Event Driven Architecture using Kafka

What’s exactly event driven architecture ?

A diagram of a service

Description automatically generated

Order service is communicating synchronously with Inventory Service, waits for a response then continues its behaviour, then we have another type of communication: asynchronously where the order service will make a request and forget about the response, so this kind of communication is called as asynchronous communication, this can be enabled with event driven architecture.

In our case we will receive an OrderPlacedEvent object event as a response and we will place it in Kafka broker. Our notification server will be the consumer, the order service will be a producer of the message and the notification service will be a consumer so it will process the message accordingly, as sending an email.

Install Kafka & set up kafka broker- take this Docker compose file:

Create a docker-compose.yml file in the root of the folder and use this configuration:

A screenshot of a computer

Description automatically generated

Zookeeper is used to orchestrate kafka clusters, so we need a zookeeper instance even if we have only 1 kafka clusters

Open the terminal and run:

* docker compose up – d

This will check if you have the images on your machine, if not it will start downloading them.

Add the dependency in your microservice pom, in our case under the order-service microservice:

A computer screen shot of white text

Description automatically generated, A flag on a grey background

Description automatically generated

Then under application.properties we will add.

A black background with orange text

Description automatically generated

The list of servers where we can find kafka installations so as we are using local installation we can just provide the url as localhost:9092

Now, under the OrderService we are going to call a Kafka Cluster whenever an order is placed .

We are going to inject the KafkaTemplateClass:

A screen shot of a computer program

Description automatically generated

Then we are going to send a notification topic together with an orderNumbere so that the notification server will understand what is the order number

A screen shot of a computer code

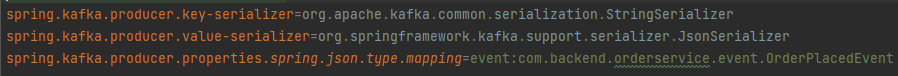
Description automatically generated

Under a newly created event package we will create and OrderPlaceEvent class

A screenshot of a computer program

Description automatically generated

We also have to provide spring how to serialize these key and value pair when sending to the kafka broker, so in application.properties we’ll have:



Now we need to create the notification service microservice.

After creating the new module we should be able to see it under root pom’s <modules> section, as it follows:

A screen shot of a computer program

Description automatically generated

We will create a new class for Listening to kafka notifications, under notification service module

A screen shot of a computer program

Description automatically generated

We need to annotate with @KaflaListener in order to listen to the topic called as “notificationTopic” as we defined it in application.properties.

In order for everything to work, we need to configure the application.properties with the deserializer of the topic as it follows:

A screen shot of a computer program

Description automatically generated

Testing:

We’ll create a POST RequestA screenshot of a computer

Description automatically generated

After we send the request

We should receive in our Notification Service Console the following:

