# Cloud-Based Document Submission System with Real-Time Notification

Tatiya Seetharatkul
*Department of Computer Science*
*and Information Management*
*Asian Institute of Technology*
*st124875@ait.asia*

***Abstract*—This project presents the design and implementation of a cloud-native document submission system utilizing Amazon Web Services (AWS). The platform enables users to securely upload documents to Amazon S3 using pre-signed URLs, ensuring access control without exposing credentials. Upon successful file upload, an AWS Lambda function is automatically triggered to extract relevant metadata, publish an email notification via Amazon Simple Notification Service (SNS), and persist metadata to Amazon DynamoDB. The system was fully developed and tested within the AWS Free Tier environment, demonstrating the feasibility of deploying a cost-effective and scalable serverless solution. Furthermore, a web-based dashboard was created using static website hosting on Amazon S3, integrated with Amazon Cognito and the AWS SDK for JavaScript to support real-time metadata retrieval and visualization. The results confirm the system's effectiveness in delivering a reliable and extensible solution for cloud-based file submissions.**

***Index Terms*—AWS Lambda, Amazon S3, Serverless Architecture, Cloud Computing, Document Upload, SNS, DynamoDB, Pre-signed URL, Static Hosting, Cognito**

## I. Introduction

The increasing demand for secure, scalable, and efficient digital workflows has driven the adoption of cloud-based solutions across a variety of domains. In particular, document submission systems benefit greatly from the flexibility and scalability afforded by cloud computing technologies. This project presents the design and implementation of a serverless, cloud-native document submission platform utilizing Amazon Web Services (AWS). The primary objective is to demonstrate the feasibility of deploying a cost-effective, resilient, and fully managed solution that enables real-time processing and monitoring of uploaded documents.

The proposed system facilitates document uploads through the use of pre-signed URLs generated by an AWS Lambda function. These URLs provide time-limited, secure access to a specified path within an Amazon S3 bucket, allowing users to upload files without requiring direct access to AWS credentials. Once a file has been successfully uploaded, an S3 event automatically triggers a second Lambda function, which is responsible for extracting metadata such as the file name, upload timestamp, and user information. This metadata is then disseminated through Amazon Simple Notification Service (SNS) to notify relevant stakeholders and concurrently stored in Amazon DynamoDB for persistent and structured storage.

To complement the backend functionality, a client-facing dashboard was developed and deployed using Amazon S3's static website hosting feature. This dashboard integrates with Amazon Cognito to enable secure, unauthenticated access for end users, and employs the AWS SDK for JavaScript to retrieve and display metadata from DynamoDB in real time. The interface includes both tabular and graphical visualizations to enhance user interaction and system transparency.

All components of the system were successfully implemented and tested within the AWS Free Tier environment, ensuring cost efficiency while maintaining the architectural principles of security, scalability, and high availability. This project illustrates the practical benefits of a serverless approach and provides a robust foundation for developing future cloud-based document management solutions.

This project was developed as part of the AT83.03 Cloud Computing course at the Asian Institute of Technology.

## II. Literature Review

Cloud computing has transformed the way organizations design, deploy, and scale applications by offering on-demand access to configurable computing resources. Amazon Web Services (AWS), as a leading cloud provider, delivers a comprehensive suite of services that support the development of serverless and event-driven architectures [1]. The use of AWS services such as Amazon S3, AWS Lambda, Amazon SNS, Amazon DynamoDB, and Amazon Cognito facilitates the creation of secure, scalable, and highly available systems.

Serverless architectures have gained significant attention due to their ability to reduce operational overhead and improve scalability. According to Baldini et al. [2], the serverless model allows developers to focus on application logic without managing server infrastructure. AWS Lambda exemplifies this paradigm by enabling function-based execution in response to specific events such as file uploads to S3. AWS Lambda supports a variety of runtimes, integrates with other AWS services, and automatically scales based on demand [3].

Pre-signed URLs provided by Amazon S3 offer a secure mechanism for temporary file uploads without exposing AWS credentials [4]. This technique is useful in client-facing applications where access control is required for uploading files to specific buckets. Amazon S3 also supports features such as object versioning, lifecycle policies, and event notifications,

making it suitable for document submission systems and real-time processing pipelines [5].

