International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 07 Issue: 06 | June 2020 www.irjet.net p-ISSN: 2395-0072

Building Microservices with Event Sourcing: A Comprehensive Review

Kumar Abhijeet[1], Prof. Smitha G R[2]

[1]Student, Dept. of Information Science and Engineering, R V College of Engineering, Bengaluru
[2]Assistant Professor, Dept. of Information Science and Engineering, R V College of Engineering, Bengaluru

---

Abstract - *The software ecosystems all around the world are moving from a monolith architecture to microservices-based architecture as it allows a system to be divided into a number of smaller, individual and independent services. Each service is flexible, robust and complete. They run as independent processes and synchronize with one another through an API (Application Program Interface). Each microservice can be implemented in a different programming language or framework.*

Adopting the microservices architecture in an ecosystem makes it easier to build and maintain apps as the whole system is split into a set of smaller fragments, as they are quite simple as in contrast to a monolith with a huge number of dependencies. But, microservices comes with its own challenges as discussed in [1], as we broke up a huge monolith into separate entities, which makes it difficult to keep the data and state of all the components in unison and synchronized. Using event sourcing as a pattern to build microservices, will solve the above issue by acting upon as the events are created. Just like a cab service, in which the booking is created after the customer requests for it. The frameworks used to achieve an event sourcing based microservice architecture is a publisher-subscriber model with stream processing capabilities along with a messaging queue system. The paper discusses the components and the consolidation of an event sourcing based microservice architecture.

Keywords: ***Microservices, Event Sourcing, Software Design***

1.0 Introduction

In the event sourcing based architecture, when an event is published from a microservice, other microservices can be reactive to those events and publish another set of events. This model can be compared to UNIX pipes [2]. A single transaction in a microservices-based system may span into multiple microservices where we can perform a transaction as a sequence of events by building reactive microservices.

In this paper, we have performed a walkthrough of the basic ingredients of an event sourcing based microservice architecture also referred to as *event-driven architecture*, involving the publisher-subscriber model, event sourcing patterns and a stream processing mechanism.

2.0 Publisher/Subscriber Model

In distributed applications, the components of the system often need to provide information to other components as events happen [4]. In the Client/Server messaging system, clients send a request to the server and then wait for its response. Apache Kafka is used to implement a publisher-subscriber model, and it runs as a cluster on multiple servers and implements the publisher-subscriber model as follows.

Publisher refers to a sender, who sends the message. Subscriber refers to a recipient, receiving that message. The only difference is that the publisher publishes the message to a topic. The subscriber receives the message from the topic. Each topic consists of a separate set of events.

In a Kafka based publisher/subscriber messaging model, there is a broker to which all publishers and subscribers will be connected as depicted in Fig. 1.

