International Journal of Science and Research (IJSR)
ISSN: 2319-7064
SJIF (2022): 7.942

Leveraging Event - Based Architecture, AWS Step Functions, AWS Batch, and DynamoDB to Run ETL or ELT Jobs Concurrently While Allowing Granular Replay Capabilities

Akshay Prabhu

Abstract: ***Traditional Extract, Transform, Load (ETL) and Extract, Load, Transform (ELT) jobs are often perceived as hardware - intensive, necessitating the use of persistent EC2 instances to handle large data sets. This conventional approach presents challenges, including the need for manual monitoring of long - running jobs and the inability to replay jobs from specific points or stages in the ETL/ELT process. Additionally, the intricate nature of ETL/ELT phases, each with potential failure points, complicates the operational management of these workflows. AWS provides a suite of serverless services such as EventBridge, S3, SNS, Lambda, Step Functions, and Batch that can be leveraged to create a scalable and resilient ETL/ELT architecture. This paper explores how integrating these services can transform traditional ETL/ELT processes into a more flexible, state - managed saga (1) with granular replay capabilities. The goal is to offer insights into how this architecture using the above - mentioned AWS services can enhance traditional data processing workflows, focusing on concurrent job execution and precise error recovery, especially targeted for Software Architects and Engineers.***

Keywords: Event - Based Architecture, AWS Step Functions, AWS Batch, Amazon, DynamoDB, ETL (Extract, Transform, Load), ELT (Extract, Load, Transform), Serverless Computing

1. Introduction

AWS offers a diverse set of serverless services that can be orchestrated to streamline ETL/ELT workflows.

AWS Step Functions provide a robust mechanism for coordinating complex workflows by defining a state machine that governs the sequence and conditions under which tasks are executed. This service integrates seamlessly with other AWS components, allowing for sophisticated error handling and retry strategies. (2)

AWS Batch is designed to handle large - scale batch processing, enabling users to run thousands of batch computing jobs with dynamic resource allocation. It integrates with AWS Step Functions to execute jobs in a scalable and cost - efficient manner. (3)

AWS Lambda is a serverless compute service that runs code in response to triggers, such as changes to data in S3 or messages in SNS. It is ideal for executing lightweight tasks and processing events in real - time. Lambda functions can be used to transform data or trigger workflows defined in Step Functions, providing a flexible and scalable approach to ETL/ELT processes. (4)

Amazon EventBridge (formerly CloudWatch Events) is a serverless event bus service that facilitates the creation of event - driven architectures by routing events from various sources to target services. It can trigger AWS Lambda functions or Step Functions based on specific events, such as file uploads to S3 or messages published to SNS. (5)

Amazon S3 provides scalable storage for raw and processed data, while Amazon SNS enables message - based communication between different components of the workflow.

Amazon DynamoDB, a fully managed NoSQL database, offers high availability and performance for storing metadata and job state information. Its integration with EventBridge and Lambda supports real - time event - driven processing capabilities. (6)

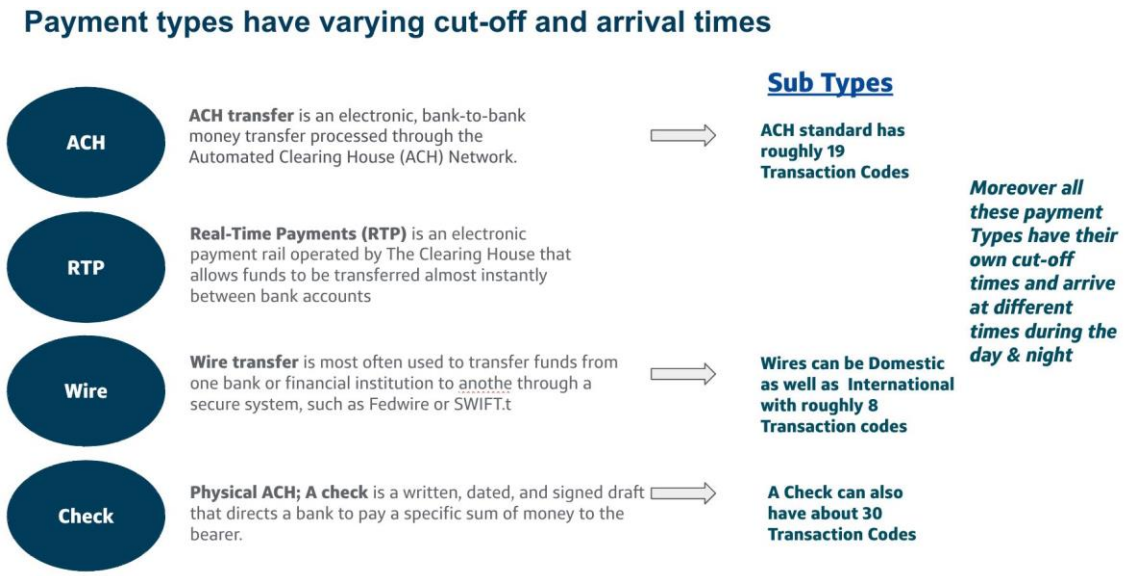
By leveraging these AWS services in harmony, organizations can build a scalable, resilient, and flexible ETL/ELT architecture that addresses the limitations of traditional methods and provides advanced capabilities for managing complex data processing tasks.

