|  |  |
| --- | --- |
| **CSV File Processing System Using AWS** | Abstract  The CSV File Processing System using AWS is a serverless architecture designed to automate the processing of CSV files uploaded to an Amazon S3 bucket. It uses AWS Lambda to trigger file processing, extract metadata (such as file size, row count, column count, and upload timestamp) using Pandas, and store this metadata in Amazon DynamoDB for future reference. The system is secured using IAM Roles and monitored through CloudWatch Logs for real-time tracking. This approach eliminates manual file processing, enhances scalability, and reduces operational costs by leveraging serverless computing, making it an ideal solution for data automation and processing workflows.  Raj Kumar Mourya |

**Project Cover Page**

**CSV File Processing System Using AWS**

Submitted by: Raj Kumar Mourya  
Submitted to: Uplyft.ai  
Date of Submission: 06 March 2025  
Version: v1.0

**Table Of Contents**

[Overview 3](#_Toc192628756)

[Core AWS Services 3](#_Toc192628757)

[Project Workflow Diagram 4](#_Toc192628758)

[Steps Involved 5](#_Toc192628759)

[Creating an S3 Bucket 5](#_Toc192628760)

[Step-by-Step Process 6](#_Toc192628761)

[Uploading a Sample File 6](#_Toc192628762)

[Creating an IAM Role for Lambda Execution 7](#_Toc192628763)

[Step-by-Step Process 7](#_Toc192628764)

[Developing and Deploying an AWS Lambda Function 8](#_Toc192628765)

[Writing the Python Code 9](#_Toc192628766)

[Testing the Lambda Function 10](#_Toc192628767)

[Deploying the Lambda Function 11](#_Toc192628768)

[Setting Up Amazon DynamoDB 11](#_Toc192628769)

[Creating a DynamoDB Table 11](#_Toc192628770)

[Example Table Structure 12](#_Toc192628771)

[Configuring Amazon CloudWatch for Monitoring 12](#_Toc192628772)

Error Handling and Troubleshooting…………………………………………………………..14

[Conclusion 15](#_Toc192628773)

**CSV File Processing System with AWS**

# Overview

In this document, we will explore the implementation of a serverless CSV file processing system built on Amazon Web Services (AWS). This architecture leverages several AWS services, each playing a crucial role in enhancing scalability, efficiency, and monitoring.

## Core AWS Services

**Amazon S3 (Simple Storage Service)**:

* S3 acts as our primary storage solution, where users can upload files. It ensures high durability and availability of data.
* The uploaded files can trigger Lambda functions based on events (e.g., when a file is created or modified).

**AWS Lambda**:

* Lambda functions are serverless compute services that automatically execute code in response to events from other AWS services like S3.
* In our file processing workflow, Lambda can manipulate files (e.g., resize images or extract data) without provisioning or managing servers, offering cost-effective scalability.

**AWS IAM (Identity and Access Management) Roles**:

* IAM roles provide security by governing access to AWS resources. They define permissions for different components of our architecture, ensuring only authorized services can perform specific actions.
* For example, a Lambda function needs to read from S3 but should not have unrestricted access to other AWS services.

**Amazon DynamoDB**:

* This NoSQL database serves as a fast and flexible data store to hold metadata related to processed files, such as status and processing time.
* DynamoDB provides seamless scalability to accommodate varying workloads, making it ideal for our real-time file processing needs.

