Standalone Guide: Implementing a Kafka Email Alert Producer

This guide provides step-by-step instructions to implement a standalone service that produces `EMAIL_ALERT` messages to be consumed by the Integration Microservice.

1. Project Setup

Ensure your project relies on **Spring Boot** and **Spring Kafka**.

Dependencies (`pom.xml`)

Include the Spring Kafka and Jackson dependencies:

```
<dependency>
    <groupId>org.springframework.kafka</groupId>
    <artifactId>spring-kafka</artifactId>
</dependency>
<dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-databind</artifactId>
</dependency>
```

Configuration (`application.properties`)

Configure the Kafka bootstrap servers and the target topic.

```
spring.kafka.bootstrap-servers=5odds2wv3hpq.streaming.ap-mumbai-1.oci.oraclecloud.com:9092
spring.kafka.properties.security.protocol=SASL_SSL
spring.kafka.properties.sasl.mechanism=PLAIN
spring.kafka.properties.sasl.jaas.config=org.apache.kafka.common.security.plain.PlainLogin
Module required username="fciannadarpan/fci-identity-
domain/oci@coforge.com/ocid1.streampool.oc1.ap-mumbai-
1.amaaaaaajs3e7viajqixmbyxufme34nhiwmleix4hf2xznaa5odds2wv3hpq"
password="_2IL>LqIO47d5_<aF9AH";
spring.kafka.producer.properties.enable.idempotence=false
topics.alerts=alerts-topic
```

2. Producer Implementation

Kafka Configuration Class

Define a producer factory to ensure all `spring.kafka` properties are loaded correctly.

```
@Configuration
public class KafkaConfig {
    @Bean
    public ProducerFactory<String, String> producerFactory(KafkaProperties
kafkaProperties) {
        Map<String, Object> configProps = kafkaProperties.buildProducerProperties();
        configProps.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG,
StringSerializer.class);
        configProps.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG,
StringSerializer.class);
        return new DefaultKafkaProducerFactory<>(configProps);
    }

    @Bean
    public KafkaTemplate<String, String> kafkaTemplate(ProducerFactory<String, String>
producerFactory) {
        return new KafkaTemplate<>(producerFactory);
    }
}
```

Email Alert Service

The service formats and sends the alert as a JSON string containing a map of properties.

```
@Service
public class EmailAlertProducer {
    private final KafkaTemplate<String, String> kafkaTemplate;
    private final ObjectMapper objectMapper;

    @Value("${topics.alerts:alerts-topic}")
    private String alertsTopic;

    private static final String EVENT_NAME_KEY = "eventName";
    private static final String EMAIL_ALERT_KEY = "EMAIL_ALERT";

    public void sendEmailAlert(String eventName, Map<String, Object> properties) {
        try {
            if (!properties.containsKey(EVENT_NAME_KEY)) {
                properties.put(EVENT_NAME_KEY, eventName);
            }
            String message = objectMapper.writeValueAsString(properties);
            kafkaTemplate.send(alertsTopic, EMAIL_ALERT_KEY, message);
        } catch (JsonProcessingException e) {
```

```
            throw new RuntimeException("Error serializing email properties", e);
        }
    }
}
```

---

3. Integration Setup (Consumer Side)

The consumer must have a corresponding template defined in `EmailAlertTemplates.xml`.

XML Template Definition

```
<event name="YourNewEventName">
    <to>%emailid%</to>
    <priority>HIGH</priority>
    <status>ON</status>
    <emailSubject><![CDATA[ Alert: %custom_subject_prop% ]]></emailSubject>
    <emailBody><![CDATA[
        Hello, <br/><br/>
        Notification for: <strong>%custom_body_prop%</strong>.
    ]]></emailBody>
</event>
```

> **Placeholders Note:**
>
> - **Subject & Body:** Use `%key%` syntax; the consumer replaces this with the value from the properties map.
> - **Dynamic To Tag:** Pass the `emailid` key in your map to populate the recipient field.

4. Usage Example

```
Map<String, Object> props = new HashMap<>();
props.put("emailid", "dynamic.user@example.com");
props.put("custom_subject_prop", "System Failure");
props.put("custom_body_prop", "Database Connection Lost");

emailProducer.sendEmailAlert("YourNewEventName", props);
```