Event-Driven Microservices using Spring Boot and Kafka

author: Ramesh Fadatare

APACHE KAFKA MICROSERVICES SPRING BOOT

← Back to Spring Boot Tutorial

In this tutorial, you will learn how to build a simple Event-Driven Microservices application using Spring Boot and Apache Kafka.

Check out my top Udemy course: **Building Microservices with Spring Boot and Spring Cloud**.

Basically, you will learn how to use Apache Kafka as a message broker for Asynchronous communication between multiple microservices.

YouTube Video

Let's begin with what is Event-Driven architecture.

What Is Event-Driven Architecture?

Event-driven architecture (EDA) is a software design pattern in which decoupled applications can asynchronously publish and subscribe to events via an event broker/message broker.

In an Event-Driven Architecture, applications communicate with each other by sending and/or receiving events or messages.

Event-driven architecture is often referred to as "asynchronous" communication. This means that the sender and recipient don't have to wait for each other to move on to their next task. Systems are not dependent on that one message. For instance, a telephone call is considered synchronous or more along the lines of the traditional "request/response" architecture. Someone calls you and asks you to do something, the requestor waits while the responder completes the task, and then both parties hang up. An asynchronous example would be text messaging. You send a message and in some cases, you don't even know who you are sending it to or if anyone's listening, but you are not waiting for a response.
Event-driven apps can be created in any programming language because event-driven is a programming approach, not a language. An event-driven architecture is loosely coupled.

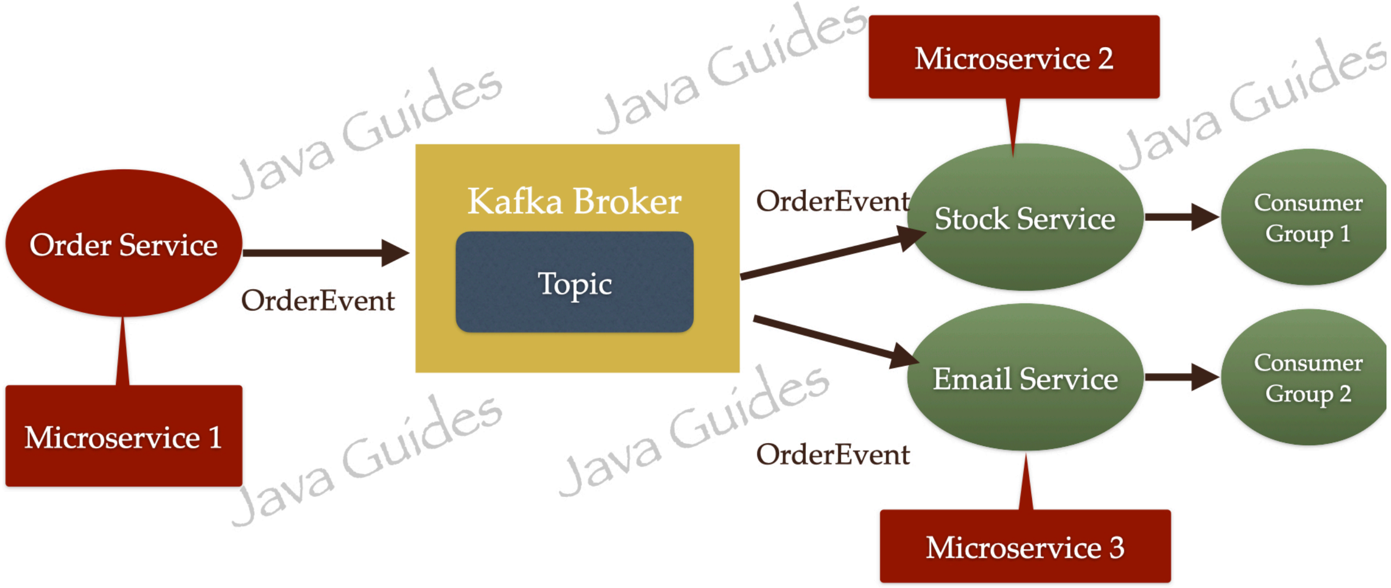
> Learn more about Event-Driven architecture at **What Is Event-Driven Architecture?**

Event-Driven Microservices Architecture Project Overview

Event-Driven Architecture is widely used in Microservices applications.