Amazon SNS (Simple Notification Service) is a fully managed pub/sub messaging service that enables the decoupling of microservices, distributed systems, and event-driven serverless applications. It supports multiple protocols, including email, HTTP/S, SMS, and AWS Lambda as delivery endpoints. SNS can be integrated with S3 and Lambda to send notifications upon file uploads or processing completions [6].

Amazon DynamoDB, a fully managed NoSQL database, is designed to deliver single-digit millisecond performance at scale. DynamoDB Global Tables extend this capability by replicating data across multiple regions, thereby improving performance and resilience for globally distributed applications [7]. DynamoDB Auto Scaling dynamically adjusts provisioned throughput capacity based on traffic patterns, which aligns with the cost optimization and performance efficiency principles of the AWS Well-Architected Framework [8]. Additional features such as DynamoDB Streams allow for real-time change tracking and event sourcing [9].

Amazon Cognito provides identity management and authentication for web and mobile applications. It allows for secure access to AWS resources through temporary credentials, enabling unauthenticated guest access for public dashboards or authenticated access for registered users [10]. Cognito User Pools provide user directory management, sign-up and sign-in flows, and multi-factor authentication, while Identity Pools enable authorization to AWS resources [11].

In summary, the integration of these AWS services facilitates the development of robust, cloud-native systems. Prior work and documentation validate the reliability, security, and efficiency of serverless models in handling real-time file processing, notifications, and data visualization in production-grade environments.

## III. Objectives

1) To design and implement a cloud-based document submission system using a fully serverless architecture on Amazon Web Services (AWS).
2) To study the use of Amazon S3 pre-signed URLs for secure and controlled document uploads.
3) To explore the use of AWS Lambda functions for automated backend processing triggered by S3 events.
4) To integrate Amazon Simple Notification Service (SNS) for real-time email notifications upon file submission.
5) To store and manage submission metadata using Amazon DynamoDB as a scalable NoSQL database.
6) To implement a client-facing dashboard for visualizing document metadata using Amazon S3 static hosting, Amazon Cognito, and the AWS SDK for JavaScript.
7) To evaluate the feasibility and limitations of deploying a fully functional system within the AWS Free Tier environment.

## IV. AWS Service Components

### A. Secure File Uploads Using Pre-signed URLs

Amazon S3 provides scalable and durable object storage, which serves as the foundation of the document submission system. To ensure secure and controlled file uploads, the system employs pre-signed URLs generated by an AWS Lambda function. A pre-signed URL grants time-limited write access to a specific S3 path, allowing users to upload documents without exposing AWS credentials or requiring additional authentication steps.

Each pre-signed URL is tied to a unique file key and includes embedded credentials and permissions, ensuring that the upload operation is scoped to a single file and limited in time. This mechanism provides fine-grained access control while offloading the upload responsibility to the client side. Once the upload is completed, Amazon S3 automatically stores the object in the designated bucket under the specified prefix (e.g., `demo-user/`).

This design pattern is particularly beneficial in serverless architectures, as it eliminates the need for intermediate storage services or upload APIs while maintaining a high level of security and scalability.

### B. Event-Driven Processing with AWS Lambda

AWS Lambda enables the execution of backend logic in response to specific events without the need to provision or manage servers. In this system, a second Lambda function is configured to be triggered automatically whenever a new object is uploaded to the S3 bucket.

This function is responsible for processing the uploaded file by extracting relevant metadata, such as the file name, size, upload timestamp, and user-defined attributes. The extracted metadata is then prepared for dissemination and storage. To support real-time communication, the function publishes a structured message to an Amazon SNS topic. Simultaneously, it inserts the metadata into a DynamoDB table named `FileMetadata`, allowing for future retrieval and analysis.

Lambda's scalability and low-latency response time make it ideal for implementing such reactive workflows, ensuring that each uploaded file is processed promptly and consistently.

### C. Real-Time Notifications with Amazon SNS

Amazon Simple Notification Service (SNS) is used to deliver email notifications immediately after a document is uploaded and processed. Once the metadata is successfully extracted, the Lambda function formats a notification message and publishes it to a predefined SNS topic.

The SNS topic is configured with an email subscription, ensuring that recipients are notified with relevant details such as the file name, upload time, and a confirmation of successful submission. SNS supports multiple protocols, but for the scope of this project, only email delivery was used.

This integration provides a simple yet effective communication channel, enhancing the user experience by acknowledging uploads in real time.

