JETIR.ORG ISSN: 2349-5162 | ESTD Year : 2014 | Monthly Issue
JOURNAL OF EMERGING TECHNOLOGIES AND INNOVATIVE RESEARCH (JETIR)
An International Scholarly Open Access, Peer-reviewed, Refereed Journal

Building Highly Scalable Systems: Harnessing the power of Eventstore Model with DynamoDB (or ScyllaDB) with Kafka

Abhinav Parhad, Jharna Chopra
[1]Research Scholar, [2]Associate Professor

Abstract: This report provides a detailed overview of a method for building systems that can grow bigger and handle lots of requests. It uses event sourcing with DynamoDB (or ScyllaDB), Kafka, and compares it to REST architecture. The report explains the basic ideas behind event sourcing and how it helps make systems scalable and durable. It also talks about the steps to follow when using event sourcing, like designing the system structure using DynamoDB for storing events and Kafka for sending events around, and setting up Event Store to manage events. The report discusses the benefits of using event sourcing over REST making systems more flexible, and handling complex tasks. It also mentions the importance of replaying events, making sure everything stays consistent, and handling events in real-time. The report ends by talking about the need to test and monitor system performance and make continuous improvements to keep the system scalable. It is a useful guide for researchers who want to use event sourcing with DynamoDB, Kafka, and Event Store to build highly scalable systems.

IndexTerms *– Event Sourcing, CQRS*

I. INTRODUCTION

Scalability is a vital component of modern software devices, especially with the exponential growth of data & user demands. Building highly scalable systems has become a necessity for organizations to handle large volumes of data, provide responsive services, and adapt to changing requirements. Traditional approaches often struggle to meet scalability requirements, leading to performance bottlenecks and system failures. Therefore, alternative architectural models are needed to overcome these challenges. This dissertation focuses on the event sourcing model as a potential solution for building highly scalable systems. One such pattern is event sourcing, which has gained significant popularity in the software development community due to its suitability for building highly scalable and resilient systems. Event sourcing is a design that captures & persists all changes to an application's state as a series of immutable activities. Instead of storing only the current state, event sourcing stores a historical log of events that have occurred. The motivation behind using event sourcing in architecture lies in its ability to provide a comprehensive audit trail, temporal queries, and support for complex business workflows some of which lacks in REST. With event sourcing, organizations can track and analyze every state change in their systems, making it easier to meet compliance requirements and perform forensic analysis. The ability to perform temporal queries enables historical analysis and facilitates the diagnosis and resolution of issues by replaying events. Moreover, event sourcing accommodates complex business workflows by capturing domain-specific events and allowing the reconstruction of application state based on those events.

II. LITERATURE REVIEW

Shree et al.,(2017) Apache Kafka's ability to be utilized as a cluster on any set of servers is one of its fundamental concepts. Record streams are stored in classes called topics by the Kafka server. Each record includes a time stamp, a value, and a key. It has two different application classes. First, to construct trustworthy real-time data stream pipelines for transferring data across apps or between systems. Second, create real-time streaming software that responds to record streams. Hundreds of megabytes of reads & writes per second from thousands of clients may be handled by a single Kafka mediator. A single cluster can act as the main data hub for a sizable organization thanks to Scalable Kafka.

Kul et al.,(2021) examined the issue of publishing and subscribing-based key-value queries for retrieving video chunks. So, for the Event-Based Micro service structure, we suggest a mixture of an asynchronous or synchronous interconnection method. Our goal

is to generate general methods for maximizing the use of current stages. Experimental findings demonstrate that suggested approach filters messages in real-time & facilitates simple system integration of additional micro services.

Benjamin et al.,(2017) offered methods for event-sourced applications' global or retrospective state extraction based on distributed event logs. With regard to distributed debugging & break pointing, which are closely related to event log-based state reconstruction, authors give an overview of earlier methods. Then, authors present and assess our method for extracting non-local state from distributed event logs that has been specially tailored for dynamic & asynchronous event-sourced devices.

