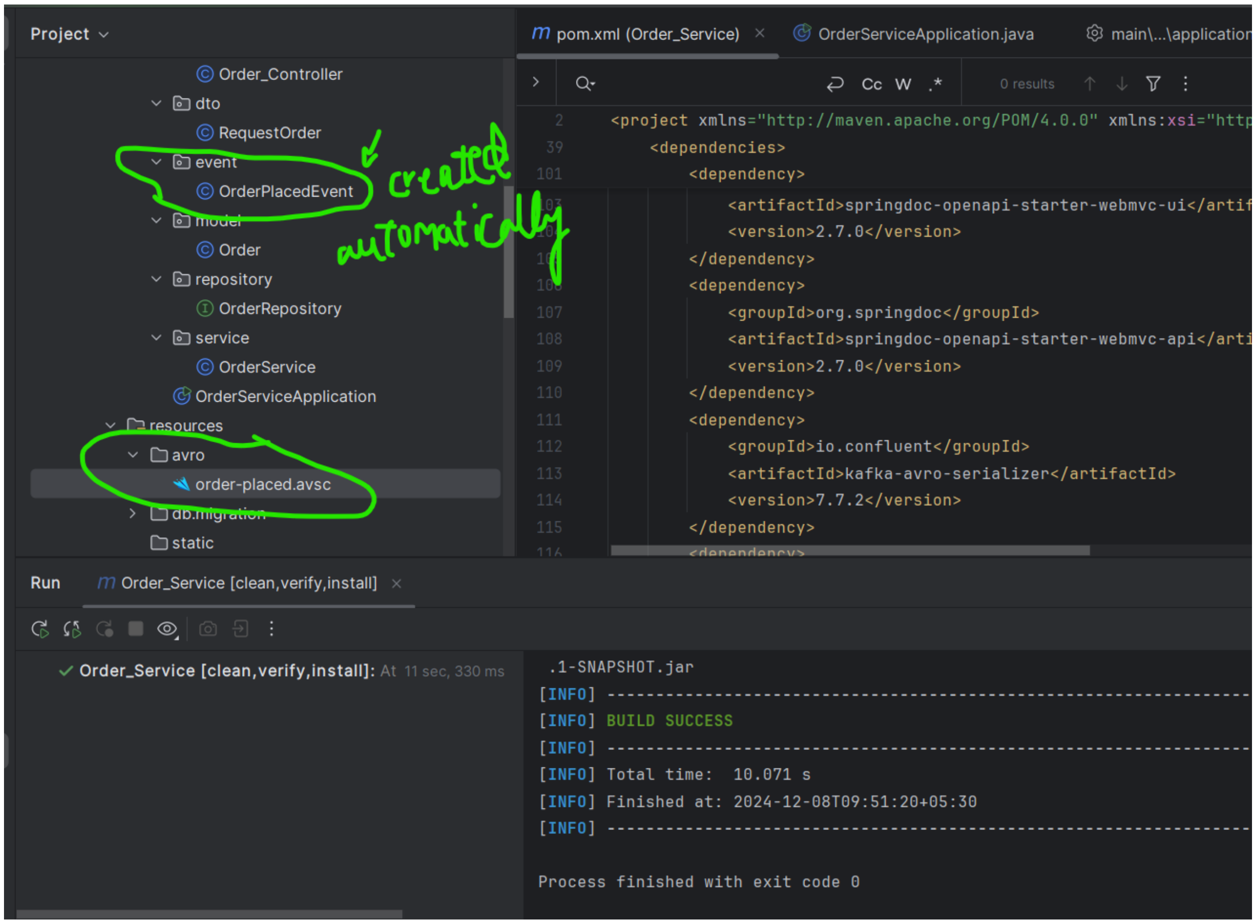
To be able to generate the Java classes automatically, we are going to use the **avro-maven-plugin**:

```
<plugin>
    <groupId>org.apache.avro</groupId>
    <artifactId>avro-maven-plugin</artifactId>
    <executions>
        <execution>
            <id>schemas</id>
            <phase>generate-sources</phase>
            <goals>
                <goal>schema</goal>
            </goals>
            <configuration>
```

```
<sourceDirectory>${project.basedir}/src/main/resources/avro</sourceDirectory>
                <outputDirectory>${project.basedir}/src/main/java/</outputDirectory>
            </configuration>
        </execution>
    </executions>
</plugin>
```

Add the above plugin under the `<plugins>` section and run **mvn clean compile** command.

