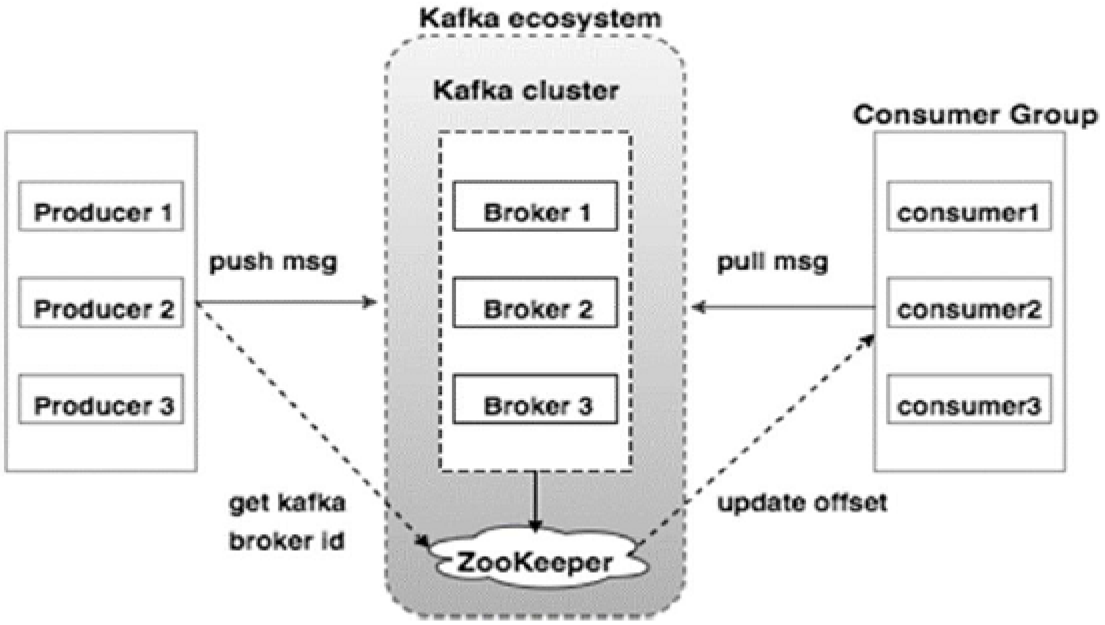
Microservices & Kafka Lab Report

Amo Samuel

Introduction

In this report, I would like to provide a detailed explanation of Kafka's architecture and message flow, as well as how I configured Kafka for a student management use case. I've also included an image to help illustrate the architecture and flow of messages in Kafka.



Source: Annon

Kafka Architecture Breakdown

In the image above, I analyzed the core components of Kafka, which include producers, brokers, consumers, and ZooKeeper (**though Kafka's newer KRaft mode can operate without ZooKeeper**). Here's how I understand each of the components:

1. **Producers**: These are responsible for sending messages to Kafka topics. In my example, I've implemented a producer that sends student data as a message. Each producer is independent and communicates directly with the brokers.
2. **Kafka Cluster**: This is made up of multiple brokers (in this case, Broker 1, Broker 2, and Broker 3). Kafka's distributed architecture ensures that messages are stored across

different partitions for scalability and redundancy. The partitioning mechanism guarantees that no single broker gets overwhelmed by the volume of messages.

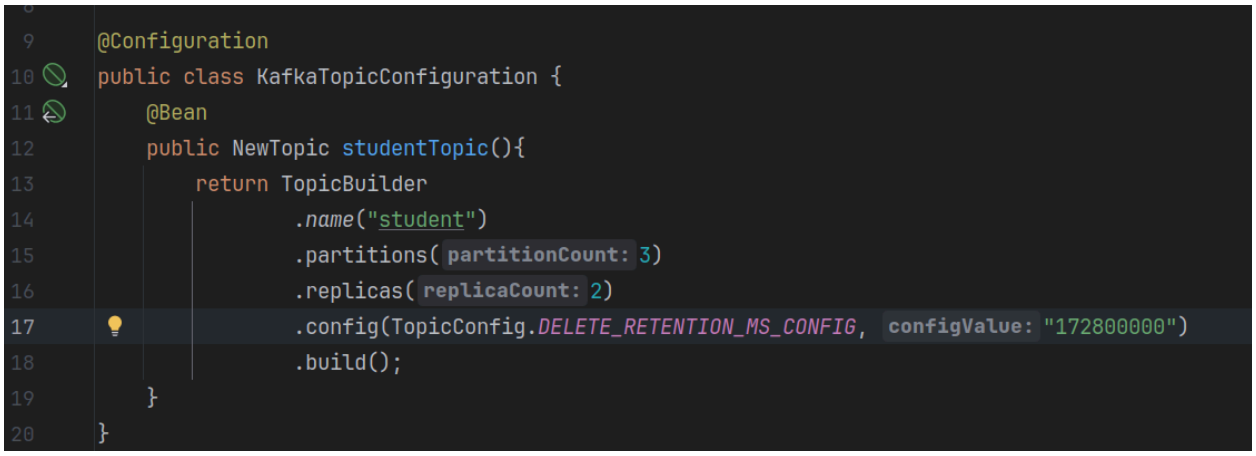
3. **ZooKeeper**: Though I know Kafka can run without ZooKeeper in KRaft mode, the image represents a traditional Kafka setup where ZooKeeper is responsible for maintaining metadata and broker status. ZooKeeper helps coordinate broker failovers and stores partition information.
4. **Consumers**: In the diagram, there's a **consumer group**, which is how Kafka ensures high availability and fault tolerance in consuming messages. Each consumer in the group processes messages from different partitions, which guarantees that message consumption is balanced.