### D. Metadata Storage with Amazon DynamoDB

Amazon DynamoDB is a fully managed NoSQL database service known for its low-latency performance and automatic scaling. In the context of this project, DynamoDB is used to persist metadata associated with each file submission.

Each metadata record includes fields such as the `file_key`, `uploader`, `timestamp`. These records are stored in the `FileMetadata` table and can be retrieved by the web dashboard for display.

DynamoDB's seamless integration with Lambda and its support for fine-grained access control make it an excellent choice for metadata storage in a serverless workflow.

### E. User Authentication with Amazon Cognito

Amazon Cognito is used to manage access to the web-based dashboard. In this implementation, an unauthenticated identity pool is configured to allow guest access to read data from DynamoDB using the AWS SDK for JavaScript. Cognito provides temporary credentials that are limited in scope and duration, ensuring secure access without requiring users to log in.

This approach enables controlled frontend access while maintaining the integrity and security of the backend resources.

### F. Static Web Hosting and Visualization

To visualize submission metadata, a front-end dashboard is developed using HTML, JavaScript, and Chart.js, and is hosted via Amazon S3's static website hosting feature. The dashboard fetches metadata records from DynamoDB using the AWS SDK and presents them in a dynamic table and bar chart.

This component enhances system transparency and allows users or administrators to monitor submission activity in real time without the need for a separate web server or backend infrastructure.

## V. System Architecture

The system consists of multiple loosely coupled AWS components (Figure 1) following a serverless paradigm. The process starts with a client requesting a pre-signed URL from a Lambda function. This URL is used to upload a file to S3 securely. Once uploaded, S3 invokes another Lambda function via an event trigger. This function parses the file metadata, sends an email via SNS, and saves details in DynamoDB. The uploaded files can then be viewed in a dynamic dashboard powered by static hosting and real-time queries using AWS Cognito identity pools.

## VI. Methodology

This section outlines the step-by-step process followed to implement a secure, serverless document submission system using Amazon Web Services (AWS). The implementation focuses on enabling file uploads to Amazon S3 via pre-signed URLs, real-time metadata processing through AWS Lambda, notification via Amazon SNS, metadata storage using Amazon DynamoDB, and a front-end dashboard secured by Amazon Cognito and hosted on S3.

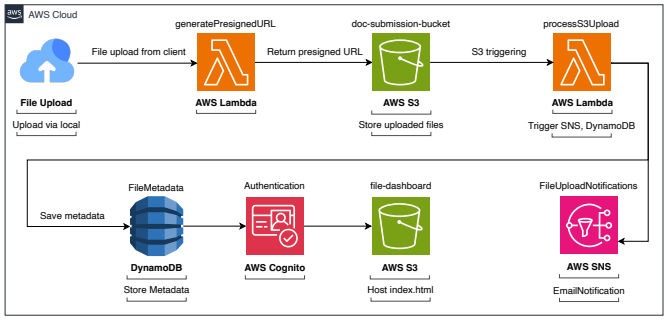


Fig. 1. System architecture of the cloud-based document submission platform.

### A. S3 Bucket Configuration

The process begins with the creation of an Amazon S3 bucket named `doc-submission-bucket` (Figure 2), which serves as the primary storage location for uploaded files. This bucket is configured with all public access blocked to maintain data security. Versioning is optionally enabled to retain multiple versions of uploaded files. All files are uploaded to a designated folder within the bucket (e.g., `demo-user/`) to facilitate organized storage and enable precise filtering using S3 event triggers.

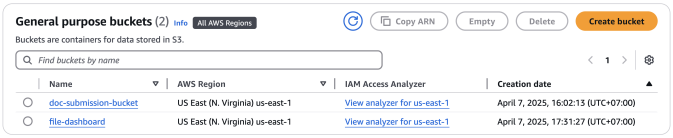


Fig. 2. Configuration of the document submission S3 bucket.

### B. Lambda Function Deployment

Two AWS Lambda functions were implemented by using Python 3.12 runtime:

- **generatePresignedURL:** Responsible for generating a time-limited pre-signed URL to allow users to securely upload files to S3 without exposing credentials. (Figure 3).
- **processS3Upload:** Triggered automatically upon a successful file upload. This function extracts metadata, publishes a notification to an SNS topic, and stores metadata in DynamoDB. (Figure 4).

