International Journal of Core Engineering & Management
Volume-5, Issue-11, February-2019, ISSN No: 2348-9510

EVENT-DRIVEN ARCHITECTURE WITH JAVA AND AWS

Prathyusha Kosuru
Project Delivery Specialist
Deloitte Consulting LLP
Salem, USA
prathyushakosuru308@gmail.com

Abstract

The integration of Java with AWS EventBridge enhances the creation of event-driven application that runs with an asynchronous architecture. AWS SQS and SNS can conveniently be used with Java to enable the decoupling and high responsiveness of distributed messaging services. Prior to February 2019, conventions such as event sourcing with Java and AWS DynamoDB Streams were necessary for building reliable systems that display how the state has evolved over time and provide the grip and reliability necessary to sustain event-driven microservices (Bermudez et al., 2014).

Keywords: Event-Driven Architecture (EDA), Java Programming, AWS Services, Microservices, Serverless Computing, Amazon SNS (Simple Notification Service), Amazon SQS (Simple Queue Service), AWS Lambda, Event Streaming, Apache Kafka, Message Broker

I. INTRODUCTION

In today's rapidly evolving digital environment, organizations are on a quest to improve their application architecture. Introducing event-driven architecture (EDA) – an active architectural style that focuses on handling real-time data and having asynchronous interactions. This modern approach lets systems react flexibly to event occurrences, enhancing the system's responsiveness and scalability. When used with other robust tools like Java and AWS, EDA changes how software developers work to develop applications. It paves the way for possibilities wherein every constituent interacts using an event-based system instead of a request-response way. Just think of an architecture you want to have that is as flexible with the fluctuations of your business processes (Clark & Barn, 2012).

II. EXPLANATION OF EVENT-DRIVEN ARCHITECTURE AND ITS SIGNIFICANCE

EDA stands for event- driven architecture and it has been described as a software design pattern of producing, detecting and handling events. What is more important in this model is the event technique where components pass information through events not directly. It also enables flexibility and scalability since the two components are untangled. It is generally because EDA is highly efficient in dealing with the data stream. It doesn't require waiting for pre-specified schemes, different form interactions against changes into an application's state. Accomplishing an occasion orientated strategy allows organizations to construct systems which will be more

appropriate and competent within emerging contexts. As a result, the functionality of the websites improves the user experience that customer gets, and efficiency. Furthermore, because it eliminates dependencies between services, and as such, EDA supports distributed systems. Therefore, it is possible for teams to create and release applications without much need to communicate extensively through the use of events. This architecture supports innovation because it can be updated and has faster iterations (Hofmann et al., 2017).

III. JAVA WITH AWS EVENTBRIDGE

AWS EventBridge is an advanced event bus that helps integrate applications through events. The goal is to provide smooth integration to bring the architecture of systems as created by various developers loosely connected. When Java is integrated with AWS Event Bridge, we can design the elastic and reactive app. You can send events from your Java application directly to EventBridge. This means your app can respond instantaneously with no need for all kinds of polling mechanisms. Java SDK does that, making interaction fluent. You can readily broadcast events or subscribe to them within an application architecture such as Spring Boot. This flexibility helps avoid focusing on infrastructure while allowing the team to concentrate on business logic. It also leads to better resource management for event-driven architectures, which are the successors of traditional architectures. It enables efficient processing of tasks, improving performance, cost characteristics of cloud services, and availability (Clark & Barn, 2012).

IV. USING JAVA APPLICATIONS WITH EVENTBRIDGE FOR REAL-TIME PROCESSING

It is possible to enhance several Java applications using Amazon Event Bridge to boost real-time processing. This serverless event bus allows developers to create applications that are always ready to respond to events and changes. AWS SDK for Java makes it very simple to work with Event Bridge. It begins with setting an event rule that outlines the behavior of your application with regard to certain events. This characteristic makes it possible to filter and direct events in a versatile manner. Among its attractive features, your Java application can publish events directly to EventBridge after being set up. They activate different kinds of targets, for example, Lambda functions or Step Functions, so that the processes run smoothly and work faster. The feature of interest here is that it can disaggregate elements in your architecture. Each service communicates asynchronously through events without being embedded very rigidly with each other. This reduces downtime and improves dependability while receiving high throughput constellations with less effort (Hofmann et al., 2017).