Kafka Configuration

Now, I'll explain how I configured Kafka for the student management system. The goal was to configure the system to produce and consume student data efficiently.

Topic Configuration

Here's the code I used to configure the Kafka topic for student data:

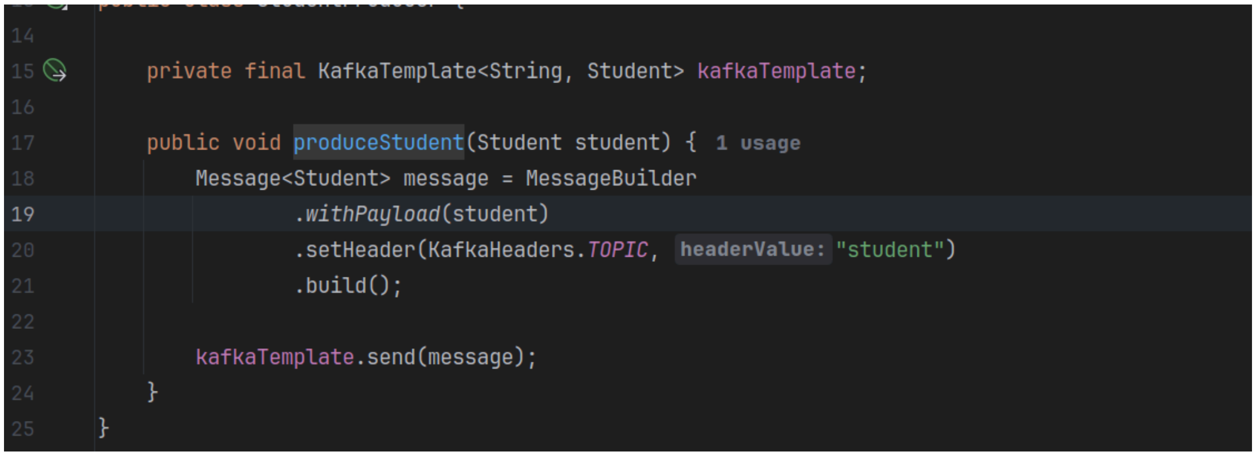


```java
@Configuration
public class KafkaTopicConfiguration {
    @Bean
    public NewTopic studentTopic(){
        return TopicBuilder
                .name("student")
                .partitions( partitionCount: 3)
                .replicas( replicaCount: 2)
                .config(TopicConfig.DELETE_RETENTION_MS_CONFIG,  configValue: "172800000")
                .build();
    }
}
```

In this configuration, I set up a topic called student with **3 partitions** and **2 replicas**. This ensures that the topic can handle large volumes of data while maintaining redundancy. The retention period for the messages is set to two days.

Student Producer Implementation

Here’s the producer code for sending student data to the student topic:

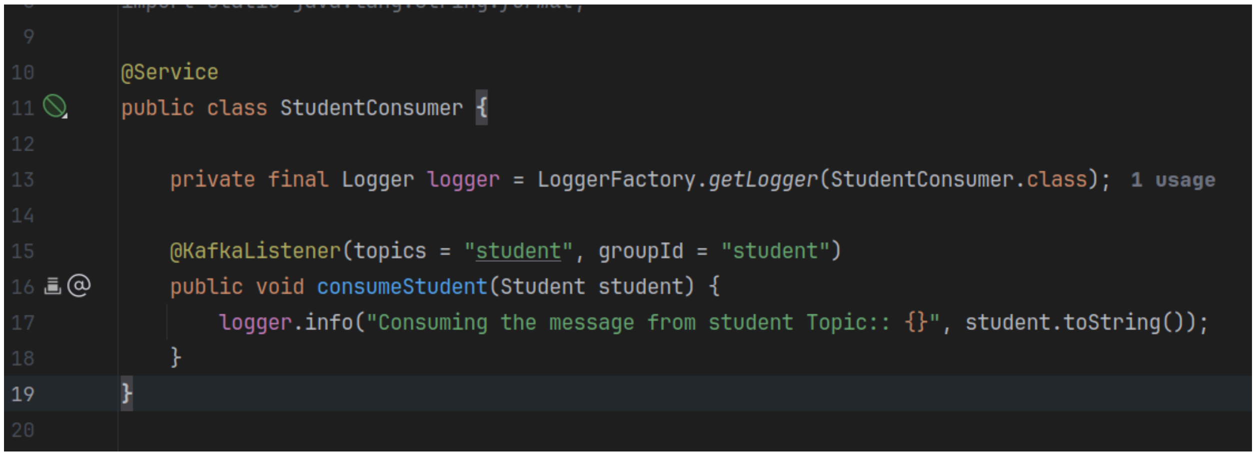


```
    private final KafkaTemplate<String, Student> kafkaTemplate;

    public void produceStudent(Student student) {  1 usage
        Message<Student> message = MessageBuilder
                .withPayload(student)
                .setHeader(KafkaHeaders.TOPIC,  headerValue: "student")
                .build();

        kafkaTemplate.send(message);
    }
}
```

