Winter 2022-CS 687 Capstone Project Progress Report
Kafka Streaming Application using Java Spring Boot

Anand Mohan
Advisor: Dr. Sion Yoon
MS in Computer Science
School of Technology & Computing (STC)
City University of Seattle (CityU)

Abstract

The purpose of this research paper is to implement a microservice approach for data streaming applications. Data is the new fuel to business, and it is growing enormously day by day. Handling and processing these data using the traditional database with batch processing is very slow and will not help businesses. So, there is a need for processing this data in real-time. Data streaming is one of the methods used for data processing in real-time. There is a mechanism that needs to publish and consume this data from the streaming application, and microservices help to achieve the same. Java and Spring framework provides an easy development environment to achieve this. Finally, a database is implemented to save the data.

Keywords: Kafka, Kafka Topics, Kafka Publishing and consuming, Streaming, Java, Spring Boot, Postgres, Relational Database

1. INTRODUCTION

As technology advances day by day, data is the new fuel for the business. So, good data can help to improve the business. For example, in the retail market, this data can help to understand the current trend of the market is, customers' choices, improve decision making, and more. So, when we say data, it is not just one or two. It is in the millions or trillions of data. Reading these data using standard REST APIs is not efficient and involves a lot of effort needed for implementation. So there comes the need to have an efficient way of handling data, and streaming is the best option for this. As part of this research, microservices will be developed which can publish and consume messages from streaming applications like Kafka and massage those data as per the need or requirement and save it in the database for specific use.

Problem Statement

With the market moving towards more streaming applications, there is a need to read these messages using any app and write them into database tables based on certain filters or requirements. This data in the table will be used for the app or future usage.

Recent market trends have shown that data streaming is the future. Many big companies like Netflix, Uber, Spotify heavily use data streaming. Reading these trillions of data in real-time helps to make decisions quickly and fast. As part of this research, microservices will be developed to read the streaming message from Kafka, and after consuming these messages, a filter is being applied based on the requirement. And finally, the required data will be saved in the database. Postgres database will be explored for this as it provides a relational open-source database.

Motivation

Publishing the message to Kafka and consuming the data with low latency is critical for the streaming app. Java spring boot technology provides many Kafka libraries that help to read

these messages with constant linear time. So, building a real-time streaming application and transforming that data to be customized as per need. Another important factor to consider during this flow is that there isn't any data loss, and better performance could be achieved.

Approach

The overall approach for this project can be divided into three parts

- Kafka deployment and implementation.
- Postgres database implementation and deployment.
- Java Spring boot app development and integration with Kafka and database.

Most of the information needed for the overall project can be obtained from the related books and by exploring the previous works done by others.

Conclusion

This project is aimed to have an end-to-end flow of streaming applications. The messages should be produced in Kafka topics by Java app by a separate service and then consumed by the Java app by another service and finally the data to be persisted inside the Postgres database.

2. BACKGROUND

As part of this research paper, I was able to identify most of the tasks that need to be performed, and I decided to categorize them into three sections as below.

- Installing and spinning up Kafka local.
- Create a Java app using Spring boot for producing and consuming the message from Kafka.
- Once the message is read, save the details to PostgreSQL.

The main objective of this research paper is to understand the details of streaming applications. Apache Kafka and Confluent are the main providers for this. They publish streams of events to a broker. The consumers of these event streaming can access each stream as per their requirements and consume the preferred event, and once consumed, these events are then retained by the broker. One of the key advantages of the event streaming process is that the records are persistent. This helps to process both historical data as well as real-time data without the worry of deletion by a broker. Most of these system works on the principle of publish/subscribe methodology. This approach allows systems to transfer events to multiple consumers asynchronously. This way, it can be quoted that this is an effective way of decoupling subsystems and managing communications independently.

Microservice creation is the second task related to this research paper. With the fast moving and competitive market, there is a need to have technology also to adapt to these changes quickly. For this, microservice plays a vital role as these can start small and iterate fast. With Java and Spring Boot, the quick start is easily possible, and packaging these source codes as a jar helps to get installed in any machine irrespective of the operating system.