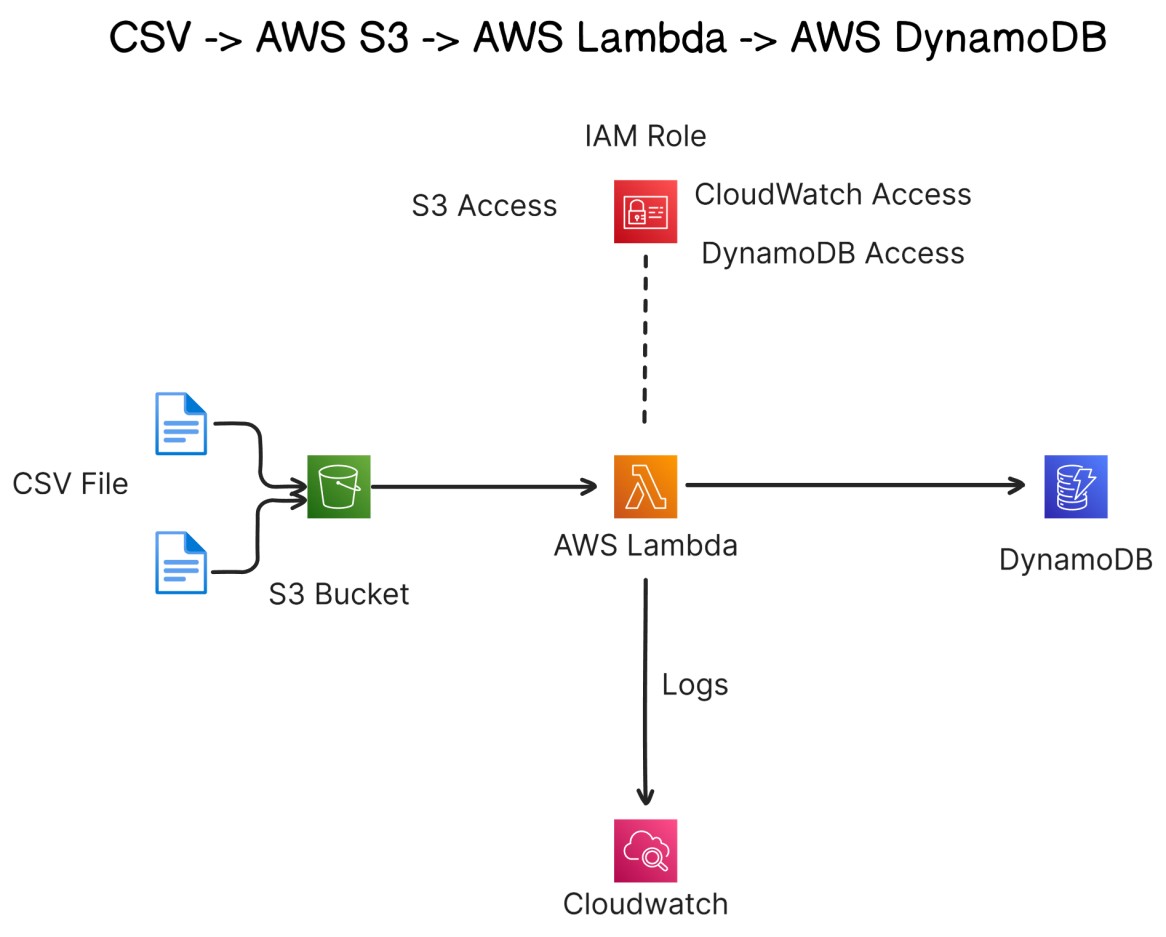
**Amazon CloudWatch**:

* CloudWatch provides monitoring and logging capabilities for all components of our serverless architecture.
* It plays a vital role in tracking application performance and resource utilization, enabling proactive management and troubleshooting.

By seamlessly integrating these services, the serverless file processing system not only achieves optimal resource utilization but also ensures rapid response times and reduced operational costs.

# Project Workflow Diagram

The project workflow diagram is an essential component that visually represents the interactions between the various components of the serverless architecture for the file processing system. This diagram illustrates how data flows through the system and how different AWS services collaborate to achieve the processing of files efficiently.



**Key Elements Illustrated in the Diagram:**

**File Uploads**: The diagram begins with users uploading files to **Amazon S3**. This serves as the entry point for all file operations.

**Event Triggering**: Upon a new file upload, an event is triggered, which instigates a **Lambda function** to process the file according to defined rules.

**IAM Role:** Provides the required access permissions to the Lambda Funtion.

**Data Handling**: Processed data can be stored in **Amazon DynamoDB**, where relevant metadata is maintained for tracking processing status and other attributes.