The below diagram shows a simple Event-Driven Microservices Architecture that we are going to implement in this tutorial:

Spring Boot Kafka Event-Driven Microservices Architecture with Multiple Consumers



In the above architecture, OrderService, StockService, and EmailService microservices are independent of each other. OrderService is a Producer application that sends an event to the Message Broker. StockService and EmailService are Consumers who will consume the events from the Message Broker.

In this tutorial, we will also see how multiple consumers will subscribe to a single Kafka topic to consume the events/messages.

Install and Setup Apache Kafka

1. Download Kafka from the official website at **https://kafka.apache.org/downloads**

2. Extract Kafka zip in the local file system

Run the following commands in order to start all services in the correct order:

3. Start Zookeeper service.

Use the below command to start the Zookeeper service:

```
# Start the ZooKeeper service
# Note: Soon, ZooKeeper will no longer be required by Apache Kafka.
$ bin/zookeeper-server-start.sh config/zookeeper.properties
```

4. Start Kafka Broker

Open another terminal session and run the below command to start the Kafka broker:

```
# Start the Kafka broker service
$ bin/kafka-server-start.sh config/server.properties
```

Once all services have successfully launched, you will have a basic Kafka environment running and ready to use.

Create 4 Microservices - OrderService, StockService, EmailService, and Base-Domains

Create OrderService Microservice

Let's create a Spring boot project using **https://start.spring.io/**

Use the below details while creating an `order-service` project:

```
        <groupId>net.javaguides</groupId>
        <artifactId>order-service</artifactId>
        <version>0.0.1-SNAPSHOT</version>
        <name>order-service</name>
        <description>Demo project for Spring Boot Order Service</description>
        <properties>
                <java.version>17</java.version>
        </properties>
```

Add Spring Web and Spring for Kafka dependencies.

Here is the complete pom.xml file for your reference:

```
<?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>3.0.0-M3</version>
                <relativePath/> <!-- lookup parent from repository -->
        </parent>
        <groupId>net.javaguides</groupId>
        <artifactId>order-service</artifactId>
        <version>0.0.1-SNAPSHOT</version>
        <name>order-service</name>
        <description>Demo project for Spring Boot Order Service</description>
        <properties>
                <java.version>17</java.version>
        </properties>
        <dependencies>
                <dependency>
                        <groupId>org.springframework.boot</groupId>
                        <artifactId>spring-boot-starter-web</artifactId>
                </dependency>
                <dependency>
                        <groupId>org.springframework.kafka</groupId>
                        <artifactId>spring-kafka</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>
        </dependencies>

        <build>
                <plugins>
                        <plugin>
                                <groupId>org.springframework.boot</groupId>
                                <artifactId>spring-boot-maven-plugin</artifactId>
                        </plugin>
                </plugins>
        </build>
        <repositories>
                <repository>
                        <id>spring-milestones</id>
                        <name>Spring Milestones</name>
                        <url>https://repo.spring.io/milestone</url>
                        <snapshots>
                                <enabled>false</enabled>
                        </snapshots>
                </repository>
        </repositories>
        <pluginRepositories>
                <pluginRepository>
                        <id>spring-milestones</id>
                        <name>Spring Milestones</name>
                        <url>https://repo.spring.io/milestone</url>
                        <snapshots>
                                <enabled>false</enabled>
                        </snapshots>
                </pluginRepository>
        </pluginRepositories>

</project>
```

Create StockService Microservice

Let's create a Spring boot project using **https://start.spring.io/**

Use the below details while creating a `stock-service` project:

```
        <groupId>net.javaguides</groupId>
        <artifactId>stock-service</artifactId>
        <version>0.0.1-SNAPSHOT</version>
        <name>stock-service</name>
        <description>Demo project for Spring Boot Stock Service</description>
        <properties>
                <java.version>17</java.version>
        </properties>
```

Add Spring Web and Spring for Kafka dependencies.