Relational databases provide an easy-to-query approach for the data. Oracle provides a good relational database, but that is expensive. So there needs to find an open-source relational database that is robust, secure, and scalable. PostgreSQL is one of the leading providers of the relational database.

3. RELATED WORK

As this research paper is more of technical paper. I was able to explore most of the concepts through the study books available. I also explored some of the related works performed related to data streaming. Some of the challenges that may encounter while implementing are mentioned below.

Literature Review

Why is there a need for data streaming?

The traditional way of handling big data is by batch processing. With this approach, some of the main disadvantages is that

- Creates stale data- As batch processing takes some time to process, there is a very high chance of processing the stale data.
- It can irreversibly transform the source data
- Source data is not persisted (htt9)

The market is very competitive now, and most of the big tech companies are moving towards an event streaming approach (Hanif, 2020). The main reasoning behind this is businesses want to consume, process, and analyze data quickly to make the decision that will impact the business directly (Tu, 2020). Traditional batch processing may take weeks or even months to process data and will have adverse impacts on business.

How does Spring Boot Java application help in easy development of data streaming?

Studies have shown that the Spring Boot framework, which is a subproject under Spring, is becoming very popular (Zhang, 2021). The

main advantage of the Spring Boot framework is that the Tomcat web server is integrated within and can run directly without deployment. Traditional framework heavily uses XML configuration, and this can be overcome by spring boot annotations. Development with the usage of annotation is fast and reduces the effort of a development cycle. As this research includes integration with third-party streaming applications like Confluent Kafka, Database, spring boot can automatically configure and manage dependencies, making it ease the development activity to integrate with this third-party application.

What is the importance of Database?

For any application, there is a need for the system to effectively store and manage data (Lubis, 2017). Database helps in this process. It organizes, describes, and store according to a certain data model. There comes the next question of whether to choose a relational database or a non-relational database. Research studies have shown that relational database is easy to use and provide more data accuracy (Fotache, 2013).
For this research paper, I opted for a relational database and should be open source. PostgreSQL is one of the leading providers of open-source relational database. It is easy to install and manage.

Review Conclusions

The studies on streaming applications and microservices show that a systematic approach is needed (Korolov, 2017). There is a need to come up with a solid design process on how to integrate these different modules. Careful and precise external libraries should be picked while developing the microservices. Extra unwanted libraries will become a burden and will make the service heavy and should be avoided.
Testing and validation is another aspect to consider once the service is fully functional. This will help in getting the issues that may encounter during its implementation and in the future enhancements.

4. APPROACH

After reading through the textbooks related to Kafka Streaming, Microservices, and Databases, I was able to come up with a broader picture of the approach that needs to be followed. A piece of very basic customer information containing the first name and last name will be streamed through the Kafka topics. More insight on the approach is detailed below.

User Requirements

Below are some of the high-level user requirements.

- Customer basic information like first name and last name are only published in the Kafka topics.
- The Java app should start in the local machine without any error.
- Kafka broker and Zookeeper should be started locally without any error.
- The microservice should be able to publish to Kafka topics successfully after successful authentication.
- The microservice should be able to consume the messages from Kafka topics after successful authentication.
- After consuming, the microservice should be able to write the details to the PostgreSQL database successfully.

Design

An overall End-to-End design is shown in Figure 1.

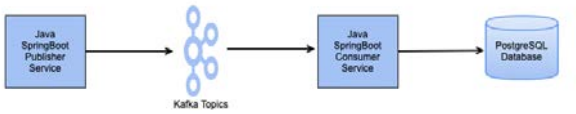

Figure 1: Overall End-to-End Design

The publisher service is only responsible for publishing the message to Kafka topics, and the consumer service is only responsible for reading the message from the Kafka topics. The entity class defined in the Java app is responsible for writing the message into the database.

