Harnessing Event Driven Cloud Architecture for Optimized System Design

Akshat Bhatt
Research in Computer and Systems Engineering
Technische Universitat Ilmenau
Ilmenau, Thuringia, Germany
akshat.bhatt@tu-ilmenau.de

***Abstract*—An event driven approach (EDA) is a means of linking components in a decentralized, loosely-affiliated arrangement. This paper examines the aptitude of event oriented systems to enhance system efficiency by means of use-case and cloud-based application approaches.**

***Index Terms*—Event Driven Architecture, Cloud Computing, System Design.**

I. Introduction

The main advantage of event driven architecture (EDA) is that it can be used to build highly distributed, resilient applications. This gives EDA an advantage over traditional systems that might be tightly coupled[3].

Since cloud technologies provide cost savings, scalable, and flexible solutions this allows organizations to leverage resources on-demand and handle large volumes of data and events[5]. When EDA-driven solutions leverage the advantages of modern cloud providers the infrastructures that are created as a result can respond to events quickly, enabling organizations respond to changing customer demands and market conditions.

To explain how EDA can be used this paper will take a use case of a synchronous payment gateway system that uses a Service Oriented Architecture to create a payment gateway. Then will discuss the problems with that approach and create an EDA based setup to mitigate the problems. Finally, will discuss how the proposed solution can be implemented using AWS cloud provider services and technologies.

This study mainly makes use of the AWS services to build an EDA-based solution. However, most of the other modern cloud providers have analogous services, although they go by different names.

II. Backgroud

A. Use-case

In modern system design patterns an efficient architecture is the one which is most independent and stable[1]. To make a system independent and self sufficient, it should be able to collect, analyze, decide, and then act based on the input factors[2]. Let's consider an example to explain an event driven system and how it fits the above definition of a highly efficient system. Traditional payment processing platforms typically employ a sales service that calls both the billing and warehousing services. The billing service will generate

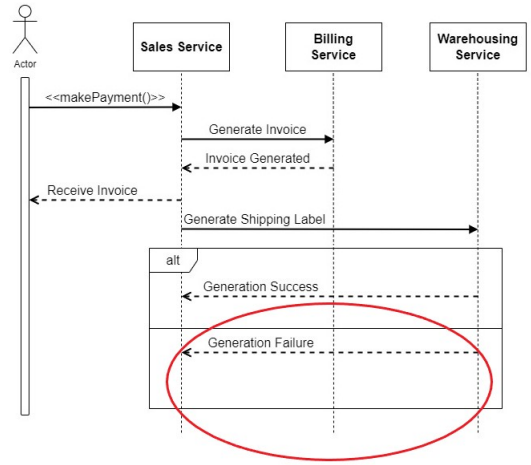

Fig. 1. legacy payment portal workflow.

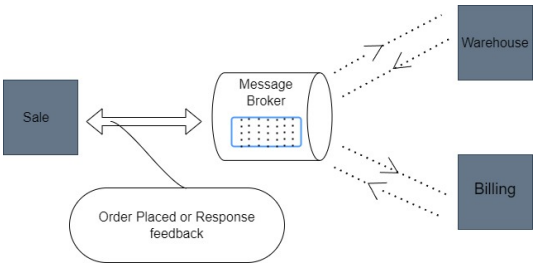

Fig. 2. Event driven design.

a receipt once payment is accepted from the customer, which is then relayed to the customer. The sales service will then call the warehousing service, which will verify the availability of products and create the associated shipping label. The shipping information is then sent to the customer.

However, since all these calls are made sequentially therefore issues may arise in the event of a failure of the warehousing service, as illustrated in 'Fig. 1'. This creates a challenging situation where the customer must be informed of the reversed payment invoice and other related compensatory actions. In similar cases like above where there is a blocking synchronous request-response model between different temporarily coupled services like gRPC and HTTP, this issue is frequent.

To circumvent this issue, we can isolate each service into consideration as an autonomous system. To accomplish this, a message broker is set up which is linked to all pertinent services. This message broker internally has a queue which retrieves message tokens or events from the sales service. The warehousing and billing service are then subscribed to the broker and all messages that are delivered to the broker queue are read by the warehouse and billing services. Both the warehouse and billing service can then determine how to respond according to these messages. When the broker obtains a message indicating that a payment has been made by the customer, the warehousing and billing services are notified accordingly. Subsequently, both components act in unison to execute their duties and send a confirmation packet back to the broker. The broker then passes the message to the sales service in the event of a successful execution, which in turn conveys the response package to the customer. Refer Fig-2 for an illustration of the proposed event driven solution.