The producer takes a Student object, wraps it in a message, and sends it to the student topic. This is a simple implementation that fits the basic requirements of the task, while also ensuring that the producer is decoupled from the message details (handled by KafkaTemplate).

Student Consumer Implementation

For the consumer, I implemented a service that listens to the student topic and processes the student data:

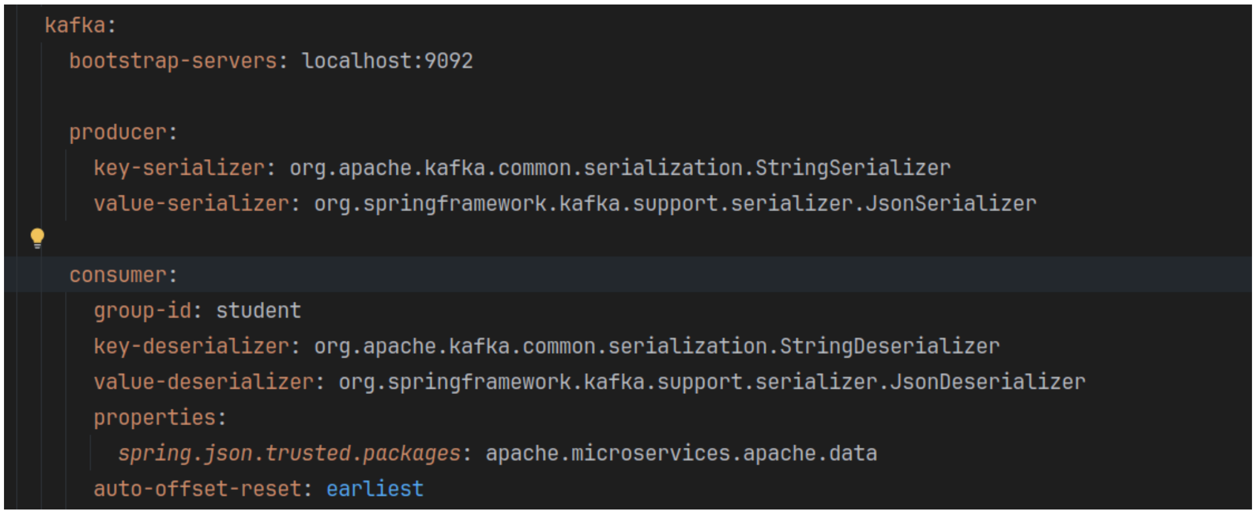


```

@Service
public class StudentConsumer {

    private final Logger logger = LoggerFactory.getLogger(StudentConsumer.class);  1 usage

    @KafkaListener(topics = "student", groupId = "student")
    public void consumeStudent(Student student) {
        logger.info("Consuming the message from student Topic:: {}", student.toString());
    }
}

```

This consumer is part of the student consumer group and logs each student message it consumes. This allowed me to monitor that messages were being correctly sent from the producer and processed by the consumer.

Kafka Configuration in Spring Boot

Below is the Spring Boot configuration I used to set up Kafka's producer and consumer:



```yaml
kafka:
  bootstrap-servers: localhost:9092

  producer:
    key-serializer: org.apache.kafka.common.serialization.StringSerializer
    value-serializer: org.springframework.kafka.support.serializer.JsonSerializer

  consumer:
    group-id: student
    key-deserializer: org.apache.kafka.common.serialization.StringDeserializer
    value-deserializer: org.springframework.kafka.support.serializer.JsonDeserializer
    properties:
      spring.json.trusted.packages: apache.microservices.apache.data
    auto-offset-reset: earliest
```

This configuration allows Kafka to run on localhost:9092 and ensures that messages are serialized/deserialized as JSON. The consumer group student is specified here, ensuring that messages are distributed properly across consumers.

Conclusion

In conclusion, I successfully configured Kafka to handle a simple student management system where producers send student data to a Kafka topic, and consumers retrieve and process that data. Through this exercise, I gained a deeper understanding of Kafka's architecture, message flow, and the nuances of configuring Kafka topics, producers, and consumers.