Now you should see a file called **OrderPlacedEvent.java** under the **com.techie.microservices.order.event** package.



```
Order_Controller
dto
  RequestOrder
event
  OrderPlacedEvent
model
  Order
repository
  OrderRepository
service
  OrderService
OrderServiceApplication
resources
  avro
    order-placed.avsc
  db.migration
  static
```

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http
    <dependencies>
        <dependency>
            <artifactId>springdoc-openapi-starter-webmvc-ui</artif
            <version>2.7.0</version>
        </dependency>
        <dependency>
            <groupId>org.springdoc</groupId>
            <artifactId>springdoc-openapi-starter-webmvc-api</arti
            <version>2.7.0</version>
        </dependency>
        <dependency>
            <groupId>io.confluent</groupId>
            <artifactId>kafka-avro-serializer</artifactId>
            <version>7.7.2</version>
        </dependency>
        <dependency>
```

```
✓ Order_Service [clean,verify,install]: At 11 sec, 330 ms

.1-SNAPSHOT.jar
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  10.071 s
[INFO] Finished at: 2024-12-08T09:51:20+05:30
[INFO] ------------------------------------------------------------------------

Process finished with exit code 0
```

Producing Messages from Order Service

Now it's time to configure Kafka in our Spring Boot application, for that we are going to add the following properties in the application.properties file.

```
#Kafka Properties
spring.kafka.bootstrap-servers=localhost:9092
spring.kafka.template.default-topic=order-placed
spring.kafka.producer.key-
serializer=org.apache.kafka.common.serialization.StringSerializer
spring.kafka.producer.value-serializer=io.confluent.kafka.serializers.KafkaAvroSerializer
spring.kafka.producer.properties.schema.registry.url=http://localhost:8085
```

before we used Json serializer. but now we changed to Avro Serializer because in our project we are using Avro.

The above properties provide the necessary configuration to run Kafka Producer and to use the Kafka Schema registry in our Order service application.

Inside the OrderService.java class, let's add the logic to send the messages to the Kafka topic using the KafkaTemplate class.

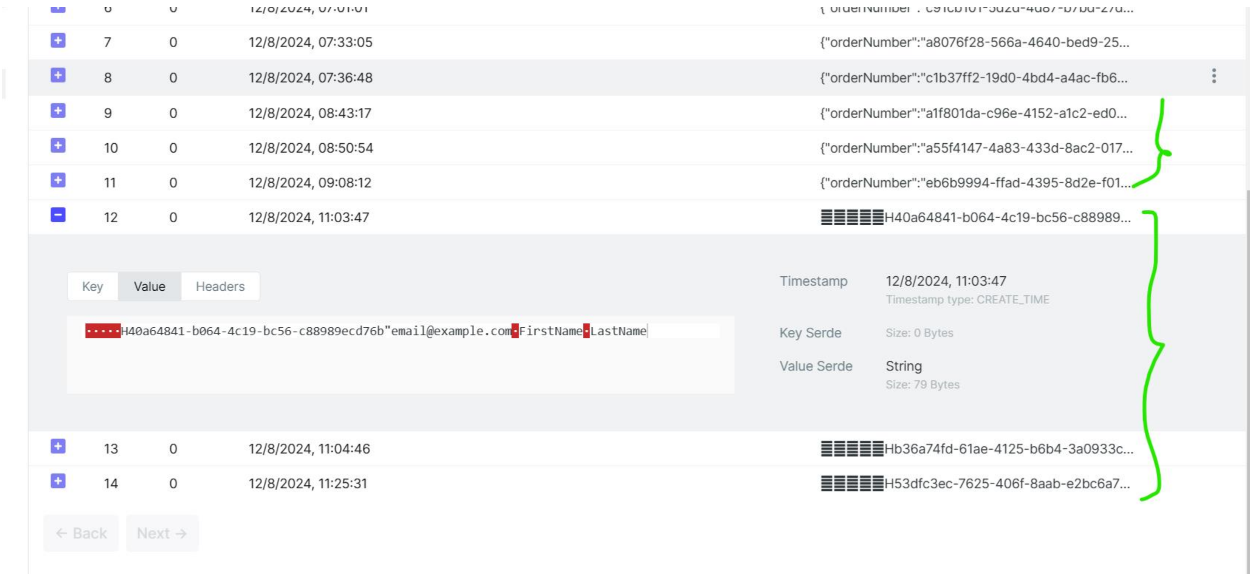
Here is how the OrderService.java class looks like:

OrderService.java

