IJARSCT

International Journal of Advanced Research in Science, Communication and Technology

International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal

ISSN: 2581-9429

Volume 5, Issue 4, April 2025

Impact Factor: 7.67

AWS Resilient Pattern: Implementing Change Data Capture with Reduced Data Loss

Jhansi Xavier and Praveen Borra

Computer Science, Florida Atlantic University, Boca Raton USA

Abstract: *This paper examines a proposed architecture pattern that has been evaluated to minimize data loss rates in AWS. Change Data Capture (CDC) is a methodology that detects and monitors alterations in a database, facilitating real-time or near-real-time data replication and synchronization among systems. When a CDC process is implemented within a database system, an effective failover mechanism is crucial to ensure the ongoing capture and transmission of changes. It is anticipated that some data loss may occur during the failover process. This article aims to explore the pattern designed to mitigate such data loss.*

Keywords: AWS, Distributed systems, Change Data Capture, Message Queue, Kafka, Failover, Data Loss, Hybrid Environment, Performance Optimization, System Resilience, Fault Tolerance

I. INTRODUCTION

Change Data Capture:

Change Data Capture (CDC) has become an optimal method for facilitating near real-time data transfer from relational databases, such as SQL Server or Oracle, to data warehouses, data lakes, or other database systems. CDC is a technique that detects and records modifications—such as inserts, updates, and deletions—occurring in a source database. This process is essential for ensuring data consistency across various platforms, including databases, applications, and data warehouses.

- CDC is a software design pattern employed to identify and handle incremental modifications at the data source. It records changes—such as inserts, updates, and deletions—as they occur, enabling data consumers to respond immediately.
- CDC serves as a technique within the ETL (Extract, Transform, Load) framework. This approach is especially beneficial for maintaining synchronization between systems, ensuring dependable data replication, supporting real-time analytics, and enabling database migrations without downtime.

Data Loss in CDC:

Data loss poses a considerable risk in change data capture implementations, which can result in interruptions in information flow. To address this challenge, organizations should establish a Failover system that consistently oversees data ingestion and processing pipelines to detect any irregularities that could lead to data loss. Additionally, this system must incorporate strong error handling procedures and ensure effective synchronization between source and destination systems. Proper synchronization is crucial in averting incidents of data loss.

Reasons of Data Loss in CDC:

- Inability to detect modifications: Should the Change Data Capture (CDC) process be unable to identify or record alterations in the source database, those modifications will not be transferred to the target systems.
- Connectivity problems: Disruptions in the connection between the source and target systems may result in data being lost or duplicated.
- Unavailability of the target system: If the target system is not operational or accessible during the CDC process's attempt to replicate changes, those changes might be lost or held in a buffer until the system becomes available again, which could result in data loss or duplication.

Impacts of Data loss in CDC:

- Inconsistent Reporting: Failure to replicate changes can result in discrepancies across various systems, causing errors and misguided decisions.
- Data Loss: The complete loss of changes may result in the absence of vital data, which can create operational challenges.
- Interrupted Business Operations: Inconsistencies and data loss can interfere with business processes and adversely affect decision-making.

Prevent data loss in CDC:

- Effective CDC Implementation: Confirm that the CDC process is properly set up and continuously monitored to accurately record all changes.
- Connection Stability: Establish strong protocols to guarantee dependable connections between the source and target systems.
- Target System Accessibility: Ensure that target systems are consistently accessible and that there are strategies in place to manage temporary disruptions.
- Data Backup and Recovery: Develop and enforce backup and recovery protocols to safeguard against potential data loss.

Resilient Design Patteren for CDC-Data Loss:

1. Database:

It functions as the data repository, which can be categorized as either document based storage or relational database storage. Modifications to an existing table or document, such as inserts, updates, or deletions, initiate a Change Data Capture (CDC) event. For instance, in DocumentDB, there is an option that can be activated to capture this CDC event. This event includes metadata along with the changes in the document's state, indicating whether it has been inserted, updated, or deleted.

2. Producer Service:

The Producer Service will handle the retrieval of CDC events from the database. It gathers changes and disseminates the CDC event to the designated Queue or Kafka topic. This service can be developed using serverless Lambda, facilitating easier maintenance.