Here is the complete pom.xml file for your reference:

```
<?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>3.0.0-M3</version>
                <relativePath/> <!-- lookup parent from repository -->
        </parent>
        <groupId>net.javaguides</groupId>
        <artifactId>stock-service</artifactId>
        <version>0.0.1-SNAPSHOT</version>
        <name>stock-service</name>
        <description>Demo project for Spring Boot Stock Service</description>
        <properties>
                <java.version>17</java.version>
        </properties>
        <dependencies>
                <dependency>
```

```
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-web</artifactId>
		</dependency>
		<dependency>
			<groupId>org.springframework.kafka</groupId>
			<artifactId>spring-kafka</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>
	</dependencies>

	<build>
		<plugins>
			<plugin>
				<groupId>org.springframework.boot</groupId>
				<artifactId>spring-boot-maven-plugin</artifactId>
			</plugin>
		</plugins>
	</build>
	<repositories>
		<repository>
			<id>spring-milestones</id>
			<name>Spring Milestones</name>
			<url>https://repo.spring.io/milestone</url>
			<snapshots>
				<enabled>false</enabled>
			</snapshots>
		</repository>
	</repositories>
	<pluginRepositories>
		<pluginRepository>
			<id>spring-milestones</id>
			<name>Spring Milestones</name>
			<url>https://repo.spring.io/milestone</url>
			<snapshots>
				<enabled>false</enabled>
			</snapshots>
		</pluginRepository>
	</pluginRepositories>

</project>
```

Create EmailService Microservice

Let's create a Spring boot project using **https://start.spring.io/**

Use the below details while creating an `email-service` project:

```
	<groupId>net.javaguides</groupId>
	<artifactId>email-service</artifactId>
	<version>0.0.1-SNAPSHOT</version>
	<name>email-service</name>
	<description>Demo project for Spring Boot Email Service</description>
	<properties>
		<java.version>17</java.version>
	</properties>
```

Add Spring Web and Spring for Kafka dependencies.

Here is the complete pom.xml file for your reference:

```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>3.0.0-M3</version>
                <relativePath/> <!-- lookup parent from repository -->
        </parent>
        <groupId>net.javaguides</groupId>
        <artifactId>email-service</artifactId>
        <version>0.0.1-SNAPSHOT</version>
        <name>email-service</name>
        <description>Demo project for Spring Boot Email Service</description>
        <properties>
                <java.version>17</java.version>
        </properties>
        <dependencies>
                <dependency>
                        <groupId>org.springframework.boot</groupId>
                        <artifactId>spring-boot-starter-web</artifactId>
                </dependency>
                <dependency>
                        <groupId>org.springframework.kafka</groupId>
                        <artifactId>spring-kafka</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>
        </dependencies>

        <build>
                <plugins>
                        <plugin>
                                <groupId>org.springframework.boot</groupId>
                                <artifactId>spring-boot-maven-plugin</artifactId>
                        </plugin>
                </plugins>
        </build>
        <repositories>
                <repository>
                        <id>spring-milestones</id>
                        <name>Spring Milestones</name>
                        <url>https://repo.spring.io/milestone</url>
                        <snapshots>
                                <enabled>false</enabled>
                        </snapshots>
                </repository>
        </repositories>
        <pluginRepositories>
                <pluginRepository>
                        <id>spring-milestones</id>
                        <name>Spring Milestones</name>
                        <url>https://repo.spring.io/milestone</url>
                        <snapshots>
                                <enabled>false</enabled>
                        </snapshots>
                </pluginRepository>
        </pluginRepositories>
```

```
</project>
```

Create BaseDomains Microservice

Let's create a Spring boot project using **https://start.spring.io/**

Use the below details while creating a `base-domains` project:

```
        <groupId>net.javaguides</groupId>
        <artifactId>base-domains</artifactId>
        <version>0.0.1-SNAPSHOT</version>
        <name>base-domains</name>
        <description>Demo project for Spring Boot and Base Domains</description>
        <properties>
                <java.version>17</java.version>
        </properties>
```

Add Lombok dependency.