```
package com.lakshmiTech.microservices.order.service;
import com.lakshmiTech.microservices.order.client.InventoryClient;
import com.lakshmiTech.microservices.order.dto.RequestOrder;
import com.lakshmiTech.microservices.order.event.OrderPlacedEvent;
import com.lakshmiTech.microservices.order.model.Order;
import com.lakshmiTech.microservices.order.repository.OrderRepository;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.kafka.core.KafkaTemplate;
import org.springframework.stereotype.Service;

import java.util.UUID;

@Service
public class OrderService {

   private static final Logger log = LoggerFactory.getLogger(OrderService.class);
    private final OrderRepository orderRepository;

    private final InventoryClient inventoryClient;

    private final KafkaTemplate<String, OrderPlacedEvent> kafkaTemplate;

```

```java
    public OrderService(OrderRepository orderRepository, InventoryClient inventoryClient,
KafkaTemplate<String, OrderPlacedEvent> kafkaTemplate) {
        this.orderRepository = orderRepository;
        this.inventoryClient = inventoryClient;
        this.kafkaTemplate = kafkaTemplate;
    }



    public void placeOrder(RequestOrder requestOrder) {

        var isProductInStock = inventoryClient.isInStock(requestOrder.getSkuCode(),
requestOrder.getQuantity());

        if (isProductInStock) {
            Order order = new Order();
            order.setOrderNumber(UUID.randomUUID().toString());
            order.setPrice(requestOrder.getPrice());
            order.setSkuCode(requestOrder.getSkuCode());
            order.setQuantity(requestOrder.getQuantity());
            orderRepository.save(order);

            //send the message to kafka topic
            //ordernumber, email
            OrderPlacedEvent orderPlacedEvent = new OrderPlacedEvent( );
            orderPlacedEvent.setOrderNumber(order.getOrderNumber());
            orderPlacedEvent.setEmail(requestOrder.getUserDetails().email());
            orderPlacedEvent.setFirstName(requestOrder.getUserDetails().firstName());
            orderPlacedEvent.setLastName(requestOrder.getUserDetails().lastName());
            log.info("Start- Sending OrderPlacedEvent {} to Kafka Topic",
orderPlacedEvent);
            kafkaTemplate.send("order-placed", orderPlacedEvent);
            log.info("End- Sending OrderPlacedEvent {} to Kafka Topic",
orderPlacedEvent);
        }
        else{
            throw new RuntimeException("product with skuCode "+ requestOrder.getSkuCode()
+ " is not in stock");
        }
    }
}
```

go to google and run http://localhost:8086. and see messages under topics. we can see latest request which was in avro format. and remaining all previous messages in JSON format. as we added avro dependecies and configurations now we are receiving messages in avro format.



6	0	12/8/2024, 07:01:01	{"orderNumber":"c91cb101-5d2d-4d87-b7bd-27d...
7	0	12/8/2024, 07:33:05	{"orderNumber":"a8076f28-566a-4640-bed9-25...
8	0	12/8/2024, 07:36:48	{"orderNumber":"c1b37ff2-19d0-4bd4-a4ac-fb6...
9	0	12/8/2024, 08:43:17	{"orderNumber":"a1f801da-c96e-4152-a1c2-ed0...
10	0	12/8/2024, 08:50:54	{"orderNumber":"a55f4147-4a83-433d-8ac2-017...
11	0	12/8/2024, 09:08:12	{"orderNumber":"eb6b9994-ffad-4395-8d2e-f01...
12	0	12/8/2024, 11:03:47	▤▤▤▤▤H40a64841-b064-4c19-bc56-c88989...