**Monitoring**: **Amazon CloudWatch** plays a crucial role, with metrics and logs being collected throughout the workflow, aiding in performance monitoring and alerting.

This diagram is vital as it encapsulates the entire workflow, facilitating a better understanding of how components interact in a serverless environment. For those who have access to it, attaching the actual diagram here will enhance the comprehension of the architecture and provide a clearer picture of the system’s design.

# Steps Involved

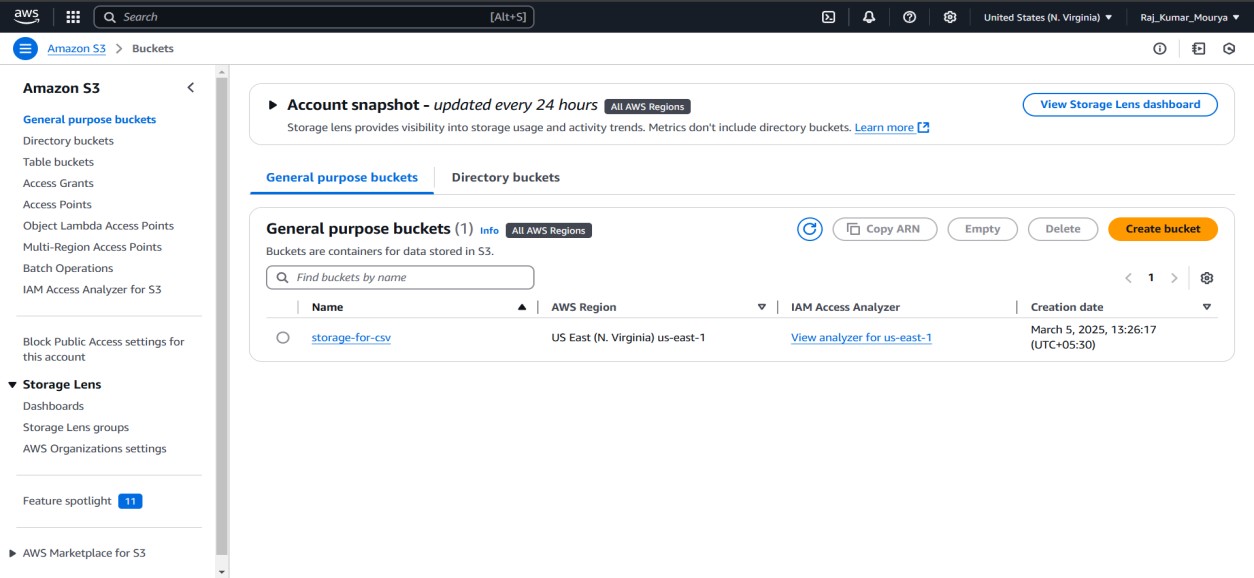
Creating a serverless file processing system requires careful execution of several key steps. Following each step meticulously is essential to ensure a successful implementation. Below is a brief outline:

1. **Setup Amazon S3**: Configure S3 to receive file uploads and set event triggers for Lambda functions.
2. **Develop AWS Lambda Functions**: Write and deploy functions to handle file processing tasks according to specifications.
3. **Define IAM Roles**: Establish IAM roles that grant the necessary permissions for Lambda and other AWS services.
4. **Integrate Amazon DynamoDB**: Set up a DynamoDB table to store metadata related to processed files.
5. **Enable CloudWatch Monitoring**: Implement CloudWatch for tracking application performance and logging important events.

These steps form the foundation of your serverless architecture.

# Creating an S3 Bucket

Creating an S3 bucket is the first step in building your serverless file processing system on AWS. This section outlines how to create an S3 bucket through the AWS Management Console and configure it appropriately for file uploads.



## Step-by-Step Process

**Log in to the AWS Management Console**:

* Visit the [AWS Management Console](https://aws.amazon.com/console/) and log in with your AWS credentials.

**Navigate to S3**:

* Once logged in, search for “S3” in the search bar or find it under the "Storage" section.