Here is the complete `pom.xml` file for your reference:

```
<?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>3.0.0-M3</version>
                <relativePath/> <!-- lookup parent from repository -->
        </parent>
        <groupId>net.javaguides</groupId>
        <artifactId>base-domains</artifactId>
        <version>0.0.1-SNAPSHOT</version>
        <name>base-domains</name>
        <description>Demo project for Spring Boot and Base Domains</description>
        <properties>
                <java.version>17</java.version>
        </properties>
        <dependencies>
                <dependency>
                        <groupId>org.springframework.boot</groupId>
                        <artifactId>spring-boot-starter</artifactId>
                </dependency>

                <dependency>
                        <groupId>org.projectlombok</groupId>
                        <artifactId>lombok</artifactId>
                        <optional>true</optional>
                </dependency>
                <dependency>
                        <groupId>org.springframework.boot</groupId>
                        <artifactId>spring-boot-starter-test</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>
                </plugins>
        </build>
        <repositories>
                <repository>
                        <id>spring-milestones</id>
                        <name>Spring Milestones</name>
                        <url>https://repo.spring.io/milestone</url>
                        <snapshots>
                                <enabled>false</enabled>
                        </snapshots>
                </repository>
        </repositories>
        <pluginRepositories>
                <pluginRepository>
                        <id>spring-milestones</id>
                        <name>Spring Milestones</name>
                        <url>https://repo.spring.io/milestone</url>
                        <snapshots>
                                <enabled>false</enabled>
                        </snapshots>
                </pluginRepository>
        </pluginRepositories>

</project>
```

Import and Setup 4 Microservices in IntelliJ IDEA

Follow the steps to import all the microservices in a single IntelliJ Idea workspace.

1. Create a main folder named "springboot-kafka-microservices".

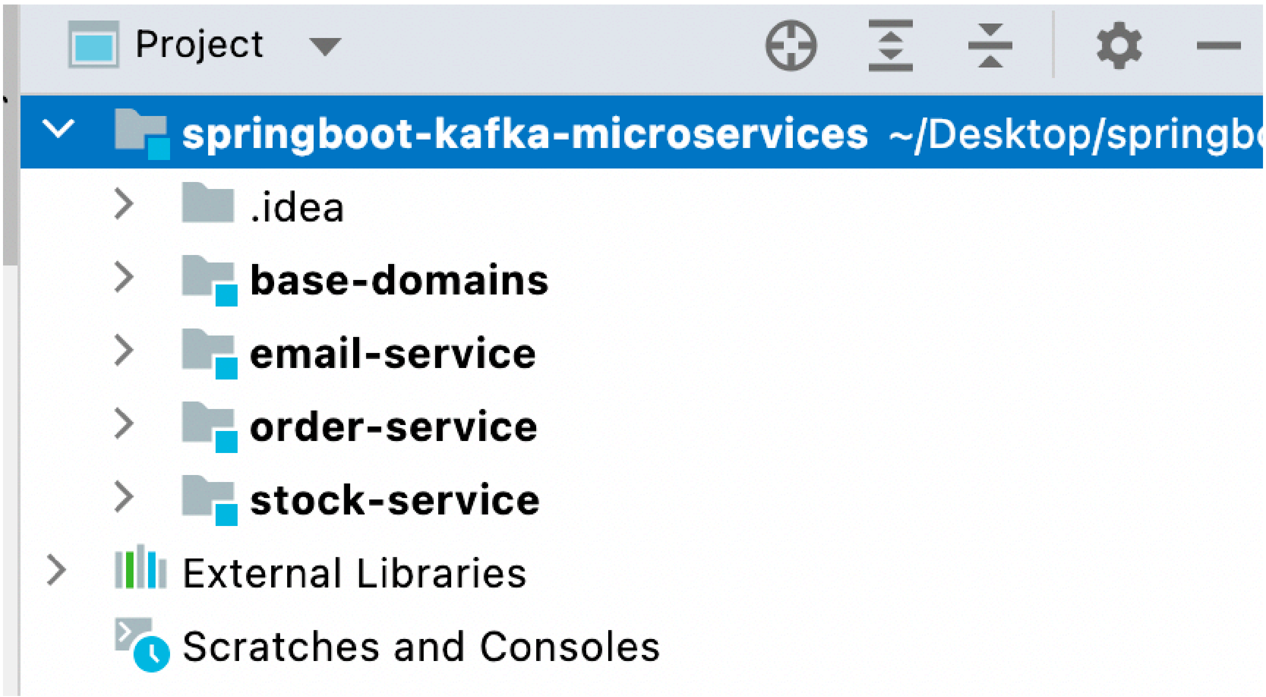
2. Move all the microservices to the main folder "springboot-kafka-microservices".