Implementation

Kafka

Apache Kafka will be first installed in the machine, and once the basic configuration is set up, it will be started in the local machine. Once the Kafka application starts running, topics will be created for publishing and consuming. (Shapira, 2021)Two partitions will be created for topics to have better scalability and replication. Messages to Kafka are produced as key-value pairs, where the key will be a unique identifier to identify the message, and the value will be the actual data. For the research purpose, a JSON representation of the data will be produced to Kafka topics. Here for the project, this JSON data consist of first name and last name. A sample payload for the data is shown in Figure 2 (Narkhede, 2017).

```
{
    "firstName": "First Name",
    "lastName": "Last Name"
}
```

Figure 2: Sample Kafka JSON data

Microservices

The Java Spring Boot application will be developed as shown on the design page. As part of this research paper, the publisher and consumer will be developed under one Java app, and the services, i.e., the publisher and consumer service, will be created under separate classes with respective annotations. Kafka libraries for Spring Boot have many annotation methods that allow to publish and consume under the same app in parallel (Heckler, 2021) (Newman, 2021).

The consumer group will consume the JSON message from the topic and will be parsed and converted to an object class. A database entity class and its service defined in the Java app helps to write these details into the database easily. (Mitra, 2021)

PostgreSQL

Figure 3 shows the database table and column details

CustomerInfo		
Column Name	Data Type	Nullable?
correlation_id	Varchar (64)	No (pk)
first_name	Varchar (32)	No
last_name	Varchar (32)	Yes

Figure 3: Database Table details

The table name will be CustomerInfo in the PostgreSQL and will be created under the schema sch_customer. This table holds the customer information that is being produced in the Kafka topics. The primary key for the table will be the correlation id. This value is the same as the key value in the Kafka message. The first name and last name columns hold the respective values from the Kafka value (Ferrari, 2020).

Technologies Used

Apache Kafka

Apache Kafka is an open-source distributed event streaming platform for high-performance data pipelines, streaming analytics, data integration application (Andre, 2021).

Java with Spring Boot Framework

Spring Boot application helps developers to create standalone applications that just run. It comes with an embedded Tomcat web server which helps to achieve this functionality without relying on an external web server. Also, auto-configuration and annotation are powerful methods available with spring boot which reduce many boilerplate codes.

PostgreSQL

Postgres is a powerful open-source relational database system with reliability, feature robustness, and performance. This database is highly extensible, i.e., developers can define their data types, build custom functions, and write code in different languages without recompiling the database.

5. Data Collection

This project does not contain any big data and must check only that the data that is being produced in the Kafka topic is finally get updated in the database. So, it would be a good approach to validate the data as a checklist. So, data collection will be divided into three parts

- The data that is being produced by the Java Spring Boot app
- The data that is being consumed by the Java Spring Boot app
- The data is being persisted inside the Postgres Database.

Java Spring Boot Producing message

The app is responsible for producing the message to Kafka topic. As part of this project, this will be handled inside the code by a separate service by randomly generating the data in the format mentioned in Figure 2. Load testing is not in the scope of this project, so a large load generation of data is not required. This data will be shown in the logs by the app and can be captured as part of the Kafka produce logs.

Java Spring Boot Consuming message

The consumption of the message is performed by the consumer service. The details will be logged as part of the log module inside the service, and the consumed data will be logged into the log modules and will be thrown into the console application.

Data is persisted in the Postgres Database

The final part is the data to be available in the Postgres database. As part of the project, there is a need to ensure that every message that is being produced into the Kafka topic should be available in the database. This data can be

collected by running a simple query against the database to get the complete details.

6. Data Analysis

This research paper is not intended for Artificial Intelligence or Machine Learning. So, as mentioned in the data collection section, the data is validated as a checklist.

Java Spring Boot Producing message

Figure 4 shows the log from the producer service. A message is being produced to a Kafka topic with a unique key. The message contains the first name and last name, which are being passed through a rest controller. The log service will log the details in the console output.

Java Spring Boot Consuming message

Figure 5 shows the log from the consumer microservice. The message that is being produced by the producer service is consumed by the consumer service, and the log service is responsible for logging the details in the console output.

Data is persisted in the Postgres Database

The final part is to verify that the data is available in the database table. As shown in Figure 6, the data that is being published on the Kafka topic is being added to the database table. The correlation id corresponds to the unique key that is generated by the service during publishing.

7. Finding

Some of the key findings of the research paper are divided into two parts. First, the libraries that are available for the development work, and second, the performance of the execution for the data flow from an end-to-end perspective.