Problem Statement

Consider a payment processing use case where various payment files need to be ingested and processed through a serverless ETL pipeline.

Payments, as we all know, are all around us, and as consumers, we care about viewing our end payments as whole transactions with all relevant metadata.

However, payment files are among the most complex, and each underlying transaction/payment needs to be constructed using various data points and fields. In a typical payment ingestion scenario, files may come from multiple sources and formats.

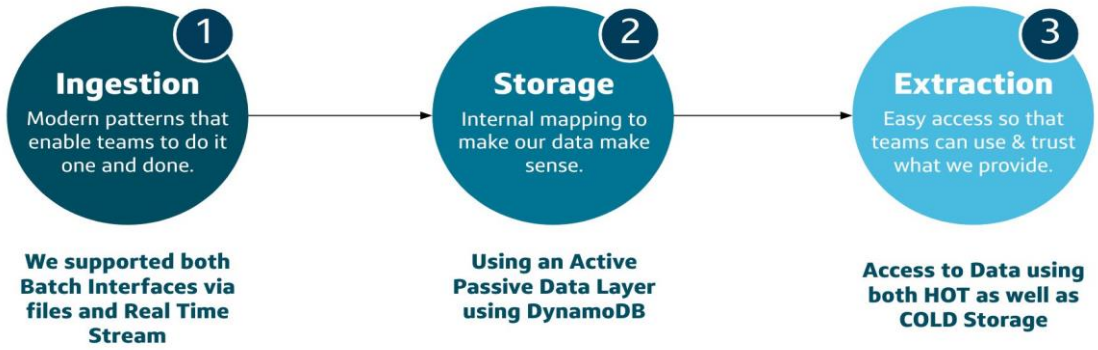


Traditional ETL approaches involve using persistent EC2 instances to perform extract, transform, and load operations. This method requires constant monitoring and lacks the flexibility to replay or restart failed jobs from specific stages.

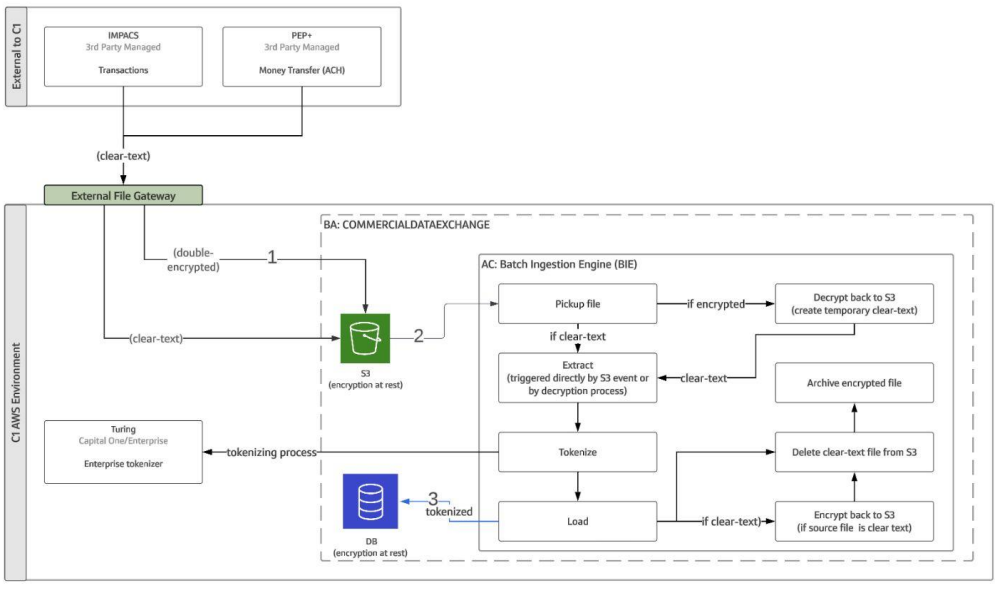
A flexible architecture is necessary to handle different types of data and complex transformation logic. The system must be capable of handling failures gracefully, allowing for granular reprocessing from specific points in the workflow.