```
•••••H40a64841-b064-4c19-bc56-c88989ecd76b"email@example.com•FirstName•LastName
```

Timestamp 12/8/2024, 11:03:47
Timestamp type: CREATE_TIME
Key Serde Size: 0 Bytes
Value Serde String
Size: 79 Bytes

13	0	12/8/2024, 11:04:46	▤▤▤▤▤Hb36a74fd-61ae-4125-b6b4-3a0933c...
14	0	12/8/2024, 11:25:31	▤▤▤▤▤H53dfc3ec-7625-406f-8aab-e2bc6a7...

This is all the logic we need to produce the events to order-placed kafka topic. Now let's see how to consume the messages in our consumer, that would be the **notification-service**.

Consuming Messages from Notification Service

Let's create a new Spring Boot application called Notification Service with the following dependencies.

- Spring Kafka
- Java Mail Sender
- Lombok
- Test Containers

After adding these dependencies, generate the project and open it in your IDE.

Now we need to add some more dependencies like Kafka Schema Registry, Avro Serializer, etc. (Copy from order service)

The complete pom.xml for notification-service looks like below:

pom.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>3.3.0</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
    <groupId>com.programming.techie</groupId>
    <artifactId>notification-service</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>notification-service</name>
    <description>notification-service</description>
    <properties>
        <java.version>21</java.version>
        <maven.compiler.source>21</maven.compiler.source>
        <maven.compiler.target>21</maven.compiler.target>
    </properties>
    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-mail</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.kafka</groupId>
            <artifactId>spring-kafka</artifactId>
        </dependency>
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <optional>true</optional>
        </dependency>
        <dependency>
            <groupId>io.confluent</groupId>
            <artifactId>kafka-avro-serializer</artifactId>
            <version>7.6.0</version>
        </dependency>
        <dependency>
            <groupId>io.confluent</groupId>
            <artifactId>kafka-schema-registry-client</artifactId>
            <version>7.6.0</version>
        </dependency>
        <dependency>
            <groupId>org.apache.avro</groupId>
            <artifactId>avro</artifactId>
            <version>1.11.3</version>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-testcontainers</artifactId>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.springframework.kafka</groupId>
            <artifactId>spring-kafka-test</artifactId>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.testcontainers</groupId>
            <artifactId>junit-jupiter</artifactId>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.testcontainers</groupId>
            <artifactId>kafka</artifactId>
            <scope>test</scope>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
                <configuration>
                    <excludes>
                        <exclude>
                            <groupId>org.projectlombok</groupId>
                            <artifactId>lombok</artifactId>
                        </exclude>
                    </excludes>
                </configuration>
            </plugin>
            <plugin>
                <groupId>org.apache.avro</groupId>
                <artifactId>avro-maven-plugin</artifactId>
                <executions>
                    <execution>
                        <id>schemas</id>
                        <phase>generate-sources</phase>
                        <goals>
                            <goal>schema</goal>
                        </goals>
                        <configuration>

<sourceDirectory>${project.basedir}/src/main/resources/avro</sourceDirectory>

<outputDirectory>${project.basedir}/src/main/java/</outputDirectory>
                        </configuration>
                    </execution>
                </executions>
            </plugin>
        </plugins>
    </build>


    <repositories>
        <repository>
            <id>confluent</id>
            <url>https://packages.confluent.io/maven/</url>
        </repository>
    </repositories>
</project>
```

Place the **order-placed.avsc** file under the **src/main/resources/avro** folder

order-placed.avsc

```
{
  "type": "record",
  "name": "OrderPlacedEvent",
  "namespace": "com.techie.microservices.order.event",
  "fields": [
    { "name": "orderNumber", "type": "string" },
    { "name": "email", "type": "string" },
    { "name": "firstName", "type": "string" },
    { "name": "lastName", "type": "string" }
  ]
}
```

Now let's configure the properties for Kafka Consumer in our Spring Boot Application's

application.properties file:

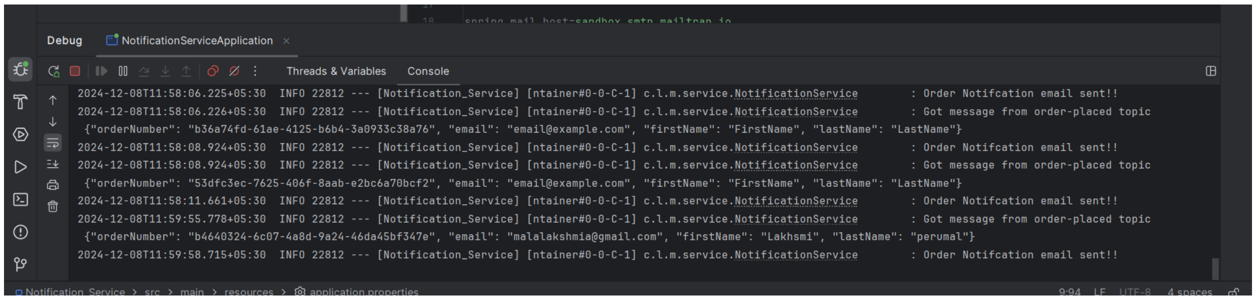
```
spring.application.name=notification-service
# Mail Properties
spring.mail.host=sandbox.smtp.mailtrap.io
spring.mail.port=2525
spring.mail.username=<username>
spring.mail.password=<password>
spring.mail.protocol=smtp
# Kafka Config
spring.kafka.bootstrap-servers=localhost:9092
spring.kafka.consumer.group-id=notificationService
spring.kafka.consumer.key-
deserializer=org.apache.kafka.common.serialization.StringDeserializer
spring.kafka.consumer.value-
deserializer=org.springframework.kafka.support.serializer.ErrorHandlingDeserializer
spring.kafka.consumer.properties.spring.deserializer.key.delegate.class=org.apache.kafka.
common.serialization.StringDeserializer
spring.kafka.consumer.properties.spring.deserializer.value.delegate.class=io.confluent.ka
fka.serializers.KafkaAvroDeserializer
spring.kafka.consumer.properties.schema.registry.url=http://localhost:8085
spring.kafka.consumer.properties.specific.avro.reader=true
```

Create a class called **NotificationService.java** that listens for the messages on the topic - "order-placed" and sends email