**Create a New Bucket**:

* Click on the **Create bucket** button.
* **Bucket Name**: Enter a globally unique name for your bucket (e.g., my-fileprocessing-bucket).
* **Region**: Choose a region for your bucket. Selecting a region close to your user base can reduce latency.

**Review and Create**:

* Review your settings and click **Create bucket**.

## Uploading a Sample File

After creating your S3 bucket, you can test the setup by uploading a sample file.

**Open Your Bucket**:

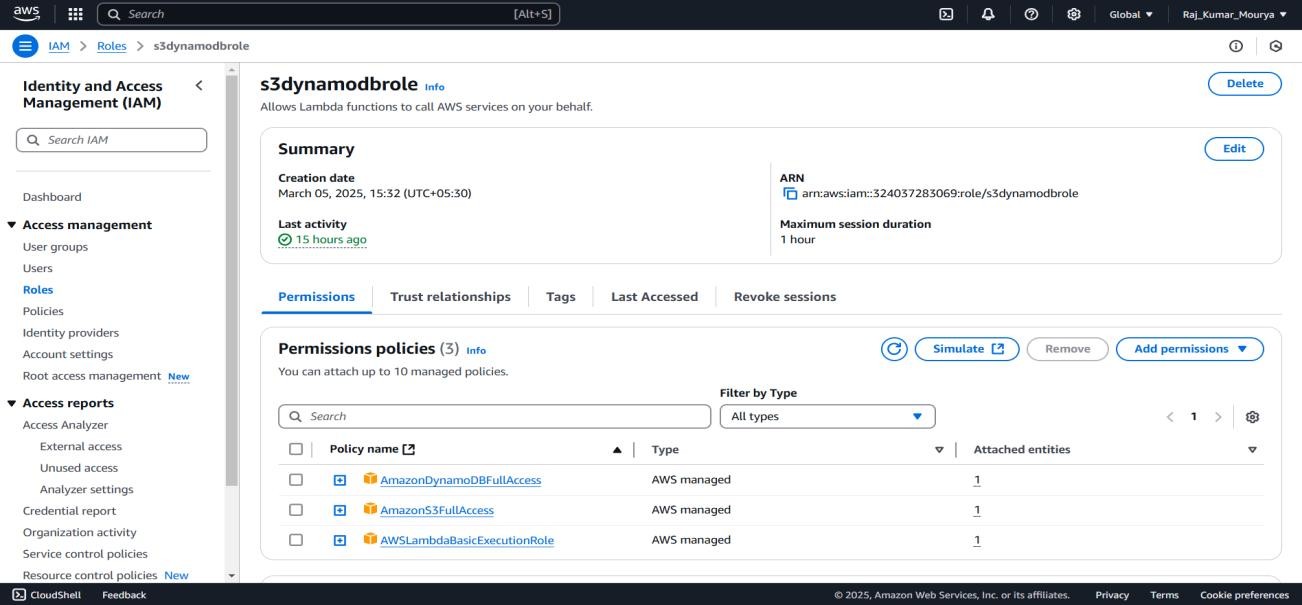
* Click on the bucket name you just created to open its dashboard.

**Upload Files**:

* Click on the **Upload** button.
* Click **Add files** and select a sample file from your local machine.
* Optionally, you can define **storage class** (e.g., Standard, Intelligent-Tiering) based on your needs for access frequency.
* Click on **Upload** once you’re ready.

# Creating an IAM Role for Lambda Execution

To effectively utilize AWS Lambda for file processing, it is essential to create an IAM (Identity and Access Management) role that grants the necessary permissions for Lambda functions to operate securely and efficiently. Below, we outline the detailed steps required to create this IAM role.



## Step-by-Step Process

**Access IAM Service**:

* On the console's homepage, search for and select **IAM**. This takes you to the Identity and Access Management dashboard.

