Apache Kafka Project

Bruno Morelli Cristian Lo Muto Vincenzo Martelli

Contents

0.1 Introduction

The application consists of a frontend that accepts requests from users and a backend that processes them. There are three types of users interacting with the service: (1) normal customers register, place orders, and check the status of orders; (2) admins can change the availability of items; (3) delivery men notify successful deliveries. The backend is decomposed in three services, following the microservices paradigm: (1) the users service manages personal data of registered users; (2) the orders service processes and validates orders; (3) the shipping service handles shipping of valid orders. Upon receiving a new order, the order service checks if all requested items are available and, if so, it sends the order to the shipping service. The shipping service checks the address of the user who placed the order and prepares the delivery.

0.2 Approach and Assumptions

Our food delivery application follows a microservices architecture, where the backend services interact asynchronously with each other using Kafka functionalities. The application leverages the HTTP protocol for communication between the frontend and the services, adopting a RESTful design realized using the Spring framework. With Spring, we benefit from its extensive support for creating RESTful APIs, handling request routing, data serialization, and managing the overall application lifecycle. Authentication is assumed to be functioning correctly. This allowed us to focus on the core functionalities of the application and on the interaction between the different services.

0.3 Design

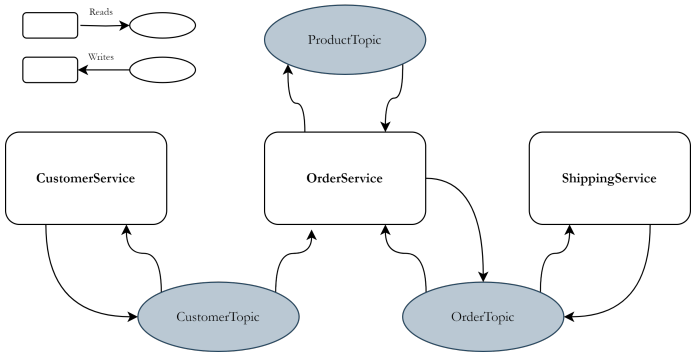

Figure 1: Services-Kafka interaction

The different services in our food delivery application interact asynchronously with each other through Kafka topics. Messages exchanged between services are formatted as strings, with attributes separated by the '#' symbol.
To ensure fault tolerance, object data are locally stored as Java objects and retrieved upon application restart or in the event of an error by reading the log of Kafka topics. As for the interaction:

- Regular users can interact with the *CustomerService* to register for the application, utilize the *OrderService* to purchase items, and leverage the shipping service to retrieve the status of their orders.
- Admins have the capability to insert new products through the *OrderService*.
- The delivery personnel can mark orders as shipped via the *ShippingService*.

All incoming http requests are handled by Spring RestControllers, one for each of the three services. The frontend is a simple command-line interface made in Python that sends HTTP requests to the backend services. All HTTP functions are provided by the *request* library.

0.3.1 CustomerService

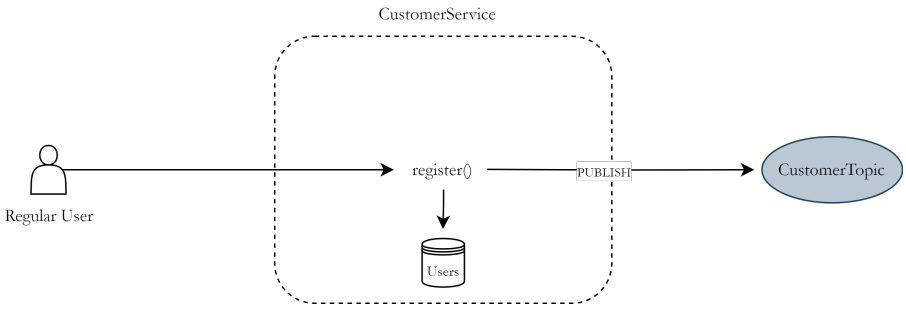

Figure 2: CustomerService

Allows regular user to register to the application. User data are stored as User objects. Whenever a new user subscribes to the application a message containg all the object fields is sent by to the *CustomerTopic*. The same topic is used to retrieve user data in case a failure occurs.