NotificationService.java

```java
package com.techie.microservices.notification.service;

import com.techie.microservices.order.event.OrderPlacedEvent;
import lombok.RequiredArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import org.springframework.kafka.annotation.KafkaListener;
import org.springframework.mail.MailException;
import org.springframework.mail.javamail.JavaMailSender;
import org.springframework.mail.javamail.MimeMessageHelper;
import org.springframework.mail.javamail.MimeMessagePreparator;
import org.springframework.stereotype.Service;

@Service
@RequiredArgsConstructor
@Slf4j
public class NotificationService {

    private final JavaMailSender javaMailSender;

    @KafkaListener(topics = "order-placed")
    public void listen(OrderPlacedEvent orderPlacedEvent){
        log.info("Got Message from order-placed topic {}", orderPlacedEvent);
        MimeMessagePreparator messagePreparator = mimeMessage -> {
            MimeMessageHelper messageHelper = new MimeMessageHelper(mimeMessage);
            messageHelper.setFrom("springshop@email.com");
            messageHelper.setTo(orderPlacedEvent.getEmail().toString());
            messageHelper.setSubject(String.format("Your Order with OrderNumber %s is
placed successfully", orderPlacedEvent.getOrderNumber()));
            messageHelper.setText(String.format("""
                            Hi %s,%s

                            Your order with order number %s is now placed successfully.

                            Best Regards
                            Spring Shop
                            """,
                    orderPlacedEvent.getFirstName().toString(),
                    orderPlacedEvent.getLastName().toString(),
                    orderPlacedEvent.getOrderNumber()));
        };
        try {
            javaMailSender.send(messagePreparator);
            log.info("Order Notifcation email sent!!");
        } catch (MailException e) {
            log.error("Exception occurred when sending mail", e);
            throw new RuntimeException("Exception occurred when sending mail to
springshop@email.com", e);
        }
    }
}
```

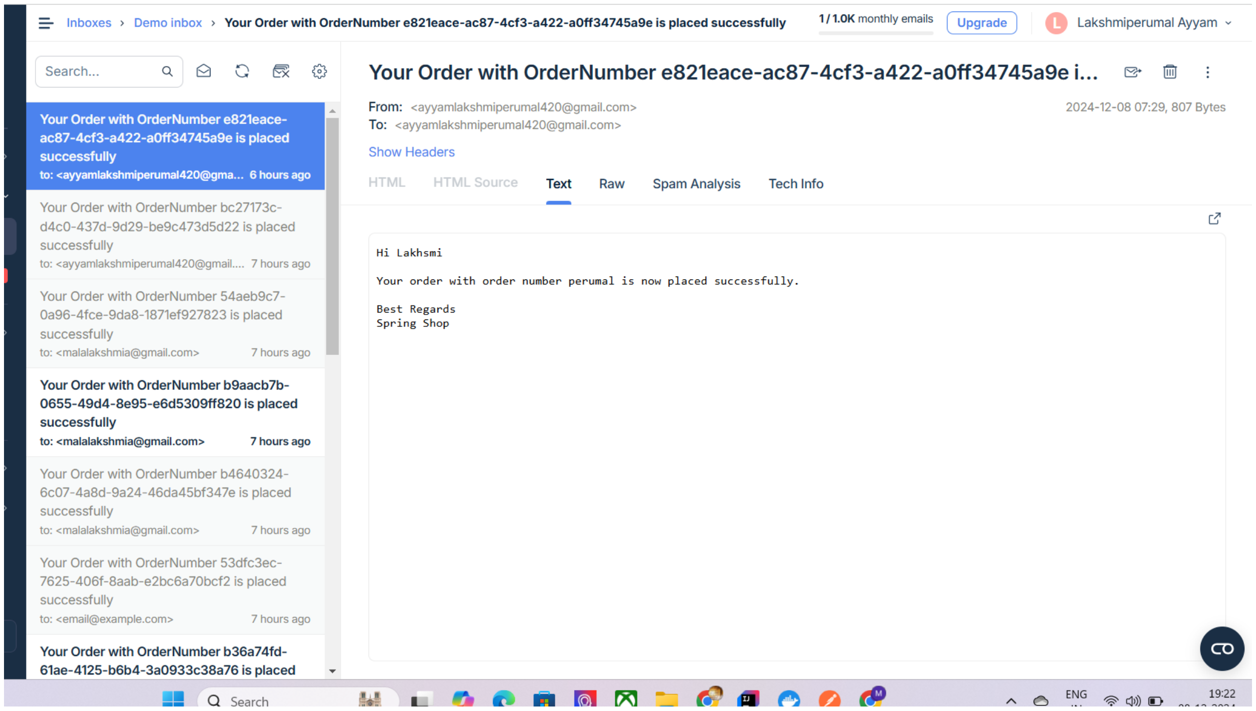
output:



```
2024-12-08T11:58:06.225+05:30  INFO 22812 --- [Notification_Service] [ntainer#0-0-C-1] c.l.m.service.NotificationService        : Order Notifcation email sent!!
2024-12-08T11:58:06.226+05:30  INFO 22812 --- [Notification_Service] [ntainer#0-0-C-1] c.l.m.service.NotificationService        : Got message from order-placed topic
 {"orderNumber": "b36a74fd-61ae-4125-b6b4-3a0933c38a76", "email": "email@example.com", "firstName": "FirstName", "lastName": "LastName"}
2024-12-08T11:58:08.924+05:30  INFO 22812 --- [Notification_Service] [ntainer#0-0-C-1] c.l.m.service.NotificationService        : Order Notifcation email sent!!
2024-12-08T11:58:08.924+05:30  INFO 22812 --- [Notification_Service] [ntainer#0-0-C-1] c.l.m.service.NotificationService        : Got message from order-placed topic
 {"orderNumber": "53dfc3ec-7625-406f-8aab-e2bc6a70bcf2", "email": "email@example.com", "firstName": "FirstName", "lastName": "LastName"}
2024-12-08T11:58:11.661+05:30  INFO 22812 --- [Notification_Service] [ntainer#0-0-C-1] c.l.m.service.NotificationService        : Order Notifcation email sent!!
2024-12-08T11:59:55.778+05:30  INFO 22812 --- [Notification_Service] [ntainer#0-0-C-1] c.l.m.service.NotificationService        : Got message from order-placed topic
 {"orderNumber": "b4640324-6c07-4a8d-9a24-46da45bf347e", "email": "malalakshmia@gmail.com", "firstName": "Lakhsmi", "lastName": "perumal"}
2024-12-08T11:59:58.715+05:30  INFO 22812 --- [Notification_Service] [ntainer#0-0-C-1] c.l.m.service.NotificationService        : Order Notifcation email sent!!
```