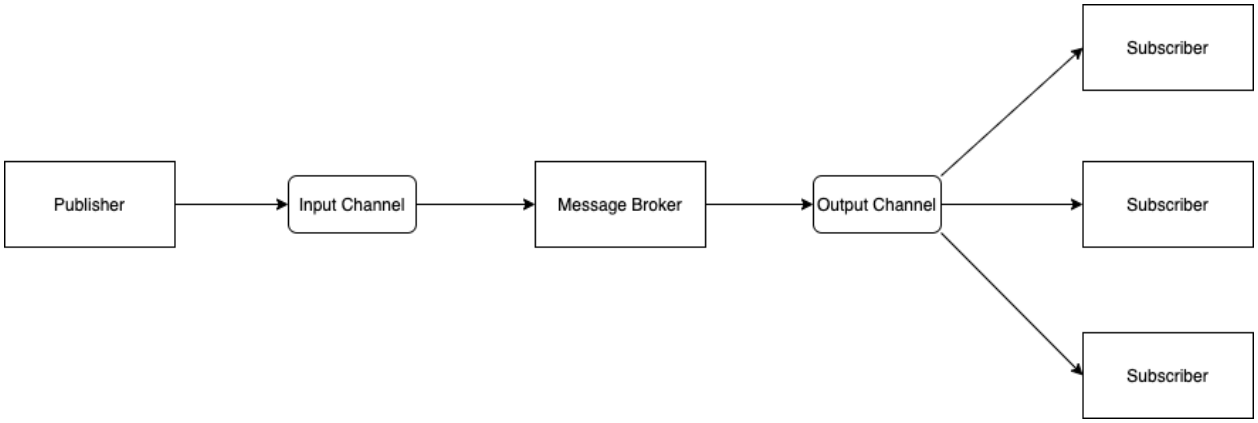

Fig. 1: All Senders communicating with all receivers through message broker

3.0 Event Sourcing Design Patterns

Event sourcing is a design pattern that defines an approach to handling operations on data that's driven by a sequence of events, recorded in an add-only store. Application code sends a series of events that describe each action that has occurred on the data to the event store, where they're persisted[11].

Let's consider a simple example to do with order tracking notifications. In this example we have many orders, and we need to know their location. A tracking application with methods to allow us to tell when an order arrives or leaves the hub can be used to keep a track of the orders.

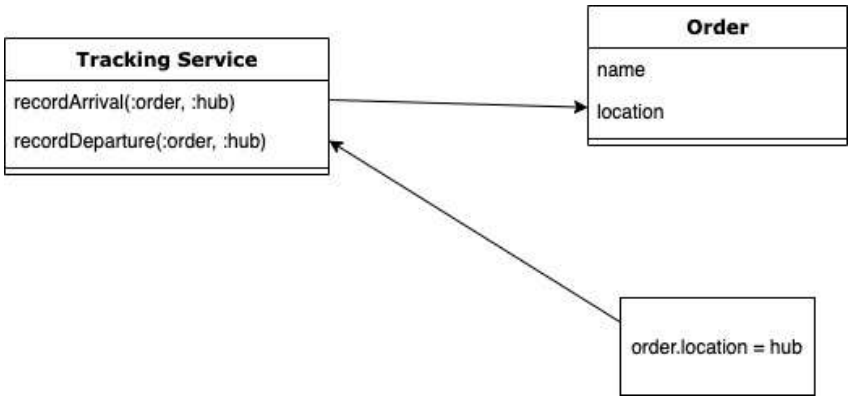

Fig. 2: A simple interface for tracking order shipping movements.

In this case, when the service is called, it finds the relevant order and updates its location. The order objects record the current known state of the order.

Introducing Event Sourcing adds a step to this process. Now the service creates an event object to record the change and processes it to update the location of the order.