III. Methodology

event driven architecture on the AWS cloud is a powerful way to process and act on data in real-time. It is a distributed architecture that enables applications to respond quickly to events and triggers, such as a customer's order, a stock market event, or a new user sign-up.

AWS provides a suite of services for building event driven architectures. These services include Amazon Simple Queue Service (SQS) for message management, Amazon DynamoDB for resilient and fast data storage, Amazon Kinesis for data streaming, Amazon EventBridge for event routing, and Amazon Lambda for server-less computing. For the example use-case we have considered before we will be using a combination of SNS, SQS and Lambda for leveraging event driven functionality. It is to be noted that most cloud providers have other alternative names for services with similar functionality like Azure Queue Storage for message management, Azure CosmosDB for resilient and fast data storage, Azure Stream Analytics for data streaming, Azure Event Grid for event routing, Azure Functions for server-less computing.

A. Working Implementation :

Consider 'Fig. 3' for illustration of the working implementation.

- `SNS` SNS is a web service that makes it easy to set up, operate, and send notifications from the cloud. It provides developers with a highly scalable, flexible, and cost-effective capability to publish messages from an application and immediately deliver them to subscribers or other applications. Here the sales service will put messages in the queue of the related topic.
- `SQS` SQS is a managed message queue service that can be used to decouple applications and components. It enables applications to send and receive messages and scale automatically with traffic. Here SQS will gather data from the SNS so in case there is a failure in either warehousing or billing service their messages will sit in their SQS queue respectively.
- `AWS Lambda` AWS Lambda is a serverless computing service[4] that makes it easy for developers to run code without provisioning or managing servers. Lambda functions can be triggered by events and used to process data or execute tasks in response to those events. Lambda functions will be used by the warehousing and billing service to pull data from the SNS topic, batch processing or pre-processing requirements.

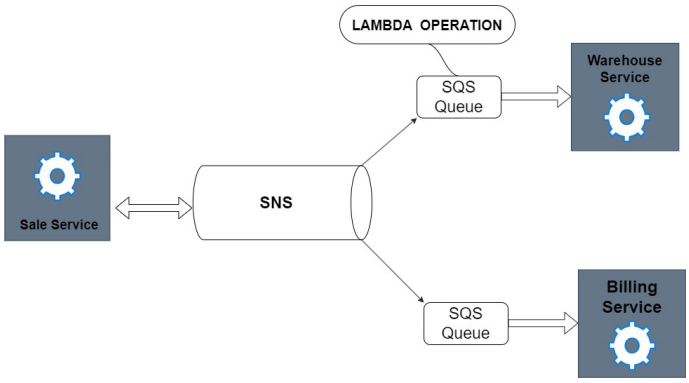

Fig. 3. AWS EDA based payment portal workflow.

IV. Conclusion

This paper elucidated the utility of employing an Event-Driven Architecture (EDA) to conceptualize an idempotent decoupled system. Thereafter, presented a use-case to exhibit its effectiveness. Additionally, it delved into the mechanism of implementing an event-driven system through the AWS Cloud.

References

[1] Mark W Maier. "Architecting principles for systems-of-systems". In: *Systems Engineering: The Journal of the International Council on Systems Engineering* 1.4 (1998), pp. 267–284.

[2] Betty HC Cheng et al. "08031–Software engineering for self-adaptive systems: A research road map". In: *Dagstuhl Seminar Proceedings*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik. 2008.

[3] Tony Clark and Balbir S Barn. "Event driven architecture modelling and simulation". In: *Proceedings of 2011 IEEE 6th International Symposium on Service Oriented System (SOSE)*. IEEE. 2011, pp. 43–54.

[4] R. Arokia Paul Rajan. "Serverless Architecture - A Revolution in Cloud Computing". In: *2018 Tenth International Conference on Advanced Computing (ICoAC)*. 2018, pp. 88–93. DOI: 10.1109/ICoAC44903.2018.8939081.

[5] Hukum Saini, Abhay Upadhyaya, and Manish Kumar Khandelwal. "Benefits of cloud computing for business enterprises: A review". In: *Proceedings of International Conference on Advancements in Computing & Management (ICACM)*. 2019.