Lima et al.,(2021) claimed that an event log is insufficient for a thorough audit of a distributed device. Even though ES has numerous benefits in means of replay ability, isolation, information approach adaptability, it still only gives a partial picture of the system when it comes to debugging. Debugging typically entails analyzing the pertinent component, going through the costly method of reproducing the events, & gathering the metadata. Instead, authors improved device visibility, which eliminates the requirement to replay the event log. Several practical problems that come up while using and debugging the ES system are resolved by suggested proposal. The fact that tracing is generally acknowledged in the business and thus adds little expense to the ES system makes it a great addition to ES. It is frequently previously established, as well as the necessary tools and guidelines are available. In addition, it is relatively cheap to add instrumentation points among communication since ES imposes an especially coherent design style consisting of separate elements interacting through the event log.

Vasconcellos et al.,(2018) the topic of Event Sourcing, an architectural approach that arranges software's classes to give a native memory of its past states, allowing the device to even replay previous calculations at will, is discussed. Authors also include a case study of how this design was used to create an ERP system. The development team found the design to be especially helpful because it significantly improved mistake traceability.

III. OBJECTIVES

The proposed work aims to achieve two primary objectives:

- To explore the idea on how Event sourcing using Dynamodb (or Scylla DB) with Kafka can be used for making a highly scalable architecture.
- To implementation an Event store and assess its benefits.

IV. METHODOLOGY

The methodology for the proposed work outlines the steps for building highly scalable systems using event sourcing with DynamoDB, Kafka, and Event Store, along with a comparison to RESTful architecture. It begins by understanding event sourcing and designing the system architecture, including the integration of DynamoDB and Kafka. The implementation of Event Store involves defining the schema, storing and retrieving events, and ensuring durability. The comparison with REST emphasizes factors like scalability; fault tolerance, real-time processing, and system rebuild capabilities. Performance and scalability testing, monitoring, fine-tuning, and iterative development are essential for optimizing the system. The methodology emphasizes the need for continuous learning and staying updated with the latest advancements in event sourcing technologies. Figure 1 illustrates the steps involved in the proposed methodology for this work.