**Create a New Role**:

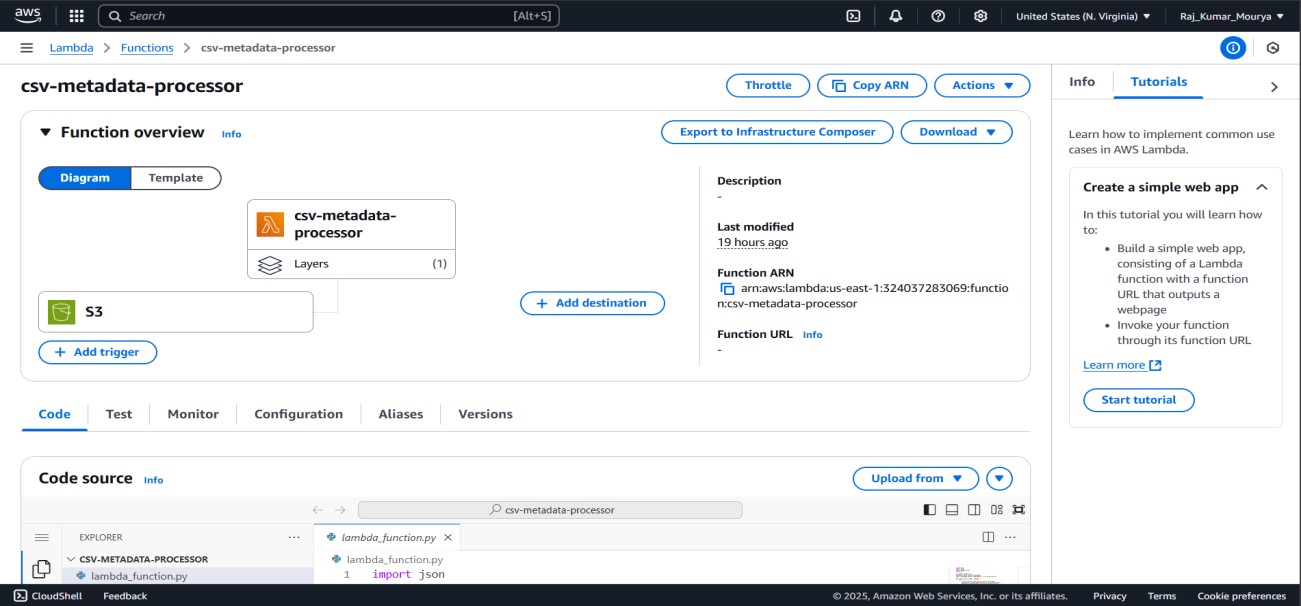
* In the IAM dashboard, click on **Roles** from the sidebar.
* Select **Create role**.

**Attach Policies**:

* Here, you will need to attach policies that define what the Lambda function will be allowed to do. A common policy may include:
  + **AmazonS3AllAccess**: Allows the Lambda function to read from S3 buckets.
  + **AmazonDynamoDBAllAccess:** Allow the Lambda function to store the data in the DynamoDB database.
  + **AWSLambdaBasicExecutionRole**: Permits the function to write logs to Amazon CloudWatch.

# Developing and Deploying an AWS Lambda Function

Creating and deploying an AWS Lambda function is a key step in building a serverless file processing system. This section delves into the necessary steps for authoring a Lambda function, assigning the appropriate IAM role, and writing Python code to handle S3 events.



**Access the AWS Lambda Console**:

* Log in to the AWS Management Console and navigate to the **Lambda** service.
* Click on **Create function**.

**Choose Author from Scratch**:

* Select **Author from scratch** for initial setup.
* **Function Name**: Give your Lambda function a meaningful name, such as FileProcessorFunction.
* **Runtime**: Choose the Python runtime (e.g., Python 3.9) for your coding needs.

**Assign an IAM Role**:

* Under **Permissions**, choose the existing IAM role created earlier (e.g., LambdaExecutionRole). This role provides the necessary permissions to access other services like S3 and DynamoDB.

## Writing the Python Code

To set up event-driven file processing, you need to write the Lambda function logic in Python. Below is an example of code that processes S3 events:



## Testing the Lambda Function

Once the function is written, it’s crucial to test its functionality before deploying:

**Test Event**:

* In the AWS Lambda console, configure a **Test Event** by simulating an S3 event structure. This helps to validate that the function processes files as intended.