3. Document Queue/Kafka topic:

This middleware stream functions as the primary event broker. The Queue/topic retains events that are published by the Producer Service. It enables event-driven communication and promotes decoupling between the Database and downstream services. Additionally, it stores Change Data Capture (CDC) events from the database for later consumption. Furthermore, it guarantees message durability and resilience against faults.

4. Consumer Service:

This service functions as an intermediary for transmitting CDC events to the Data Lake or Secondary database, which serves as a replica. Additionally, it facilitates synchronous transmission and buffering by managing the flow of messages to the target system. It has the capability to batch or throttle messages as required.

5. Fault Tolerance Queue/Topic:

This Queue/Kafka topic is designated for the collection and storage of information regarding errors and faults that occur during the CDC event processing. In instances where the Producer Service is unable to address an error right away, the fault is sent to the Fault Tolerance Queue/Topic. This design enables the architecture to log and manage these faults independently, ensuring that the primary data flow remains uninterrupted.

6. Fault Handler Service:
A specialized Service/Lambda function serves as a consumer for the Fault Queue/Topic. This function is tasked with managing error events, which includes retrying unsuccessful operations or executing designated error-handling workflows. The architecture distinctly separates error management from the primary event flow.
Additionally, it may initiate notifications to monitoring systems or prompt manual interventions.

7. Dead Letter Queue (DLQ) Topic:
Should an event fail to be processed successfully after a specified number of attempts, it will be directed to this Queue/Topic. This mechanism allows for the examination and subsequent processing of problematic events without hindering or endlessly retrying within the main data flow, thereby improving the overall resilience of the system. It guarantees that no data is lost and that issues can be addressed manually or rectified for future processing.

Advantages:

- The Producer and Consumer must be set up with retry mechanisms to manage temporary failures in Queue/Kafka.
- Errors that cannot be resolved through immediate retries are directed to the Fault Queue, allowing them to be processed separately without disrupting the primary event flow.
- For ongoing failures, events are forwarded to the Dead Letter Queue (DLQ). The DLQ facilitates the examination and future processing of problematic events, preventing them from being lost or subjected to endless retries.
- This Fault Handler Service makes attempts to retry failed operations or address routing issues for manual intervention, thereby ensuring effective fault isolation and averting cascading failures.

Future Considerations

- AWS Lambda can be utilized to implement producer and consumer services, offering a serverless solution that is easy to maintain. Furthermore, the same producer/consumer configuration can be reused across different systems for similar functions.
- In distributed systems, a failover mechanism can be established as a pattern to improve asynchronous communication between the systems. The integration of Kafka or message queues will facilitate better management of this process.
- The system will automatically scale in response to the volume of Change Data Capture (CDC) events originating from the database. Additionally, it is feasible to link the scaling capabilities to the number of messages present in the Kafka topic.

II. CONCLUSION

This design offers a strong framework for managing Change Data Capture (CDC) modifications within a database and processing them via an event-driven architecture for real-time analytics. By carefully selecting and implementing resources, this design guarantees scalability, fault tolerance, and a clear division of responsibilities. It is particularly effective for data-centric applications that demand high throughput and reduced coupling between components.

REFERENCES

[1]. R. Kreps, "Managing Fault Tolerance and High Availability in Apache Kafka," ACM Transactions on Information Systems, vol. 38, no. 4, pp. 15-28, Dec. 2020.
[2]. Hemadri Lekkala, "An Implementation of Change Data Capture (CDC)," International Journal of Computer Trends and Technology, Volume 71 Issue 2, 15-18, February 2023.
[3]. Liang Hao, Tao Jiang, "Methods for Solving the Change Data Capture Problem", Advances in Natural Computation, Fuzzy Systems and Knowledge Discovery (pp.781- 788).

IJARSCT

International Journal of Advanced Research in Science, Communication and Technology

International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal

ISSN: 2581-9429

Volume 5, Issue 4, April 2025

Impact Factor: 7.67

[4]. Ramasankar Molleti. (2024). Ensuring Optimal Performance and Resilience for the Kafka Platform in A Hybrid Environment. International Journal of Intelligent Systems and Applications in Engineering, 12(1), 829.

[5]. S. Vasudevan, P. Bhat, and M. Shenoy, "Enhancing Data Stream Processing in Hybrid Cloud Environments with Apache Kafka," Journal of Cloud Computing, vol. 9, no. 3, pp. 150-162, Sep. 2019.