### C. Event-Based S3 Trigger

The `processS3Upload` function is connected to the S3 bucket via an event trigger. The trigger is configured to respond to `PUT` events, limited by prefix `demo-user/` and suffix `.pdf` to ensure it only processes relevant uploads. This mechanism provides near real-time response upon file submission.


Fig. 3. Lambda functions: generatePresignedURL.

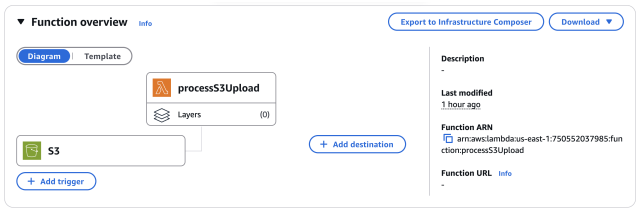


Fig. 4. Lambda functions: processS3Upload.

## D. IAM Policy Configuration

Dedicated IAM roles were attached to the Lambda functions to enforce the principle of least privilege. The `generatePresignedURL` function (Figure 5) has permission to write to the S3 bucket, while `processS3Upload` (Figure 6) has policies that allow it to publish messages to an SNS topic and write entries into the DynamoDB table.


Fig. 5. IAM policies for Lambda function : generatePresignedURL.


Fig. 6. IAM policies for Lambda function : processS3Upload.

## E. Notification Service Integration

Amazon Simple Notification Service (SNS) is used to send email notifications to a pre-configured subscriber (Figure 7). The topic is created and linked to an email endpoint. After confirming the subscription, the `processS3Upload` function publishes structured messages upon each successful upload.

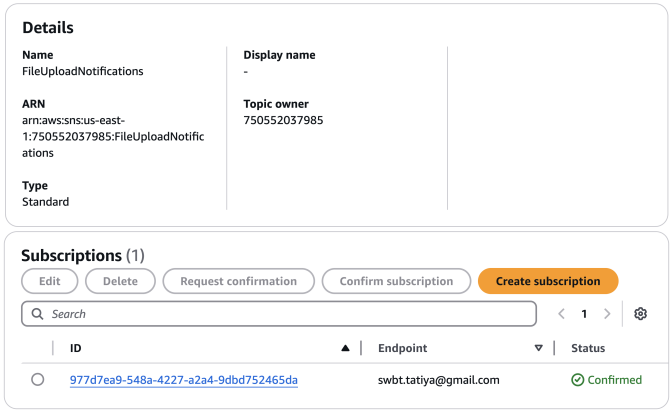


Fig. 7. SNS topic and email subscription setup for real-time notifications.

## F. DynamoDB Table Creation

A DynamoDB table named `FileMetadata` is created to store the file metadata (Figure 8). The table uses `file_key` as the primary key. Metadata such as timestamp and uploader details are stored for each file submission, allowing for structured retrieval and analysis.

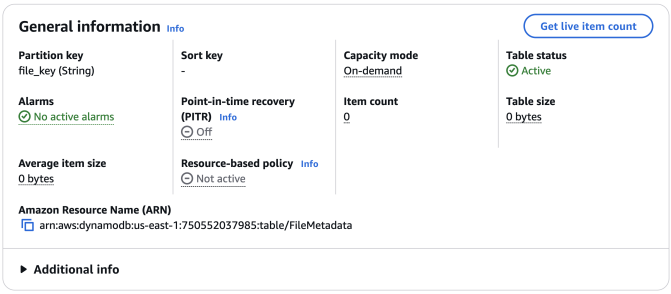


Fig. 8. DynamoDB table for storing metadata.

## G. Web Dashboard Implementation and Hosting

A static HTML/JavaScript dashboard is developed to visualize the file metadata stored in DynamoDB. The dashboard uses the AWS SDK for JavaScript and authenticates using Amazon Cognito Identity Pools. Only unauthenticated guest access is enabled for controlled public access.

The dashboard is hosted via Amazon S3 static website hosting. The HTML file is uploaded to a separate bucket, and a public-read bucket policy is configured to allow global access to the hosted dashboard.

**Step 1: Configure Cognito Identity Pool**

To enable unauthenticated guest access, a Cognito Identity Pool named `FileDashboardPool` is created (Figure 9). This identity pool allows temporary AWS credentials to be issued for accessing DynamoDB and other services without requiring user login.