3. Import the main folder "springboot-kafka-microservices" in IntelliJ IDEA.

4. Load the maven scripts for all the spring boot microservices

Make sure that the pom.xml should be loaded for all the microservices in IntelliJ IDEA.

Here is the complete project setup screenshot for your reference:



Make sure to change the Tomcat server ports as shown below:

1. Keep 8080 default port for the order-service project

2. Change server port for `stock-service` as 8081 in an application.properties file:

```
server.port=8081
```

3. Change server port for an `email-service` as 8082 in an application.properties file:

```
server.port=8082
```

Create Order and OrderEvent DTO Classes in Base-Domains

Open base-domains microservice and quickly create `Order` and `OrderEvent` DTO classes.

Order

```
package net.javaguides.basedomains.dto;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@AllArgsConstructor
@NoArgsConstructor
public class Order {
    private String orderId;
    private String name;
    private int qty;
    private double price;
}
```

OrderEvent

```
package net.javaguides.basedomains.dto;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@AllArgsConstructor
@NoArgsConstructor
public class OrderEvent {
    private String message;
    private String status;
    private Order order;
}
```

Configure Kafka Producer in an OrderService Microservice

Open the `application.properties` file of the `order-service` project and configure Kafka producer.

```
spring.kafka.producer.bootstrap-servers: localhost:9092
spring.kafka.producer.key-serializer: org.apache.kafka.common.serialization.StringSerializer
spring.kafka.producer.value-serializer: org.springframework.kafka.support.serializer.JsonSerializer
spring.kafka.topic.name=order_topics
```

Let's understand the meaning of the above properties.

We are using the following Producer property to convert Java object into JSON:

```
spring.kafka.producer.value-serializer: org.springframework.kafka.support.serializer.JsonSerializer
```

`spring.kafka.producer.bootstrap-servers` - Comma-delimited list of host: port pairs to use for establishing the initial connections to the Kafka cluster. Overrides the global property, for consumers.

`spring.kafka.producer.key-serializer` - Serializer class for keys.

`spring.kafka.producer.value-serializer` - Serializer class for values

Configure Kafka Topic in OrderService Microservice

In an order-service project, create a `config` package. Within the `config` package, create a class named `KafkaTopicConfig` and add the following configuration to create the Kafka topic.

```
package net.javaguides.orderservice.config;

import org.apache.kafka.clients.admin.NewTopic;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.kafka.config.TopicBuilder;

@Configuration
public class KafkaTopicConfig {

    @Value("${spring.kafka.topic.name}")
    private String topicName;

    // spring bean for kafka topic
    @Bean
    public NewTopic topic(){
        return TopicBuilder.name(topicName)
                .build();
    }
}
```

Create Kafka Producer in OrderService Microservice

In an order-service project, create a package named `kafka`. Within a `kafka` package, create a class named `OrderProducer` and add the following content to it:

```
package net.javaguides.orderservice.kafka;

import net.javaguides.basedomains.dto.OrderEvent;
import org.apache.kafka.clients.admin.NewTopic;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.kafka.core.KafkaTemplate;
import org.springframework.kafka.support.KafkaHeaders;
import org.springframework.messaging.Message;
import org.springframework.messaging.support.MessageBuilder;
```

```java
import org.springframework.stereotype.Service;

@Service
public class OrderProducer {

    private static final Logger LOGGER = LoggerFactory.getLogger(OrderProducer.class);

    private NewTopic topic;

    private KafkaTemplate<String, OrderEvent> kafkaTemplate;

    public OrderProducer(NewTopic topic, KafkaTemplate<String, OrderEvent> kafkaTemplate) {
        this.topic = topic;
        this.kafkaTemplate = kafkaTemplate;
    }

    public void sendMessage(OrderEvent event){
        LOGGER.info(String.format("Order event => %s", event.toString()));

        // create Message
        Message<OrderEvent> message = MessageBuilder
                .withPayload(event)
                .setHeader(KafkaHeaders.TOPIC, topic.name())
                .build();
        kafkaTemplate.send(message);
    }
}
```

Note that we are sending an `OrderEvent` object to a Kafka Topic.