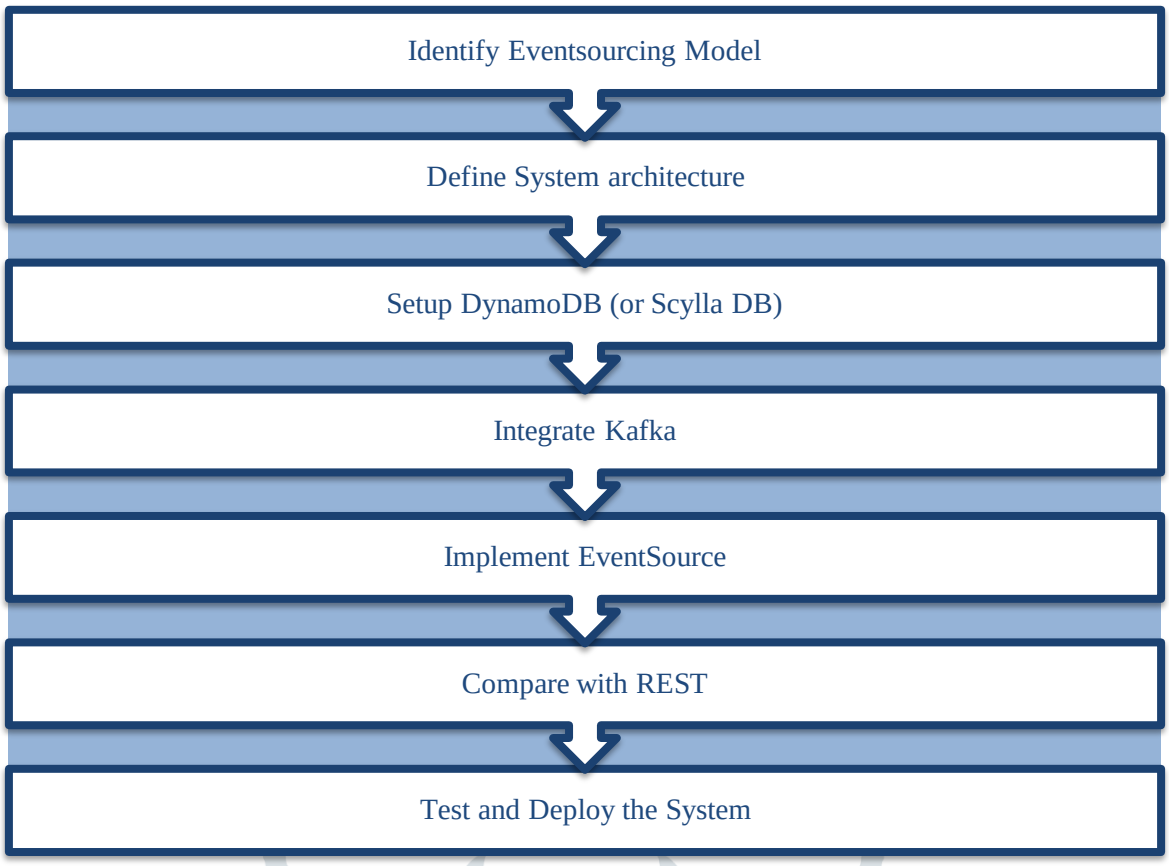

Figure 1: Steps for Methodology

V. RESULTS AND DISCUSSION

Algorithms Used

CQRS, which stands for Command and Query Responsibility Segregation, is a pattern that separates read and update operations in a data store. By implementing CQRS, you can enhance your application's performance, scalability, and security. This approach provides flexibility for system evolution and prevents conflicts during domain-level updates. CQRS separates reads and writes into different models, utilizing commands for data updates and queries for data retrieval.

Commands in CQRS should be task-based rather than data-centric, enabling better understanding and clarity (e.g., "Book hotel room" instead of "set ReservationStatus to Reserved"). Commands can be asynchronously processed by placing them on a queue, allowing for more efficient execution compared to synchronous processing. On the other hand, queries in CQRS are read-only operations and return DTOs (Data Transfer Objects) without domain knowledge.

While isolating the models, as depicted in the diagram, is a common practice in CQRS, it is not mandatory. The benefits of implementing CQRS include independent scaling of read and write workloads, optimized data schemas for efficient querying and updating, enhanced security by controlling write access to domain entities, separation of concerns leading to maintainable and flexible models, and simplified queries by utilizing materialized views in the read database.

Event Store Algorithm

Event Store is a database technology designed specifically for event sourcing, which is a pattern used in software development to capture and store changes to an application's state as a sequence of events. Event sourcing focuses on storing the events that have occurred in an application, and not just the current state of the data. Event Store provides the capability to store and retrieve events in an efficient and scalable manner. It allows applications to persist events, query them, and replay them to reconstruct the application's state at any point in time. Event Store ensures that events are stored in an immutable and append-only fashion, preserving the history of changes.

By using Event Store, developers can build event-driven architectures that enable event sourcing and event-driven communication between different components or services. This approach provides benefits such as auditing, temporal queries, versioning, and flexibility in handling complex business workflows.

Amazon Web Services (AWS) EC2 machines are used to power these tests. Kafka - m5.large which contains 2 vCPUs and 8 GiB Memory with a network speed of up to 10 Gbps with 3 nodes and 3 replication factor with 5 partitions. Scylla – i3.large which

contains 2 vCPUs and 15.25 GiB Memory and 1 x 475 NVMe SSD with a network speed of up to 10 Gpbs with 3 nodes and 3 replication factor.

Event Store Algorithm

1) Write to Event Store Algorithm

- Create a Kafka producer with the specified Kafka brokers.
- Produce an event message to the specified Kafka topic using the producer.
- Wait for the message delivery report to ensure successful delivery.

2) Read from Event Store Algorithm

- Create a session with the specified AWS region.
- Create a Dynamo DB client using the AWS session.
- Define a query input with the specified DynamoDB table name and a time range condition.
- Execute the query against DynamoDB and retrieve the result.
- Iterate over the retrieved items and deserialize each item into an event object using the deserialization logic.

3) Helper function to de-serialize event data

- Implement the logic to deserialize the event data from the DynamoDB item format to the desired event object format.

4) Main function

- Define the event data to be written to the event store.
- Call the "Write to Event Store" algorithm to write the event data.
- Specify the start time and end time for the desired time range.
- Call the "Read from Event Store" algorithm with the specified time range to retrieve events.