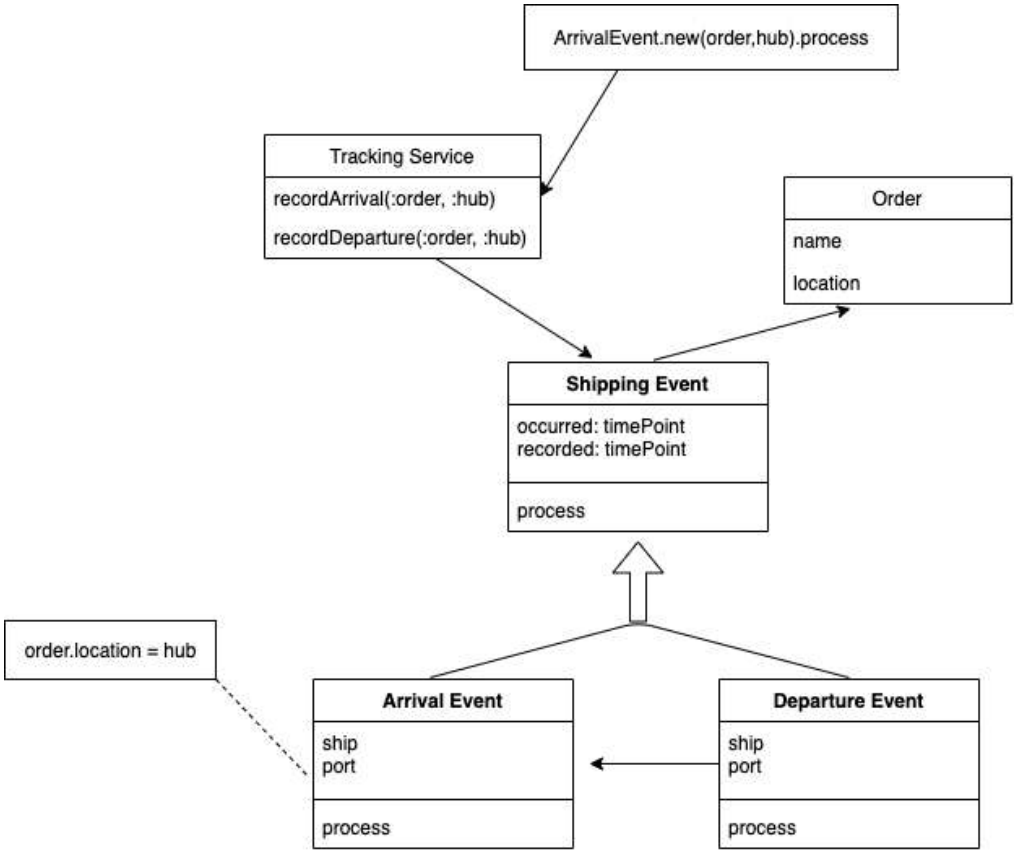

Fig 3: Using an event to capture the change.

The key to Event Sourcing is that all changes to the domain objects are initiated by the event objects and have kept the log of all the changes as well. The event log can be used to build a plethora of different facilities like time travel, temporal queries, and event replays. Hence, in a microservice architecture, the different microservices create and ingest events keeping an Event Log, which solves the data and state inconsistency.

4.0 Stream Processing Mechanism

Stream processing mainly involves the manipulation of a stream of data, which here is an *event*. Event stream processing works by handling a data set by one data point at a time. Rather than view data as a whole set, event stream processing is about dealing with a flow of continuously created data [7].

Kafka streams API is a great example of a stream processing framework and represents an unbounded, continuously updating dataset of immutable records where each record is defined as a key-value pair and incorporates the important streams processing concepts like properly distinguishing between event time and processing time, windowing support, and simple yet efficient management and real-time querying of application state. This is extensively used in the event-based architecture, to process and get viable data from the event streams in real-time, and synchronizes with Kafka to set up specific retry strategies upon data invalidation and other error checking patterns [6].

5.0 Putting the pieces together

The event-driven architecture discussed above, uses the above three modules in unison, to generate and process streams of events, which are facilitated by a publisher-subscriber model, enabling event sourcing in the ecosystem. Any microservice can emit an event to another service and will be processed through Kafka streams, then ingested to a Kafka model which acts on the event, and triggers the subscriber completing the workflow.