Implementation

Our end goal is to extract the file contents, transform them into singular payment/transaction records so that each item represents a transaction, and then load it into DynamoDB. The solution will have three key stages:



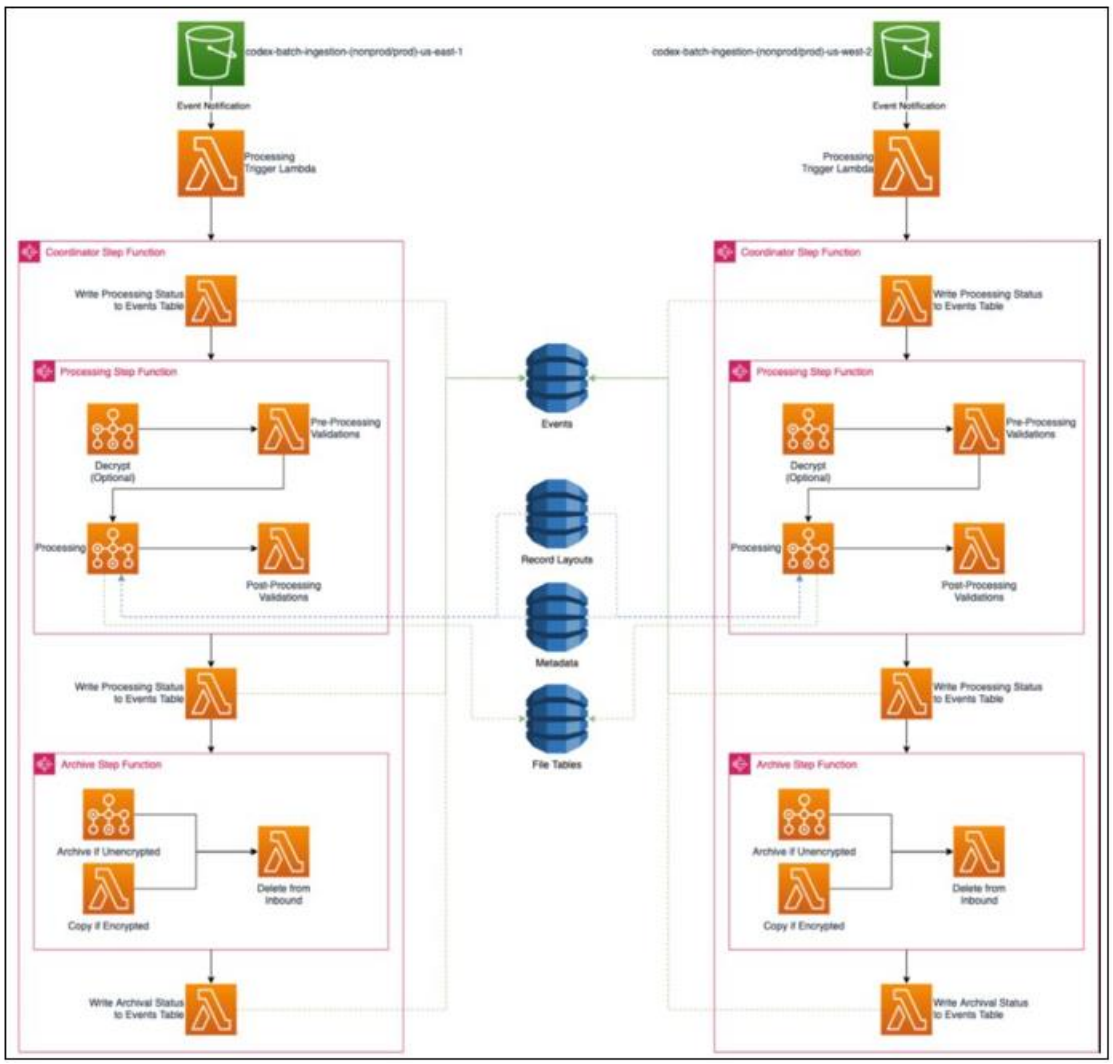
Let's start with Stage 1 & 2



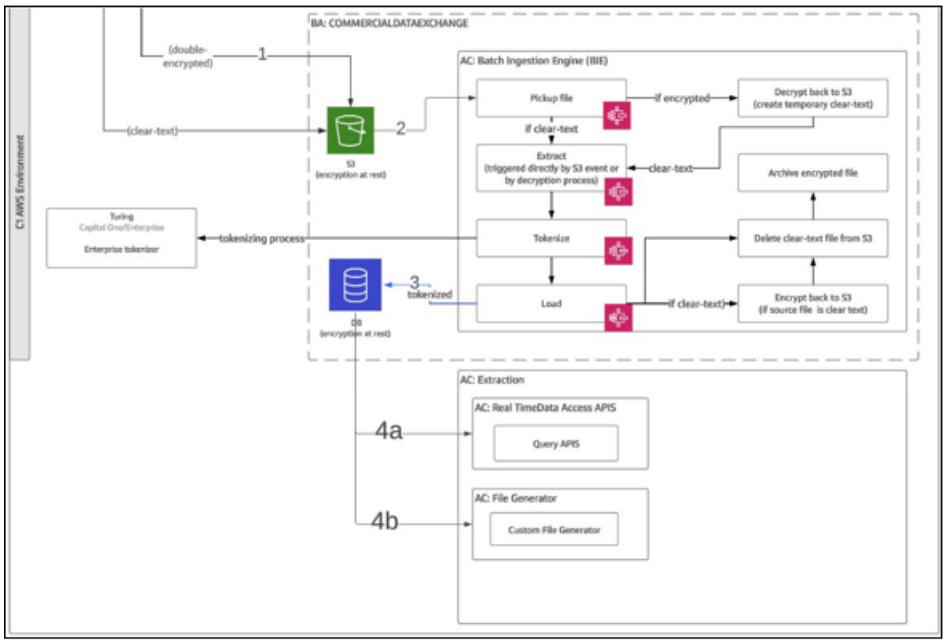
For illustrative purposes only

Step 1. Handles file transfer, encryption/decryption of the file, and brings the file into your internal company ecosystem.
Step 2. This is the heavy lifting stage, where we need to decrypt, tokenize contents, archive, and potentially encrypt the file back. But each of these logical components are blocks of function that can run independently.
Step 3. Now that the raw data is transformed, you can load each transaction into DynamoDB.

Overlaying this logical flow with AWS serverless services would resemble something like the below diagram across regions, creating a highly resilient event - driven file ingestion system.



For Stage 3, based on client needs, we may need to add APIs or file generator components by adding steps 4a and 4b.



Our overall Responsibilities and AWS Service Ownership Boundary as a result of these decisions can be summarized with below takeaways

1) **Event - Driven Triggers**: Use EventBridge to trigger ETL workflows based on events such as file uploads to S3 or message notifications from SNS.
2) **State Management:** Employ AWS Step Functions to define and manage the ETL workflow, including error handling and retries. The state machine will orchestrate the sequence of tasks, utilizing AWS Batch for heavy lifting.
3) **Batch Processing:** AWS Batch will process large volumes of data concurrently, with the results being stored in S3 or DynamoDB as required.
4) **Error Handling and Replay:** Implement granular error handling within Step Functions to capture failure points. Use DynamoDB to store metadata and state information, enabling the system to replay jobs from specific stages as needed.
5) **Data Consistency:** Ensure data consistency and reliability by using DynamoDB for state tracking and S3 for data storage.

2. Conclusion