**Run the Test**:

* Click **Test** to execute the function against the simulated event. Review the logs for any errors or print statements you added to monitor processing flow.

## Deploying the Lambda Function

**Save and Deploy**:

* After successfully testing, click on the **Deploy** button to publish the function. AWS automatically manages the deployment process.

**Set Trigger for Lambda**:

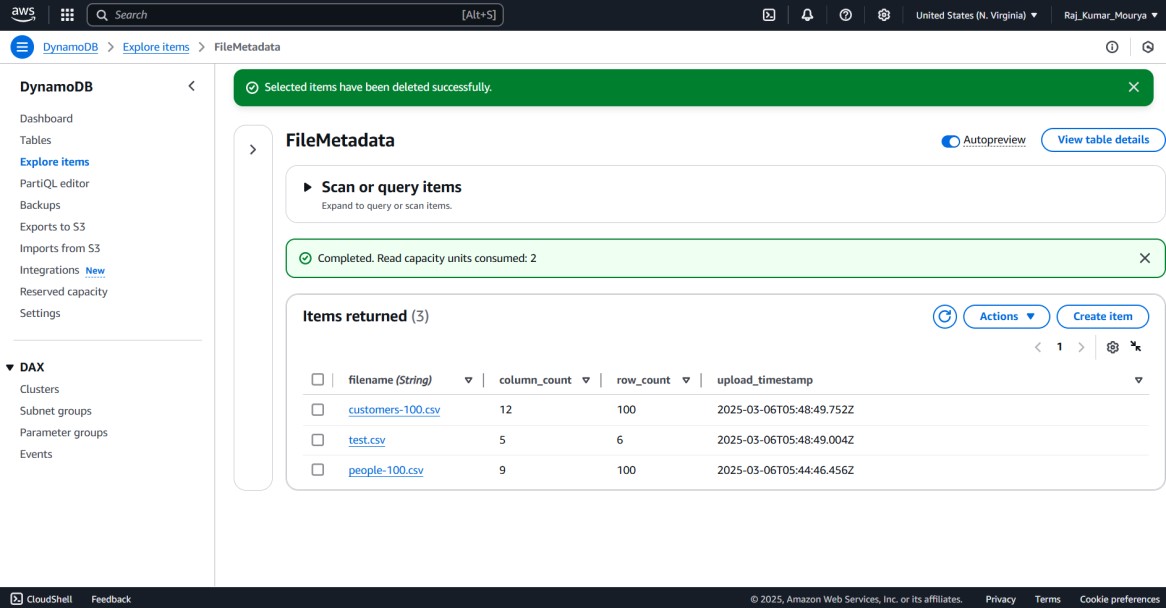
* Connect the Lambda function to the S3 bucket to automate the invocation. In the S3 console, navigate to the bucket properties and add an event notification that triggers the Lambda function upon file uploads.

**Monitoring Deployments**:

* Enable **CloudWatch Logs** to monitor the execution and performance, providing insights into any operational issues that may arise during file processing.

# Setting Up Amazon DynamoDB

Setting up Amazon DynamoDB is essential for storing and managing metadata related to the processed files in your serverless architecture. This section will walk you through creating a DynamoDB table, defining the primary key, and adding necessary attributes to support your file processing needs.



## Creating a DynamoDB Table

**Navigate to DynamoDB**:

* Search for “DynamoDB” in the search bar or find it under the "Database" section.

**Create a New Table**:

* Click on the **Create table** button.