We created a `KafkaTemplate<String, OrderEvent>` since we are sending Java Objects to the Kafka topic that'll automatically be transformed into a JSON byte[].

In this example, we created a `Message` using the `MessageBuilder`. It's important to add the topic to which we are going to send the message too.

Create REST API to Send Order and Test Kafka Producer in OrderService Microservice

Let's create a simple POST REST API to send Order information as a JSON object.

In an order-service project, create a package named `controller`. Within a `controller` package, create a class named `OrderController` and add the following content to it:

```java
package net.javaguides.orderservice.controller;

import net.javaguides.basedomains.dto.Order;
import net.javaguides.basedomains.dto.OrderEvent;
import net.javaguides.orderservice.kafka.OrderProducer;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import java.util.UUID;

@RestController
@RequestMapping("/api/v1")
public class OrderController {

    private OrderProducer orderProducer;

    public OrderController(OrderProducer orderProducer) {
        this.orderProducer = orderProducer;
    }

    @PostMapping("/orders")
    public String placeOrder(@RequestBody Order order){

        order.setOrderId(UUID.randomUUID().toString());

        OrderEvent orderEvent = new OrderEvent();
        orderEvent.setStatus("PENDING");
        orderEvent.setMessage("order status is in pending state");
        orderEvent.setOrder(order);

        orderProducer.sendMessage(orderEvent);

        return "Order placed successfully ...";
    }
}
```

Configure Kafka Consumer in StockService Microservice

Open the `application.properties` file of the stock-service project and configure Kafka consumer.

```
server.port=8081
spring.kafka.consumer.bootstrap-servers: localhost:9092
spring.kafka.consumer.group-id: stock
spring.kafka.consumer.auto-offset-reset: earliest
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.trusted.packages=*
spring.kafka.topic.name=order_topics
```

We are using the following Consumer property to convert JSON into Java object:

```
spring.kafka.consumer.value-deserializer: org.springframework.kafka.support.serializer.JsonDeserializer
```

Let's understand the meaning of the above properties.

`spring.kafka.consumer.bootstrap-servers` - Comma-delimited list of host:port pairs to use for establishing the initial connections to the Kafka cluster. Overrides the global property, for consumers.

`spring.kafka.consumer.group-id` - A unique string that identifies the consumer group to which this consumer belongs.

`spring.kafka.consumer.auto-offset-reset` - What to do when there is no initial offset in Kafka or if the current offset no longer exists on the server.

`spring.kafka.consumer.key-deserializer` - Deserializer class for keys.

`spring.kafka.consumer.value-deserializer` - Deserializer class for values.

Create Kafka Consumer in StockService Microservice

Let's create a Kafka Consumer to receive JSON messages from the topic.

In a stock-service project, create a package named `kafka`. Within a `kafka` package, create a class named `OrderConsumer` and add the following content to it:

```java
package net.javaguides.stockservice.kafka;

import net.javaguides.basedomains.dto.OrderEvent;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.kafka.annotation.KafkaListener;
import org.springframework.stereotype.Service;

@Service
public class OrderConsumer {

    private static final Logger LOGGER = LoggerFactory.getLogger(OrderConsumer.class);

    @KafkaListener(
            topics = "${spring.kafka.topic.name}"
            ,groupId = "${spring.kafka.consumer.group-id}"
    )
    public void consume(OrderEvent event){
        LOGGER.info(String.format("Order event received in stock service => %s", event.toString()));

        // save the order event into the database
    }
}
```

Note that we are using `@KafkaListener` annotation to receive messages/events from the Kafka topic.