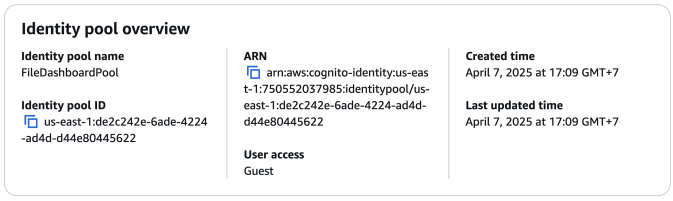


Fig. 9. Cognito Identity Pool `FileDashboardPool` with guest access, showing Identity Pool ID and ARN for SDK integration.

### Step 2: Grant DynamoDB Read Access via IAM

An IAM inline policy is attached to the unauthenticated role generated by the Cognito identity pool. This policy grants `dynamodb:Scan` permission for the `FileMetadata` table, allowing the dashboard to fetch data securely from DynamoDB (Figure 10).


Fig. 10. IAM inline policy granting unauthenticated users permission to scan the `FileMetadata` DynamoDB table.

### Step 3: Set Bucket Policy for Public Access

The dashboard is uploaded to a public S3 bucket and hosted using static website hosting (Figure 11). To allow public users to access the HTML file, a bucket policy is applied that grants `s3:GetObject` access to all objects in the bucket (Figure 12).

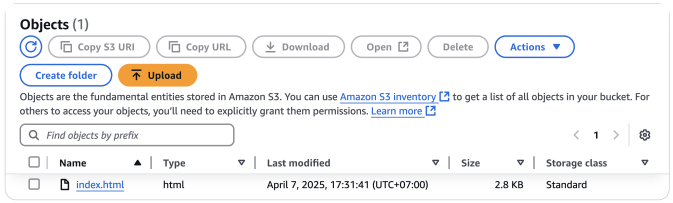


Fig. 11. Configuration of the static website hosting S3 bucket.

### *H. End-to-End Testing*

To verify the complete system functionality, a file is uploaded using the pre-signed URL obtained from the API (Figure 13). Upon successful upload (Figure 14), the system automatically triggers the metadata processing Lambda function, resulting in:

- Email notification via SNS (Figure 15).
- Metadata entry in DynamoDB (Figure 16).
- Visualization on the hosted dashboard (Figure 17).

The pre-signed URL is tested using command-line tools such as `curl`, simulating an upload from a client environment.


Fig. 12. S3 bucket policy allowing public read access to the static web dashboard.

## VII. Results

Upon a successful upload, the system is designed to:

- Store uploaded files in 'doc-submission-bucket' bucket.
- Send an email notification via SNS.
- Store metadata in the 'FileMetadata' DynamoDB table.
- Display metadata in a dynamic dashboard.
- Log processing activity in Amazon CloudWatch.

## VIII. Challenges and Limitations

The initial development of this project was conducted within the AWS Academy Cloud Foundations Sandbox, which introduced several service-level restrictions. Specifically, the environment did not permit:

- Publishing to Amazon SNS topics
- Writing to Amazon DynamoDB tables
- Creating Amazon Cognito identity pools

Due to these limitations, early testing focused on verifying system logic through CloudWatch logs. These logs confirmed that file uploads to S3 triggered the appropriate Lambda functions, metadata was parsed correctly, and intended payloads for SNS and DynamoDB were generated.

To enable full implementation and validation, the system was later migrated to the AWS Free Tier environment. This allowed unrestricted access to all required services and enabled complete end-to-end testing, including real-time notifications, metadata persistence, and secure dashboard access via Cognito. The migration confirmed the system's functional correctness and deployment readiness.

## IX. AWS Well-Architected Framework Alignment

This cloud-based document submission system was designed in alignment with the AWS Well-Architected Framework, adhering to its Five Pillars: Operational Excellence, Security, Reliability, Performance Efficiency, and Cost Optimization. Each pillar is addressed as follows:

For **Operational Excellence**, the system uses AWS Lambda and Amazon CloudWatch to enable real-time monitoring and debugging through automated logging. Amazon S3 event triggers are configured to invoke the `processS3Upload`

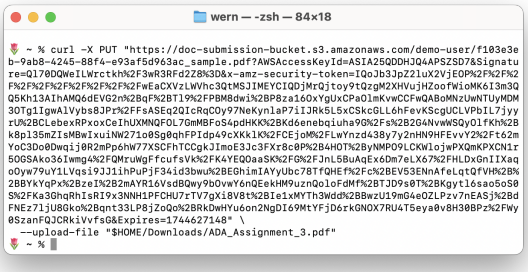


Fig. 13. File is uploaded using `curl` and a pre-signed URL generated from the backend Lambda function.