EventStore Consumer Algorithm

- Create a Kafka consumer with the specified Kafka brokers and consumer group.
- Subscribe to the desired Kafka topic(s) for event consumption.
- Start an infinite loop to continuously consume events from Kafka.
- Within each iteration, poll for new messages from Kafka.
- Process each received message by extracting the event data and handling it accordingly.
- Commit the offset of the processed message to Kafka to ensure it is marked as consumed.
- Handle any errors that occur during the event processing.
- Write to the event to DynamoDB.

Throughput

Throughput measures the rate at which events are processed or written to the event store within a given time period.

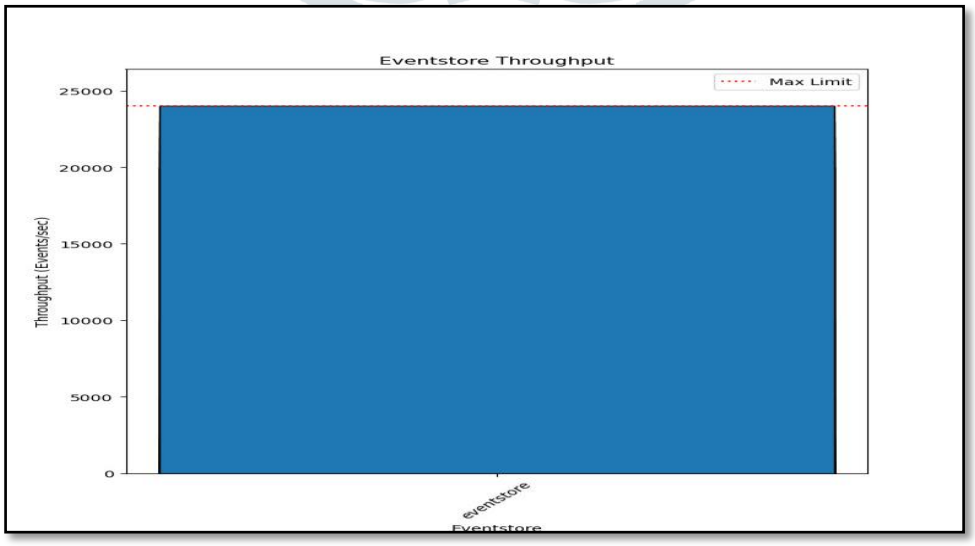

Figure 2: Throughput

Figure 2 for throughput typically shows the number of events processed or written per unit of time (e.g., events per second or events per minute). It helps to assess the system's ability to handle event ingestion or processing load.

Latency

Latency represents the time taken for an event to be successfully processed or written to the event store, measured from the time it was generated or received.

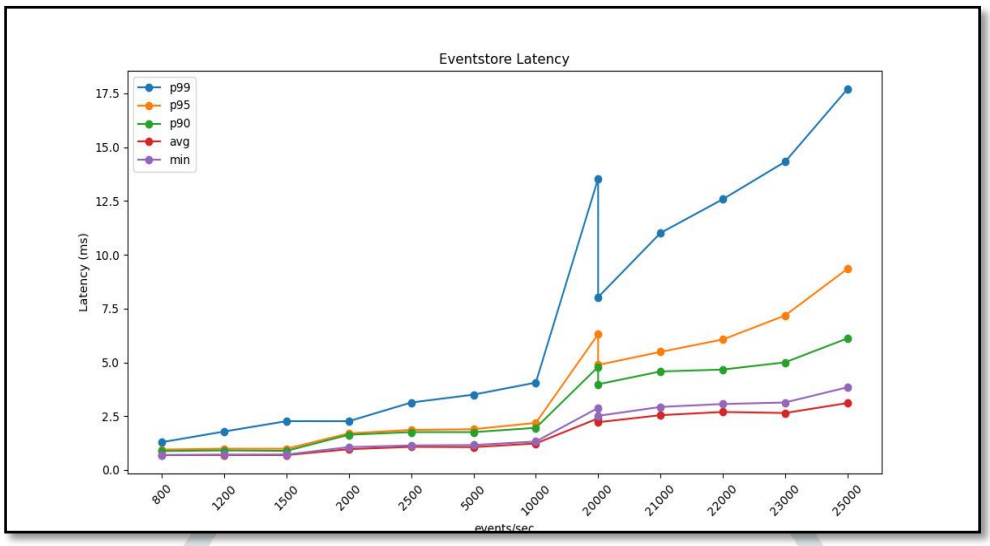

Figure 3: Latency

The latency graph (figure 3) illustrates the distribution of event processing or writes latencies. It often displays metrics such as average latency, percentiles (e.g., 90th or 99th percentile), or maximum latency. Monitoring latency helps ensure that the system meets the required response time or service level agreements.