0.3.2 OrderService

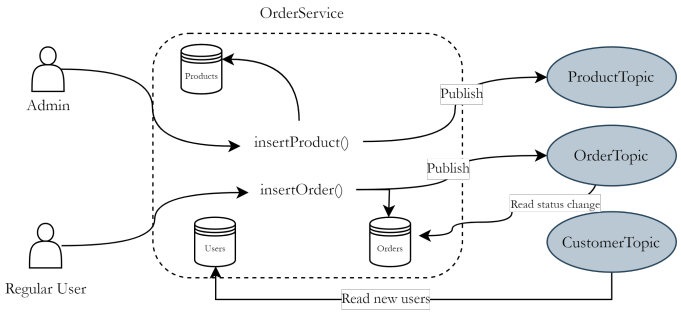

Figure 3: OrderService

The *OrderService* manages order insertion and products insertion/update. Orders are stored as *Order* objects and contain a reference to a product and the requested amount. Products are stored as *Product* objects.Upon a new order insertion, the service verifies if the requested amount is available and if the user is correctly subscribed, if so the order status is marked as VALID. If the order gets validated a message representing the order is sent to the *OrderTopic* so that it can be selected by shipping operators for the delivery. Simultaneously, a thread reads messages of *OrderTopic* sent by the *ShippingService* to check for status updates. If an order gets marked as ABORTED (meaning the shipping has failed) then the initial amount of the corresponding product is restored. All modifications to the quantity of a product are reported as messages to the *ProductTopic* so that they can be easily retrieved if a failure occurs. Fault tollerance for Orders and Users happens in the same way as previously mentioned.

0.3.3 ShippingService

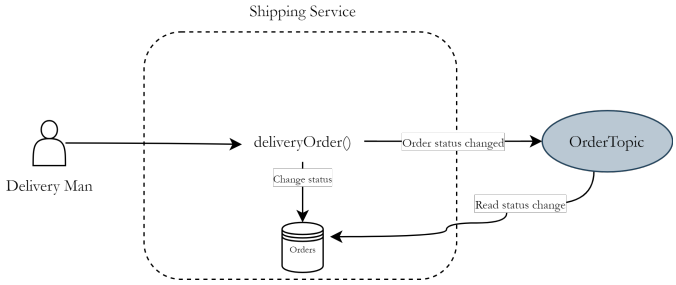

Figure 4: CustomerService

The *ShippingService* manages orders shipping. For testing purposes, whenever it reads a messages from *OrderTopic* there is a 20 % chance for the order to be marked as ABORTED. If that's not the case it can be selected for shipping by delivery men and thus marked as SHIPPED. In both scenarios, the new status is communicated trough a message to *OrderTopic*.

0.4 Test

0.4.1 Installation

The installation process outlined in this section of the documentation covers the setup and configuration of the Food Delivery application. Please ensure that you have Java, Maven, Kafka, and Python installed and properly configured on your machine before proceeding with these steps.

1. Start the Kafka broker by launching Zookeeper and the Kafka server.
   - Start Zookeeper: `zookeeper-server-start.sh config/zookeeper.properties`
   - Start Kafka server: `kafka-server-start.sh config/server.properties`
2. Open the `application.properties` file in the project directory. This file contains various configuration properties for the application.

3. Modify the `application.properties` file to specify which services you want to run. Each service has a corresponding property that can be set to `true` or `false` to enable or disable it, respectively.

4. To build the Spring application using Maven go in the source directory and run:

   - `mvn clean`
   - `mvn package -DskipTests`

5. Launch the application's JAR located in the source directory using JRE (Java version $\geq$17 is required).

   - `java -jar` *`jarFile`*

   This will generate the JAR file.

6. Ensure that you have the necessary Python dependencies installed. In particular, you need to have the `requests` package installed.

   - `pip install requests`

7. Once the `requests` package is installed, navigate to the frontend directory of the application using the terminal or command prompt.

8. Run the frontend of the application using the Python interpreter by executing the following command:

   - `python app.py`

To demonstrate the functionality of our app in a distributed scenario, we utilized virtual machines to host each microservice. One virtual machine also acted as the Kafka Broker, while the front end can run on any device within the local network.

0.5 References

- Kafka Documentation
- Spring Documentation