Libraries for Development

As I was using the Spring Boot framework, the development of the entire application required interaction with Kafka and Postgres. Spring Boot framework comes with many standard third-party libraries that will ease the development. Many of the boilerplate codes that are needed will be handled by the libraries through the backend, which helps the developers just to focus on the development. The boilerplate code needed only the configuration value as per the project need. Because of this, the Java app integration with Kafka as well as Postgres worked well.

Performance of the Data

Though the performance of the application was not in the scope of the research, here, I am going to explain the performance of a single message. Figure 7 shows the end-to-end time taken for a single message. Within microseconds, a single data object published to the Kafka topic is available in the database. This shows the efficiency of the Kafka streaming application. So, in real-time, when there are millions of data to handle, it is obvious that the performance will not be compromised.

9. Conclusion

As mentioned in the introduction section, this research paper focuses on developing microservices that can publish and consume messages from streaming applications like Kafka and massage those data as per the need or requirement and save it in the database for specific use. This objective was fully achieved as part of this project work.

Kafka publishing and consuming can be performed in many ways, and this research paper explored the Json way of publishing and consuming data. It is also noticed that within microseconds, a single data published to a Kafka topic is available in the final database table. So, when coming to real-time use, where there are millions of data to be handled, it is obvious that the performance will not be compromised, and the app can scale quickly to satisfy the need.

10. Future Work

There is still a lot of work that could be done for this project. The project completely works on the local machines. As a first step, it can be explored how this app can be available beyond a local setup. For example, is there any open-source public cloud for Kafka and Postgres services?

As mentioned in the conclusion section, this research paper focuses on the JSON approach for data transmission. There is also another mode of data transmission called the Avro approach, and this uses the concept of the schema for Kafka topics. A separate study can be explored in this area on publishing and consuming messages to Kafka with schema registry and Avro approach.

11. REFERENCE

Tu, D. Q., Kayes, A. S. M., Wenny, R., & Nguyen, K. (2020). IoT streaming data integration from multiple sources. *Computing.Archives for Informatics and Numerical Computation, 102*(10), 2299-2329. http://dx.doi.org/10.1007/s00607-020-00830-9

Hanif, M., Lee, C., & Helal, S. (2020). Predictive topology refinements in distributed stream processing system. *PLoS One, 15*(11)http://dx.doi.org/10.1371/journal.pone.0240424

Zhang, F., Sun, G., Bowen, Z., & Liang, D. (2021). Design and Implementation of Energy Management System Based on Spring Boot Framework. *Information, 12*(11), 457. http://dx.doi.org/10.3390/info12110457

Andre, D. R., Freitas, N., Duarte Alemão, Guedes, M., Martins, R., & Barata, J. (2021). Event-Driven Interoperable Manufacturing Ecosystem for Energy Consumption Monitoring. *Energies, 14*(12), 3620. http://dx.doi.org/10.3390/en14123620

Korolov, M. (2017). Microservices offer speed and flexibility, but at a price: The benefits of microservices are many, but they also come with their own set of security and management challenges. *CSO (Online),* https://www.proquest.com/trade-journals/microservices-offer-speed-flexibility-at-price/docview/1873330691/se-2?accountid=1230

Lubis, A. R., Fachrizal, F., & Maulana, H. (2017). Database Management Optimization Using PostgreSQL Replication Database in Database System. *Advanced Science Letters, 23*(5), 4132-4135. http://dx.doi.org/10.1166/asl.2017.8286

Fotache, M., & Cogean, D. (2013). NoSQL and SQL Databases for Mobile Applications. Case Study: MongoDB versus PostgreSQL. *Informatica Economica, 17*(2), 41-58. https://www.proquest.com/scholarly-journals/nosql-sql-databases-mobile-applications-case/docview/1462815925/se-2

Heckler, M. (2021). Spring Boot: Up and Running (1st edition). O'Reilly Media, Inc.

Shapira, G., Palino, T., Sivaram, R., & Narkhede, N. (2021). Kafka: The Definitive Guide, 2nd Edition (2nd edition). O'Reilly Media, Inc.

Narkhede, N., Shapira, G., & Palino, T. (2017). Kafka: the definitive guide: real-time data and stream processing at scale. O'Reilly Media, Inc.

Mitra, R. & Nadareishvili, I. (2021). Microservices : up and running : a step-by-step guide to building a microservices architecture (1st edition). O'Reilly.

