Kafka Listener Microservice Setup Guide

Step 01 : add kafka.propertiest file

```
##############################################################
#####################################################
#######  Kafka properties
##############################################################
##########################
##############################################################
#####################################################

spring.kafka.consumer.bootstrap-servers=localhost:9092
spring.kafka.consumer.group-id=group-1
#spring.kafka.consumer.auto-offset-reset=earliest
#spring.kafka.consumer.key-deserializer=org.apache.kafka.comm
on.serialization.StringDeserializer
#spring.kafka.consumer.value-deserializer=org.apache.kafka.co
mmon.serialization.StringDeserializer

spring.kafka.consumer.key-deserializer=
org.apache.kafka.common.serialization.StringDeserializer
#spring.kafka.consumer.value-deserializer=
org.springframework.kafka.support.serializer.StringDeserializ
er
spring.kafka.consumer.value-deserializer=
org.springframework.kafka.support.serializer.JsonDeserializer
spring.kafka.consumer.properties.spring.json.trusted.packages
=*

kafka.mail.topic = notification-mail-events
```

Step 02 : add parseFromJsonString method in DTO (EmailDetailsDto) class

```java
import com.fasterxml.jackson.core.JsonProcessingException;
import com.fasterxml.jackson.databind.ObjectMapper;
import jakarta.validation.constraints.Email;
import jakarta.validation.constraints.NotNull;
import jakarta.validation.constraints.Size;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@AllArgsConstructor
@NoArgsConstructor
public class EmailDetailsDto {
   @NotNull(message = "recipient cannot be null")
   @Email(message = "must be a valid email")
   @Size(max = 50)
   private String recipient;
   private String msgBody;
   private String subject;
   private String paymentId;

   /**
    * Parses a JSON-formatted string into an   {@code EmailDetailsDto} object.
    *
    * This method uses the Jackson ObjectMapper to deserialize a JSON string
    * representing an {@code EmailDetailsDto} object. If the deserialization
is
    * successful, an instance of {@code EmailDetailsDto} is returned.
    *
    * @param jsonString The JSON-formatted string to be parsed into an object.
    * @return An {@code EmailDetailsDto} object if the parsing is successful;
otherwise, returns null.
    * @throws IllegalArgumentException If the provided JSON string is null or
empty.
    * @throws RuntimeException If an error occurs during the deserialization
process,
    *                              such as an invalid JSON format or incompatible
data types.
    */
   public static EmailDetailsDto parseFromJsonString(String jsonString) {
       try {
           ObjectMapper objectMapper = new ObjectMapper();
           return objectMapper.readValue(jsonString, EmailDetailsDto.class);
       } catch (JsonProcessingException e) {
           // Handle the exception (e.g., log it or throw a custom exception)
           e.printStackTrace();
           return null;
       }
   }
}
```

Step 03 : add KafkaMailConfig.java file

```java
import com.tnpay.notificationmicroservice.dto.EmailDetailsDto;
import com.tnpay.notificationmicroservice.util.MailSenderUtil;
import org.springframework.context.annotation.Configuration;
import org.springframework.kafka.annotation.KafkaListener;
import org.springframework.mail.MailException;

@Configuration
public class KafkaMailConfig {

   public static final String MAIL_TOPIC = "notification-mail-events";
   public static final String GROUP_ID = "group-1";
   private final MailSenderUtil mailSenderUtil;

   public KafkaMailConfig(MailSenderUtil mailSenderUtil) {
       this.mailSenderUtil = mailSenderUtil;
   }

   /**
     * Listens for incoming messages from the Kafka topic specified by MAIL_TOPIC
and processes them.
     * <p>
     * This method acts as a Kafka message listener, receiving messages from the
specified topic and
     * processing them to send emails using the configured mail sender utility.
     *
     * @param message The incoming Kafka message containing JSON-formatted data
representing an email.
     * @throws RuntimeException If an error occurs during message processing, such
as parsing errors,
     *                          mail sending failures, or other runtime exceptions.
     */
   @KafkaListener(topics = MAIL_TOPIC, groupId = GROUP_ID)
   public void sendMail(String message) {
       System.out.println("kafka message: " + message);
       EmailDetailsDto parsedEmailDetailsDto =
EmailDetailsDto.parseFromJsonString(message);
       // Check if the parsing was successful
       if (parsedEmailDetailsDto != null) {
           System.out.println("Parsed EmailDetailsDto: " + parsedEmailDetailsDto);
           try {
               mailSenderUtil.mailSending(parsedEmailDetailsDto);
           } catch (MailException e) {
               throw new RuntimeException("Mail sending Exception: " +
e.getMessage());
           }
       } else {
           System.out.println("Failed to parse JSON string.");
       }
   }
}
```