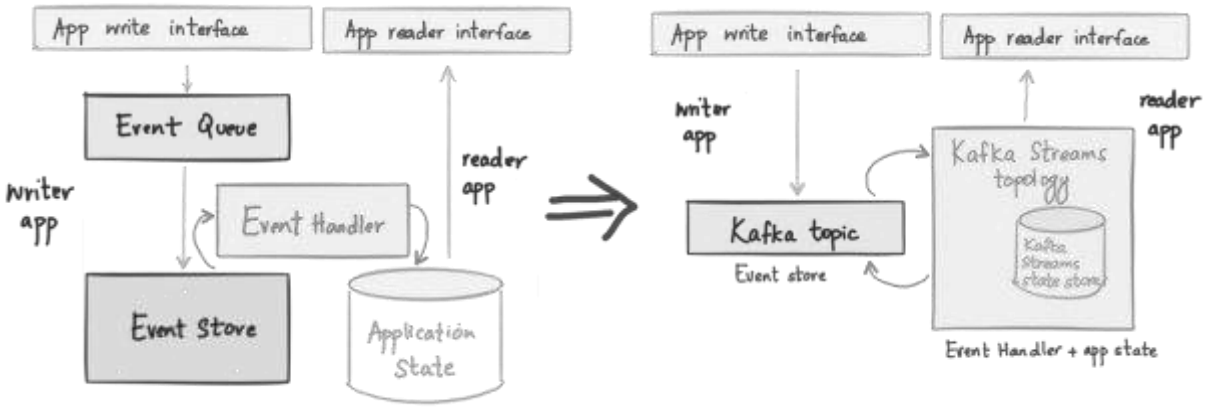

Fig. 4: A basic flow of an event-driven architecture

6.0 Conclusions

This paper presents a general study on event sourcing based microservice architecture and its essential components. An event-driven architecture is used widely in product based ecosystems, as this model is more scalable and reliable than its other counterparts [10]. Extensive use of Kafka can be done both as a publisher-subscriber model and a stream processing platform along with third part message broker systems as well. But, this architecture should not be used at all scenarios as discussed in [1], where consistency is the main concern. But in conclusion, where resilience and availability are desired, there is no better match than an event sourcing based architecture.

References

1. I. Soderquist, "Event Driven Data Processing Architecture," *2007 Design, Automation & Test in Europe Conference & Exhibition*, Nice, 2007, pp. 1-5.

2. B. Erb, D. Meißner, G. Habiger, J. Pietron and F. Kargl, "Consistent retrospective snapshots in distributed event-sourced systems," 2017 International Conference on Networked Systems (NetSys), Gottingen, 2017, pp. 1-8.

3. F. Y. Oh, S. Kim, H. Eom, H. Y. Yeom, J. Park and Y. Lee, "A scalable and adaptive cloud-based message brokering service," *13th International Conference on Advanced Communication Technology (ICACT2011)*, Seoul, 2011, pp. 498-501.

4. H. Kawazoe, D. Ajitomi and K. Minami, "A test framework for large-scale message broker system for consumer devices," *2015 IEEE 5th International Conference on Consumer Electronics - Berlin (ICCE-Berlin)*, Berlin, 2015, pp. 24-28.

5. B. R. Hiraman, C. Viresh M. and K. Abhijeet C., "A Study of Apache Kafka in Big Data Stream Processing," *2018 International Conference on Information , Communication, Engineering and Technology (ICICET)*, Pune, 2018, pp. 1-3.

6. K. Kato, A. Takefusa, H. Nakada and M. Oguchi, "A Study of a Scalable Distributed Stream Processing Infrastructure Using Ray and Apache Kafka," *2018 IEEE International Conference on Big Data (Big Data)*, Seattle, WA, USA, 2018, pp. 5351-5353.

7. X. J. Hong, H. Sik Yang and Y. H. Kim, "Performance Analysis of RESTful API and RabbitMQ for Microservice Web Application," *2018 International Conference on Information and Communication Technology Convergence (ICTC)*, Jeju, 2018, pp. 257-259.

8. L. Duan, C. Sun, Y. Zhang, W. Ni and J. Chen, "A Comprehensive Security Framework for Publish/Subscribe-Based IoT Services Communication," in *IEEE Access*, vol. 7, pp. 25989-26001, 2019.

9. D. Serain, "Client/server: Why? What? How?," *International Seminar on Client/Server Computing. Seminar Proceedings (Digest No. 1995/184)*, La Hulpe, Belgium, 1995, pp. 1/1-111 vol.1.

10. Xiwei Feng, Chuanying Jia and Jiaxuan Yang, "Semantic web-based publish-subscribe system," *2008 IEEE International Conference on Service Operations and Logistics, and Informatics*, Beijing, 2008, pp. 1093-1096