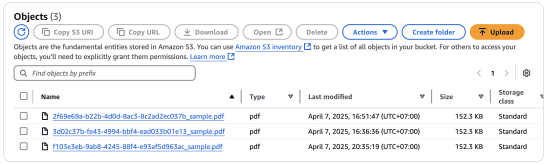


Fig. 14. The uploaded file appears in the specified Amazon S3 bucket.

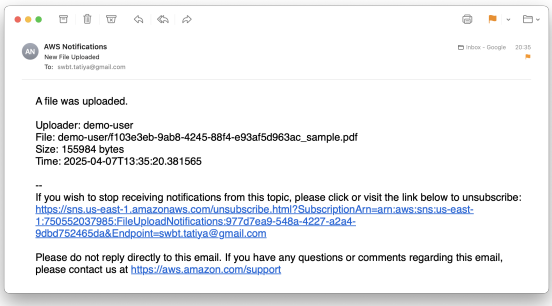


Fig. 15. An SNS-triggered email notification is sent upon successful upload.

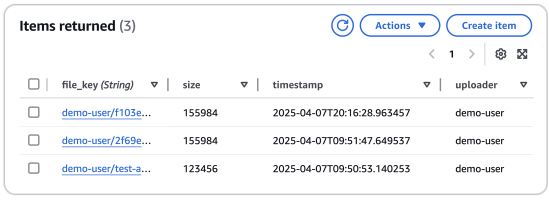


Fig. 16. Metadata such as file key, size, timestamp, and uploader is automatically stored in the DynamoDB table.

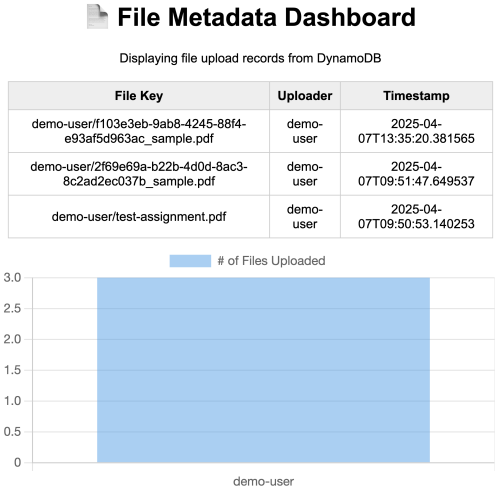


Fig. 17. The uploaded file metadata is displayed in a dynamic table on the web dashboard.

Lambda function immediately upon file uploads, eliminating the need for manual polling or scheduled checks. The operational processes are automated through serverless architecture, allowing tasks to be repeatable, consistent, and free from human error.

In terms of **Security**, Amazon Cognito handles user authentication, providing secure access through the use of temporary credentials. The system leverages pre-signed URLs to enable secure file uploads directly to Amazon S3, thereby minimizing the exposure of direct write access to storage resources. IAM policies are designed following the principle of least privilege, although some restrictions in the AWS Academy sandbox environment limited the full customization of permission settings.

Regarding **Reliability**, the system benefits from Amazon S3's exceptional 99.999999999% durability for document storage, ensuring that uploaded files are preserved with minimal risk of loss. AWS Lambda functions are stateless, scalable, and configured to automatically retry in specific failure scenarios, improving the robustness of the workflow. Amazon DynamoDB provides highly available metadata storage, with built-in fault tolerance to maintain system reliability even under varying load conditions.

The **Performance Efficiency** pillar is achieved through the adoption of an event-driven architecture, where compute resources automatically scale based on the number of incoming file uploads. No server infrastructure provisioning is required, which accelerates development speed and optimizes resource usage. Additionally, the use of pre-signed URLs for client uploads enhances client-side performance by reducing the need for repeated authentication overhead.

Finally, for **Cost Optimization**, the system adopts a pay-per-use billing model, ensuring that charges only accrue for actual Lambda executions, S3 storage operations, and DynamoDB interactions. Because no idle compute resources are provisioned, unnecessary costs are avoided. Furthermore, the entire system is designed to remain within the limits of the AWS Free Tier, making it highly cost-effective and suitable for academic and experimental deployment scenarios.