V. CONCLUSION

This research work emphasizes the value of using Event sourcing and Event store technologies to build scalable architectures, showing their benefits compared to traditional REST APIs. By implementing an Event store, the benefits of this approach were compared to traditional REST architecture. The exploration showed that Event sourcing, along with DynamoDB (or Scylla DB) and Kafka, has great potential for creating a highly scalable system. Additionally, implementing an Event store offered several advantages over the usual REST approach, highlighting the importance of event-driven systems for improved scalability and flexibility. The report explores using Event sourcing with DynamoDB (or Scylla DB) and Kafka for a scalable system. Throughput measures event processing rate. Latency graph illustrates event processing distribution. Monitoring latency ensures desired response time. Error rate identifies issues, storage utilization tracks growth, and resource utilization optimizes system operation.

REFERENCES

[1]. Sharafaldin, I.; Lashkari, A.H.; Ghorbani, A. Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization; Canadian Institute for Cybersecurity (CIC): Fredericton, NB, Canada, 2018; pp. 108–116.

[2]. Chen, L.; Kuang, X.; Xu, A.; Suo, S.; Yang, Y. A Novel Network Intrusion Detection System Based on CNN. In Proceedings of the 2020 Eighth International Conference on Advanced Cloud and Big Data (CBD), Taiyuan, China, 5–6 December 2020; pp. 243–247. [CrossRef]

[3]. Gautam, R.K.S.; Doegar, E.A. An Ensemble Approach for Intrusion Detection System Using Machine Learning Algorithms. In Proceedings of the 2018 8th International Conference on Cloud Computing, Data Science & Engineering (Confluence), Noida, India, 11–12 January 2018; pp. 14–15. [CrossRef]

[4]. Al-Yaseen, W.L.; Othman, Z.A.; Nazri, M.Z.A. Multi-level hybrid support vector machine and extreme learning machine based on modified K-means for intrusion detection system. Expert Syst. Appl. 2017, 67, 296–303. [CrossRef]

[5]. Kanimozhi, P.; Victoire, T.A.A. Oppositional tunicate fuzzy C-means algorithm and logistic regression for intrusion detection on cloud. Concurr. Comput. Pract. Exp. 2022, 34, e6624. [CrossRef]

[6]. Chen, Y.; Yuan, F. Dynamic detection of malicious intrusion in wireless network based on improved random forest algorithm. In Proceedings of the 2022 IEEE Asia-Pacific Conference on Image Processing, Electronics and Computers (IPEC), Dalian, China, 14–16 April 2022; pp. 27–32. [CrossRef]

[7]. Jabez, J.; Muthukumar, B. Intrusion Detection System (IDS): Anomaly Detection Using Outlier Detection Approach. Procedia Comput. Sci. 2015, 48, 338–346. [CrossRef]

[8]. Kurniawan, Y.; Razi, F.; Nofiyati, N.; Wijayanto, B.; Hidayat, M. Naive Bayes modification for intrusion detection system classification with zero probability. Bull. Electr. Eng. Inform. 2021, 10, 2751–2758. [CrossRef]

[9]. Chauhan, N. Naïve Bayes Algorithm: Everything You Need to Know. Available online: https://www.kdnuggets.com/2020/06/ naive-bayes-algorithm-everything.html#:~{}:text=One%20of%20the%20disadvantages%20of,all%20the%20probabilities%20 are%20multiplied (accessed on 9 September 2022).

[10]. Gu, J.; Lu, S. An effective intrusion detection approach using SVM with naïve Bayes feature embedding. Comput. Secur. 2021, 103, 102158. [CrossRef]