4.1 Advanced Features and Integration

EventBridge also provides some additional features that make it even more valuable for Java applications. One such feature is the event bus management feature which allows developers to define custom event buses so that event of a specific type can be isolated from the other events. This is especially important when there are many different events to organize within the scope of a large application. Still, if connected with other AWS services, the EventBridge is packed with another powerful feature. For instance, one can integrate it with Amazon CloudWatch by which it can monitor and log events as well. In one way, the integration allows developers to have better comprehension of their application.

4.2 Scalability and Cost-Effectiveness

A worse thing to say about EventBridge is that it is a naturally scalable solution when used with Java applications. This is because; it is a fully managed service that adapts to the amount of events received without prior intervention. This means that the functionality of your application can support heavy traffic at some point in its life or gradual increases in traffic as the days go by. Furthermore, Customers can only pay for the actual usage of EventBridge through its effective and flexible pay-per-use pricing model regardless of the business size. There are no license fees; you only pay for the processing of individual cases and that can be much cheaper than the maintenance of a separate event processing infrastructure.

4.3 Best Practices and Implementation

All Java applications that include EventBridge should adhere to the following best practices: This comprises subscribing to the right event patters when filtering events, implementing error and retry mechanisms and setting up dead-letter queues where unmatched events occurred. With these practices and the usage of EventBridge at their best, developers can make and develop scalable and efficient Java applications that perform excellent in realtime event processing and responsiveness.

V. INTEGRATING SQS AND SNS WITH JAVA

Accomplishing the Amazon SQS and SNS services in Java will revolutionize your Java application's messaging system. By leveraging these services, you establish strong microservices communication through which you can digitally enable them. First, one has to create an object where there are easy-to-use libraries available in AWS SDK for Java. As you recall, invoking and sending messages to an SQS queue or publishing a notice using SNS only requires three lines of code. By using SQS, it is absolutely guaranteed that a message is safe before it is processed by the recipient. This is specifically important for applications that require some level of failure tolerance. While with SNS, you can freely broadcast updates to as many subscribers as you want at the same time. Suppose certain user activities result in events that can be pushed using SNS while the same events are placed in SQS for subsequent treatment by worker services. The modularity that this synergy brings improves scalability and allows for the graceful separation of elements in your schema (Kim &Wellings, 2010).

VI. BENEFITS OF USING AMAZON SQS AND SNS IN EVENT-DRIVEN APPLICATIONS

AWS is valuable to event-driven applications when applied because of its modules, Amazon SQS and SNS, refine event-driven application functionalities significantly. They build a strong communication interface between various modules. SQS can be used to queue messages. This makes it possible for messages not to be lost in the instance that systems are down or are perhaps offline for some time. This allows it to support asynchronous processing, which may actually enhance the total performance of the system. However, SNS allows sending actual and real-time notifications to many subscribers at the very first instance. This feature is very important when some occurrence takes place, and something needs to be done on the spot. Altogether, SQS and SNS make the implementation of decoupled architectures possible. Services can run uninterrupted without the service on the opposite end, having to know its existence at all. Flexibility is achieved

when a company's demand grows or shrinks. Both services take care of scaling requirements as a background chore that does not require further intervention by the developers, thus giving them space to provide interesting new features to users while not having to worry about the lowest-level infrastructure challenges (Poornalinga& Rajkumar, 2016).

VII. EVENT SOURCING IN JAVA

The event sourcing, is a very influential pattern in the software architecture, especially when used in the Java language. The concept behind it is to save changes in the state not as the state itself but as an event that has occurred. This approach affords developers the ability to build an application state at some point in time. When implementing event sourcing in Java, a change is described as an event and is unalterable. These events are kept as a record within an append-only structure, so it is simple to monitor them and repeat the process if required. This is of great assistance in debugging and auditing exercises. In the case of applications that rely on event sourcing, Axon Framework shows reliable support for Java platforms. They simplify the response and management of commands, events and projections. One important benefit is that it improves the data synchronization of distributed applications. Because each action results in the recording of a new event, the historical record can always be kept accurate while accommodating the required business processes (Sbarski&Kroonenburg, 2017).