Kafka Producer Microservice Setup Guide

Step 01 : add kafka Producer properties

```
#Producer configuration
spring.kafka.producer.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
```

Step 02 : add KafkaConfig.java classfile

```java
import org.apache.kafka.clients.admin.NewTopic;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.kafka.config.TopicBuilder;

@Configuration
public class KafkaConfig {

   @Bean
   public NewTopic topic() {
       return TopicBuilder
               .name(AppConstants.MAIL_TOPIC)
               //  .name(AppConstants.ORDER_TOPIC_NAME)
               //.partitions(3)
               //  .replicas(3)
               .build();
   }
}
```

Step 03 : add “parseToJsonString” -method to DTO class

```
public String parseToJsonString() {
    try {
        ObjectMapper objectMapper = new ObjectMapper();
        return objectMapper.writeValueAsString(this);
    } catch (JsonProcessingException e) {
        // Handle the exception (e.g., log it or throw a
custom exception)
        e.printStackTrace();
        return null;
    }
}
```

Step 04 : add KafkaService.java classfile

/**
* Updates a notification by sending the provided {@code EmailDetailsDto} as a message to a Kafka topic.
*
* This method sends the serialized JSON representation of the {@code EmailDetailsDto} object to a Kafka topic
* named {@link AppConstants#MAIL_TOPIC}. The Kafka topic is intended for processing and handling email notifications.
*
* @param mail The {@code EmailDetailsDto} object representing the notification to be updated.
* @return {@code true} if the notification update message is successfully sent to Kafka; otherwise, {@code false}.

```
* @throws IllegalArgumentException If the provided {@code EmailDetailsDto} object is
null.
* @throws RuntimeException If an error occurs during the serialization of the {@code
EmailDetailsDto} object or
*                          while sending the message to Kafka, resulting in a runtime
exception.
*/
```

```java
public boolean updateNotification(EmailDetailsDto mail) {
   if (mail == null) {
       throw new IllegalArgumentException("EmailDetailsDto cannot be
null.");
   }

   try {
       // Send the serialized JSON representation of the EmailDetailsDto
to Kafka
       kafkaTemplate.send(AppConstants.MAIL_TOPIC,
               mail.parseToJsonString());
       return true;
   } catch (Exception e) {
       // Handle exceptions, log the error, and propagate a runtime
exception
       throw new RuntimeException("Error updating notification: " +
e.getMessage(), e);
   }
}
```

Kafka Cluster Setup Guide:

Kafka with KRaft

Generate a Cluster UUID

```
$ KAFKA_CLUSTER_ID="$(bin/kafka-storage.sh random-uuid)"
```

Format Log Directories

```
$ bin/kafka-storage.sh format -t $KAFKA_CLUSTER_ID -c
config/kraft/server.properties
```

Start the Kafka Server

```
$ bin/kafka-server-start.sh config/kraft/server.properties
```

Once the Kafka server has successfully launched, you will have a basic Kafka environment running and ready to use.

TERMINATE THE KAFKA ENVIRONMENT

Now that you reached the end of the quickstart, feel free to tear down the Kafka environment—or continue playing around.

1. Stop the producer and consumer clients with `Ctrl-C`, if you haven't done so already.
2. Stop the Kafka broker with `Ctrl-C`.
3. Lastly, if the Kafka with ZooKeeper section was followed, stop the ZooKeeper server with `Ctrl-C`.

If you also want to delete any data of your local Kafka environment including any events you have created along the way, run the command:

```
$ rm -rf /tmp/kafka-logs /tmp/zookeeper /tmp/kraft-combined-logs
```