**Define the Table Details**:

* **Table Name**: Enter a unique table name, such as ProcessedFilesMetadata.
* **Primary Key**: Define a primary key using:

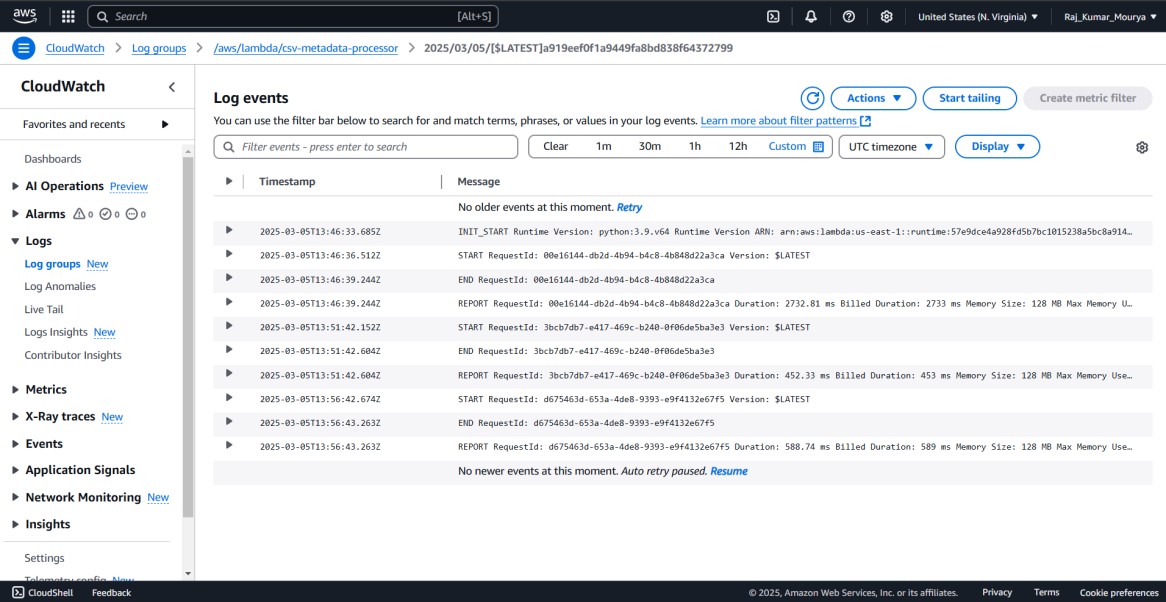
– **Partition Key**: This key is mandatory and can be the Filename (a unique identifier for each file).

## Example Table Structure

|  |  |  |
| --- | --- | --- |
| **Attribute** | **Type** | **Description** |
| FileName | String | Name of the file |
| Column Count | Number | Total no of column in the file |
| Row Count | Number | Total no of row in the file |
| Timestamp | Number | Define the upload date & time |

# Configuring Amazon CloudWatch for Monitoring

Monitoring your serverless architecture is crucial for ensuring optimal performance and reliability. Amazon CloudWatch provides powerful tools to collect and analyze metrics and logs from AWS services, such as AWS Lambda. This section describes the configuration process for using CloudWatch to monitor your Lambda function, enable logging, and set up alarms.



**Basic Logging**:

* By default, AWS Lambda automatically integrates with CloudWatch Logs. When you create or update a Lambda function, you can enable logging using the predefined policy **AWSLambdaBasicExecutionRole**, allowing the function to write to CloudWatch Logs.

**Accessing Logs**:

* After execution, you can view logs in the **CloudWatch** console. Navigate to

**Logs**, select the log group associated with your Lambda function (formatted as /aws/lambda/<LambdaFunctionName>), and view individual log streams for execution details.

**Log Format**:

* Use structured logging, which makes it easier to parse logs. For example, use log statements that include timestamps, file names, and processing states:

By configuring CloudWatch for monitoring, you can gain valuable insights into your serverless architecture, ensuring it remains scalable, efficient, and responsive to user needs.

**Error Handling and Troubleshooting**

* **File format issues**: Ensure CSV files are properly formatted.
* **Missing Pandas Module**: Use a Lambda Layer for pandas.
* **IAM permission issues**: Verify the Lambda IAM role has access to S3 and DynamoDB.
* **Debugging**: Use **CloudWatch Logs** for error tracking.

# Conclusion

The implementation of a serverless architecture for the file processing system using AWS services marks a significant success in enhancing operational efficiency. The seamless integration of Amazon S3, AWS Lambda, IAM roles, DynamoDB, and CloudWatch has provided several noteworthy benefits.