The proposed serverless ETL/ELT architecture represents a significant advancement over traditional all - or - nothing ETL processes. By leveraging AWS services such as Step Functions, Batch, DynamoDB, and others, this approach provides several key benefits:

1) Scalability: Serverless architecture allows for dynamic scaling, handling varying workloads efficiently without the need for persistent infrastructure.
2) Granular Replay: The integration of DynamoDB for state management and Step Functions for workflow orchestration enables the system to replay jobs from specific stages, enhancing fault tolerance and error recovery.
3) Operational Efficiency: The serverless model reduces operational overhead by automating scaling, resource management, and error handling, allowing engineers to focus on improving the ETL process rather than managing infrastructure.
4) Flexibility: The architecture can adapt to different types of data and transformation requirements, making it suitable for diverse use cases such as payment processing.

This approach offers a robust and resilient solution for modern ETL/ELT workflows, addressing the limitations of traditional methods and providing advanced capabilities for managing complex data processing tasks.

References

[1] Saga Pattern AWS Perspective. Available at: https: //docs. aws. amazon. com/prescriptive - guidance/latest/modernization - data - persistence/saga - pattern. html [Accessed 26 August 2024].
[2] AWS. AWS Step Functions Documentation. Available at: https: //docs. aws. amazon. com/step - functions/latest/dg/welcome. html [Accessed 26 August 2024].
[3] AWS. AWS Batch Documentation. Available at: https: //docs. aws. amazon. com/batch/latest/userguide/what - is - batch. html [Accessed 26 August 2024].
[4] AWS. AWS Lambda Documentation. Available at: https: //docs. aws. amazon. com/lambda/latest/dg/welcome. html [Accessed 26 August 2024].
[5] AWS. Amazon EventBridge Documentation. Available at: https: //docs. aws. amazon. com/eventbridge/latest/userguide/what - is - amazon - eventbridge. html [Accessed 26 August 2024].
[6] AWS. Amazon DynamoDB Documentation. Available at: https: //docs. aws. amazon. com/amazondynamodb/latest/developerguide/Introduction. html [Accessed 26 August 2024].