Newman, S. (2021). Building microservices : designing fine-grained systems (Second edition.). O'Reilly Media, Incorporated.

Ferrari, L., & Pirozzi, E. (2020). Learn PostgreSQL : build and manage high-performance database solutions using PostgreSQL 12 and 13 (1st edition). Packt Publishing.

APPENDIX-A

```
2022-02-27 01:02:20.092  INFO 32947 --- [nio-1338-exec-1] com.cityu.cs687.producer.KafkaProducer   :
Published message to Kafka successfully with:
key: a141087e-81da-424a-b11c-a1b6b5dfb60f
Message:
{
    "firstName":"CS687",
    "lastName":"Research Project"
}
```

Figure 4: Kafka Producer Service publishing

```
2022-02-27 01:02:20.107  INFO 32947 --- [ntainer#0-0-C-1] com.cityu.cs687.consumer.KafkaConsumer   :
Received Key: a141087e-81da-424a-b11c-a1b6b5dfb60f
2022-02-27 01:02:20.108  INFO 32947 --- [ntainer#0-0-C-1] com.cityu.cs687.consumer.KafkaConsumer   :
Received Message: {
    "firstName":"CS687",
    "lastName":"Research Project"
}
```

Figure 5: Kafka Consumer Service Consuming

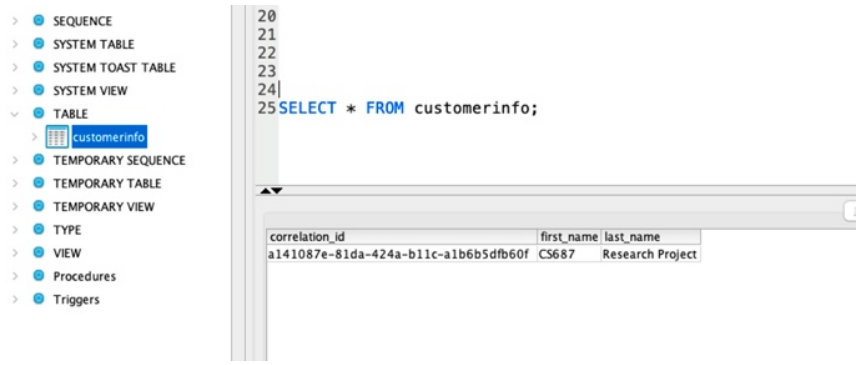

Figure 6: Postgres Database with the entry

```
2022-03-06 13:38:55.982  INFO 74564 --- [nio-1338-exec-8] com.cityu.cs687.producer.KafkaProducer   :
Published message to Kafka successfully with:
key: 48b102f9-1551-4e8a-b1c0-68acc762c08b
Message:
CustomerInfo(firstName=CS687_3, lastName=Research Project_3)
2022-03-06 13:38:55.987  INFO 74564 --- [ntainer#0-0-C-1] com.cityu.cs687.consumer.KafkaConsumer   :
Received Key: 48b102f9-1551-4e8a-b1c0-68acc762c08b
2022-03-06 13:38:55.987  INFO 74564 --- [ntainer#0-0-C-1] com.cityu.cs687.consumer.KafkaConsumer   :
Received Message: CustomerInfo(firstName=CS687_3, lastName=Research Project_3)
Hibernate:
    select
        customerin0_.correlation_id as correlat1_0_0_,
        customerin0_.first_name as first_na2_0_0_,
        customerin0_.last_name as last_nam3_0_0_
    from
        customerinfo customerin0_
    where
        customerin0_.correlation_id=?
Hibernate:
    insert
    into
        customerinfo
        (first_name, last_name, correlation_id)
    values
        (?, ?, ?)
2022-03-06 13:38:55.990  INFO 74564 --- [ntainer#0-0-C-1] com.cityu.cs687.consumer.KafkaConsumer   :
Saved to DB48b102f9-1551-4e8a-b1c0-68acc762c08b
```

Figure 7: End-to-End time taken for a single message

APPENDIX-B

Demonstration Video:
https://youtu.be/_jFUy3EyLTI

GitHub Repo:
https://github.com/mohananand-city/cs687_kafka_springboot