Configure and Create Kafka Consumer in EmailService Microservice

Open the `application.properties` file of the `email-service` project and configure Kafka consumer.

```
server.port=8082
spring.kafka.consumer.bootstrap-servers: localhost:9092
spring.kafka.consumer.group-id: email
spring.kafka.consumer.auto-offset-reset: earliest
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.trusted.packages=*
spring.kafka.topic.name=order_topics
```

We are using the following Consumer property to convert JSON into Java object:

```
spring.kafka.consumer.value-deserializer: org.springframework.kafka.support.serializer.JsonDeserializer
```

Let's understand the meaning of the above properties.

`spring.kafka.consumer.bootstrap-servers` - Comma-delimited list of host:port pairs to use for establishing the initial connections to the Kafka cluster. Overrides the global property, for consumers.

`spring.kafka.consumer.group-id` - A unique string that identifies the consumer group to which this consumer belongs.

`spring.kafka.consumer.auto-offset-reset` - What to do when there is no initial offset in Kafka or if the current offset no longer exists on the server.

`spring.kafka.consumer.key-deserializer` - Deserializer class for keys.

`spring.kafka.consumer.value-deserializer` - Deserializer class for values.

Let's create a Kafka Consumer to receive JSON messages from the topic.

In a email-service project, create a package named `kafka`. Within a `kafka` package, create a class named `OrderConsumer` and add the following content to it:

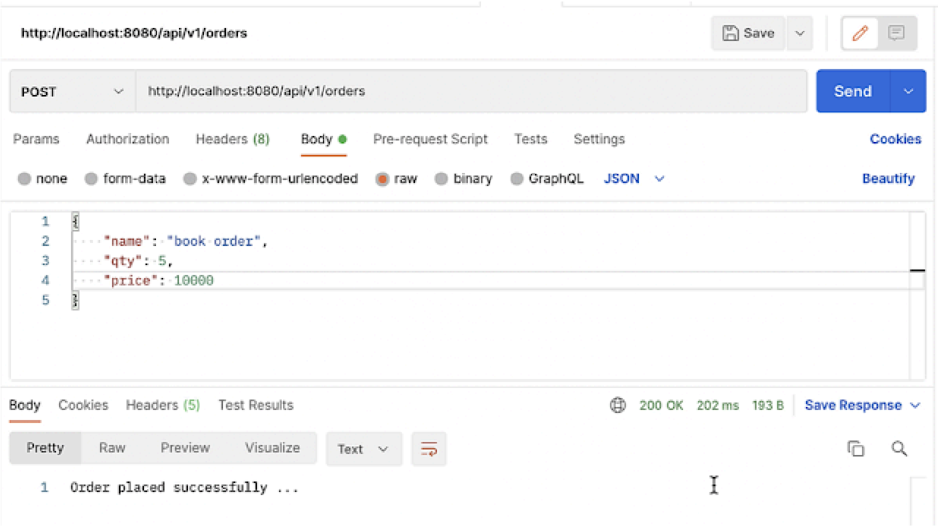
```
package net.javaguides.emailservice.kafka;

import net.javaguides.basedomains.dto.OrderEvent;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.kafka.annotation.KafkaListener;
import org.springframework.stereotype.Service;

@Service
public class OrderConsumer {

    private static final Logger LOGGER = LoggerFactory.getLogger(OrderConsumer.class);

    @KafkaListener(
            topics = "${spring.kafka.topic.name}"
            ,groupId = "${spring.kafka.consumer.group-id}"
    )
    public void consume(OrderEvent event){
        LOGGER.info(String.format("Order event received in email service => %s", event.toString()));

        // send an email to the customer
    }
}
```

Run 3 Microservices Together and Have a Demo

Let's run all the below 3 Spring boot microservices projects:

1. order-service
2. stock-service
3. email-service

Let's make a REST API call to send an Order:



POST http://localhost:8080/api/v1/orders

```
{
    "name": "book order",
    "qty": 5,
    "price": 10000
}
```

```
Order placed successfully ...
```

Next, verify the logs in the console for all the 3 microservices.

Conclusion

In this tutorial, you learned how to build a simple Event-Driven Microservices application using Spring Boot and Apache Kafka. You also learned how to create multiple consumers who will subscribe to a single Kafka topic to consume the events/messages.