VIII. CONCEPT OF EVENT SOURCING AND ITS ADVANTAGES

Event sourcing is quite suitable and has a robust architectural pattern where an application state is captured as a series of events. Unlike simple saving of the layout of the database, all cooperations are saved in a transaction log, which is strictly write-only. This approach has very many merits, as shown below. First, it maintains the full history of each transaction, which substantially reduces the time necessary for debugging and, if necessary, auditing. State-based software is advantageous to developers as it allows them to follow the event history leading to a particular state. Also, event sourcing improves scalability. Every application creates events, and as it grows, it can replay events to construct its current state without involving frequent changes in databases or schemas. In addition, this technique enhances the system's capability to hide faults compared to previous approaches, thus enhancing the achievement of better fault tolerance. Since every event is saved faithfully, coming up with states after crashes turns out to be easy and effective. Adopting event sourcing makes it easier for developers to create sound applications and maintain them during their applications' lifespan (Varia, 2011).

IX. CHALLENGES AND SOLUTIONS

When used in event-driven architecture, several problems can be encountered, including integration problems. This can be a specific challenge of most push systems: guaranteeing the transmission of messages. Especially if there are no efficient tools, messages can be missed or arrive too late. In order to overcome this problem, Amazon SQS can be used for queueing and Amazon SNS for notification improvements. For these services to operate, they make it possible for messages to be held until the target consumer processes the message. A third issue is the management of event schema changes. While this is not impossible, it is usually easier during the

design and development phase of an event schema. With applications evolving and transforming over time, preserving backwards compatibility becomes highly important to avoid disrupting existing integrations. When dealing with change, a couple of things to consider: using versioning for events manages changes gracefully. This enables different versions of consumers to co-exist while in the same process, step by step, transforming them into new schemas. Evaluation of events and debugging is also challenging in composite event-driven systems. Using tools such as AWS CloudWatch gives an understanding of how the system behaves and allows you to trace issues through logs and metrics at your disposal (Sbarski&Kroonenburg, 2017).

X. CONCLUSION

Event-driven architecture is now an important element of software development practice. This being the case, its potential to improve responsivity and scalability makes it quite appealing to many developers. Multiplied by Java and AWS, it becomes significantly more beneficial. These two allow applications to efficiently and effectively organize, process and minimize the time taken to address events as they occur. Each flexible element, EventBridge, SQS, and SNS, adds much value. Besides that, event sourcing has advantages for the same applications by giving them clarity and traceability. This means that developers can easily reconstruct previous states and simplify various debugging processes. This approach leads to better architectural solutions that are resilient when the demands start shifting. So, as organizations continue to extol their digital transformation journey, these technologies will be critical to compete in today's dynamic world (Varia, 2011).

REFERENCE

1. Bermudez, I., Traverso, S., Munafo, M., &Mellia, M. (2014). A distributed architecture for the monitoring of clouds and CDNs: Applications to Amazon AWS. IEEE Transactions on Network and Service Management, 11(4), 516-529.
2. Clark, T., & Barn, B. S. (2012, February). A common basis for modelling service-oriented and event-driven architecture. In Proceedings of the 5th India Software Engineering Conference (pp. 23-32).
3. Hofmann, M., Schnabel, E., & Stanley, K. (2017). Microservices Best Practices for Java. IBM Redbooks.
4. Kim, M., &Wellings, A. (2010). Efficient asynchronous event handling in the real-time specification for java. ACM Transactions on Embedded Computing Systems (TECS), 10(1), 1-34.
5. Poornalinga, K. S., & Rajkumar, P. (2016). Continuous integration, deployment and delivery automation in AWS cloud infrastructure. Int. Res. J. Eng. Technol.
6. Sbarski, P., &Kroonenburg, S. (2017). Serverless architectures on AWS: with examples using Aws Lambda. Simon and Schuster.
7. Varia, J. (2011). Best practices in architecting cloud applications in the AWS cloud. Cloud Computing: Principles and Paradigms, 457-490. And book. Mill Valley, CA: University Science, 1989.