## X. Conclusion

This cloud-based document submission system was designed in alignment with the AWS Well-Architected Framework, adhering to its Five Pillars: Operational Excellence, Security, Reliability, Performance Efficiency, and Cost Optimization. The system demonstrates a secure, serverless architecture using fully managed AWS services, including Amazon S3 for storage, AWS Lambda for compute, Amazon SNS for notifications, Amazon DynamoDB for metadata storage, and Amazon Cognito for access control.

The methodology successfully integrates these services into a cohesive, event-driven architecture that delivers scalability, fault tolerance, and real-time responsiveness. All components interact seamlessly, and the system performs as expected under test conditions. Pre-signed URLs enable secure uploads, while Lambda triggers automate metadata extraction and processing.

The dashboard component, hosted via S3 and authenticated using Cognito, offers an intuitive interface for real-time data visualization.

In environments that require global availability and low latency, Amazon DynamoDB Global Tables serve as a robust foundation. These tables provide automatic multi-region replication and conflict resolution, ensuring that applications remain highly available even in the event of regional degradation. By eliminating the need for custom replication code, Global Tables simplify the development of distributed systems.

Furthermore, DynamoDB Auto Scaling contributes to cost efficiency and operational simplicity. It continuously monitors traffic and adjusts provisioned throughput to meet demand, thereby reducing unnecessary overprovisioning. Previous research has shown that auto scaling can reduce costs significantly while maintaining performance standards, especially in workloads with fluctuating traffic.

While earlier versions of this system were tested under the constraints of an academic sandbox, this final implementation was fully realized within the AWS Free Tier. All services and workflows were implemented and validated in a real environment, confirming both functional correctness and practical feasibility. This project provides a strong foundation for future enhancements and production-level deployment, where additional features such as authentication, version control, or analytics may be added to further enrich the system.

## ACKNOWLEDGMENT

The author would like to express sincere gratitude to Dr. Nisit Pukrongta for his guidance and support throughout the development of this project as part of the AT83.03 Cloud Computing course. Special thanks are also extended to Mr. Manash Mahanta for his assistance and valuable feedback during the implementation process.

Appreciation is further extended to the AWS Academy program and the Asian Institute of Technology for providing access to cloud infrastructure and learning resources that enabled the successful completion of this work.

## PROJECT REPOSITORY

The complete source code and configuration files for this project are available on GitHub at this repository[1].

## REFERENCES

[1] Amazon Web Services, “Overview of Amazon Web Services,” [Online]. Available: https://docs.aws.amazon.com/whitepapers/latest/aws-overview/aws-overview.pdf

[2] I. Baldini et al., “Serverless Computing: Current Trends and Open Problems,” in Research Advances in Cloud Computing, Springer, 2017.

[3] Amazon Web Services, “AWS Lambda Developer Guide,” [Online]. Available: https://docs.aws.amazon.com/lambda/latest/dg/welcome.html

[4] Amazon Web Services, “Using pre-signed URLs,” [Online]. Available: https://docs.aws.amazon.com/AmazonS3/latest/userguide/ShareObjectPreSignedURL.html

[5] Amazon Web Services, “Amazon S3 features,” [Online]. Available: https://docs.aws.amazon.com/AmazonS3/latest/userguide/Welcome.html

[6] Amazon Web Services, “Amazon SNS Developer Guide,” [Online]. Available: https://docs.aws.amazon.com/sns/latest/dg/welcome.html

[7] Amazon Web Services, “Global Tables,” [Online]. Available: https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/GlobalTables.html

[8] Amazon Web Services, “DynamoDB Auto Scaling,” [Online]. Available: https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/AutoScaling.html

[9] Amazon Web Services, “Change Data Capture for DynamoDB,” [Online]. Available: https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Streams.html

[10] Amazon Web Services, “Amazon Cognito Developer Guide,” [Online]. Available: https://docs.aws.amazon.com/cognito/latest/developerguide/

[11] Amazon Web Services, “Cognito User Pools and Identity Pools,” [Online]. Available: https://docs.aws.amazon.com/cognito/latest/developerguide/cognito-identity.html

[1] https://github.com/werrnnnwerrrnnnnnn/document-submission-system