Debug
NotificationServiceApplication
Threads & Variables
Console
Notification_Service > src > main > resources > application.properties
9:94 LF UTF-8 4 spaces

and check in mailtrap for mails sent information.



Your Order with OrderNumber e821eace-ac87-4cf3-a422-a0ff34745a9e is placed successfully
to: <ayyamlakshmiperumal420@gma... 6 hours ago

Your Order with OrderNumber bc27173c-d4c0-437d-9d29-be9c473d5d22 is placed successfully
to: <ayyamlakshmiperumal420@gmail.... 7 hours ago

Your Order with OrderNumber 54aeb9c7-0a96-4fce-9da8-1871ef927823 is placed successfully
to: <malalakshmia@gmail.com> 7 hours ago

Your Order with OrderNumber b9aacb7b-0655-49d4-8e95-e6d5309ff820 is placed successfully
to: <malalakshmia@gmail.com> 7 hours ago

Your Order with OrderNumber b4640324-6c07-4a8d-9a24-46da45bf347e is placed successfully
to: <malalakshmia@gmail.com> 7 hours ago

Your Order with OrderNumber 53dfc3ec-7625-406f-8aab-e2bc6a70bcf2 is placed successfully
to: <email@example.com> 7 hours ago

Your Order with OrderNumber b36a74fd-61ae-4125-b6b4-3a0933c38a76 is placed

Your Order with OrderNumber e821eace-ac87-4cf3-a422-a0ff34745a9e i...

From: <ayyamlakshmiperumal420@gmail.com>
To: <ayyamlakshmiperumal420@gmail.com>

2024-12-08 07:29, 807 Bytes

```
Hi Lakhsmi

Your order with order number perumal is now placed successfully.

Best Regards
Spring Shop
```