

**Name :** Viven Hotwani

**Roll No :** 17

**Class :** D15C

**Topic No :** 17

## Case Study

# Serverless Data Processing with DynamoDB - Case Study Report

## 1. Introduction

### Case Study Overview

This case study demonstrates the implementation of a serverless data processing pipeline using AWS services. The system automatically processes JSON files uploaded to an S3 bucket and stores relevant data in DynamoDB, showcasing event-driven architecture and serverless computing principles.

### Key Features and Applications

- **Event-Driven Processing:** Automatic triggering of Lambda functions on S3 file uploads
- **Serverless Architecture:** No server management required, pay-per-use model
- **Real-Time Data Processing:** Immediate processing of uploaded JSON files
- **Scalable Storage:** Use of DynamoDB for flexible, schema-less data storage

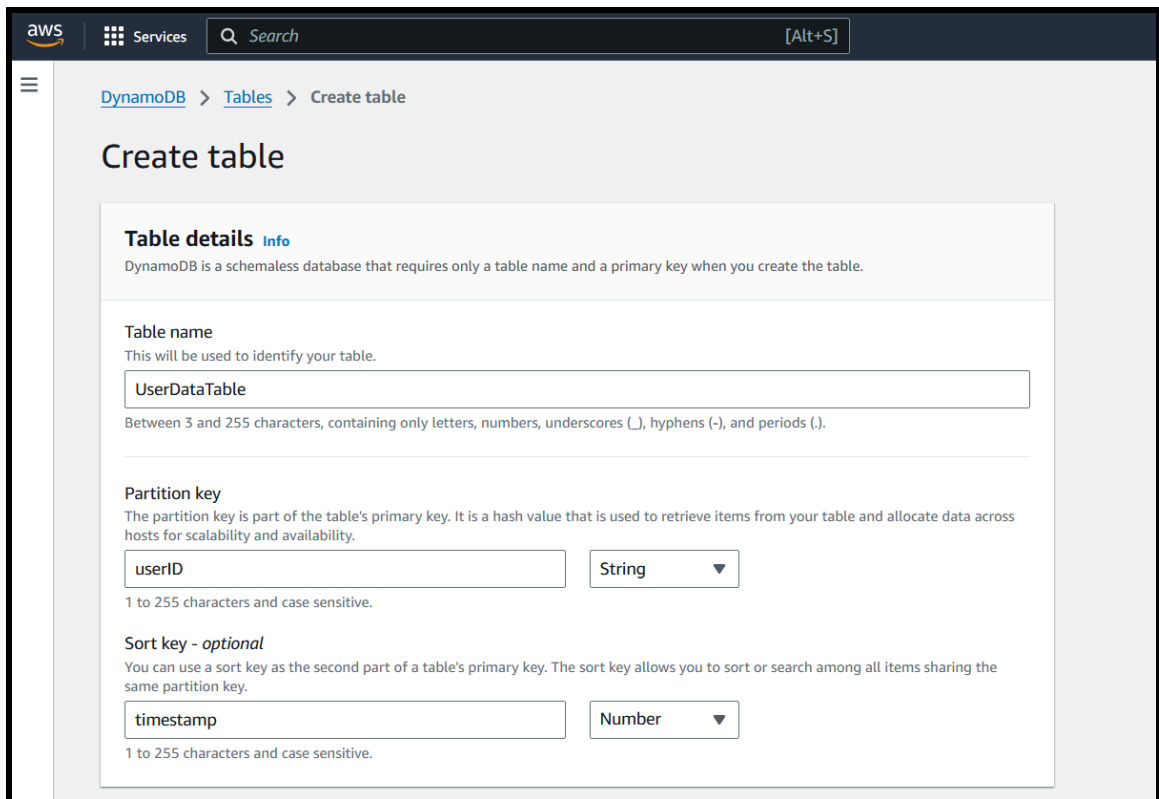
### Practical Applications

- User activity tracking systems
- E-commerce order processing
- IoT device data collection
- Log file processing and analysis

## 2. Step-by-Step Implementation

## Step 1: Creating the DynamoDB Table

1. Navigate to AWS DynamoDB Console
2. Create a new table with the following configuration:  
Table Name: UserDataTable  
Partition Key: userID (String)  
Sort Key: timestamp (Number)
3. Enable auto-scaling for read/write capacity



The screenshot shows the AWS DynamoDB console's 'Create table' page. The breadcrumb navigation at the top reads 'DynamoDB > Tables > Create table'. The main heading is 'Create table'. Below this, there is a 'Table details' section with an 'Info' link. A descriptive text states: 'DynamoDB is a schemaless database that requires only a table name and a primary key when you create the table.' The form contains three main sections: 1. 'Table name': A text input field containing 'UserDataTable'. Below the field, a note specifies: 'Between 3 and 255 characters, containing only letters, numbers, underscores (\_), hyphens (-), and periods (.).' 2. 'Partition key': A text input field containing 'userID' and a dropdown menu set to 'String'. A note below states: 'The partition key is part of the table's primary key. It is a hash value that is used to retrieve items from your table and allocate data across hosts for scalability and availability. 1 to 255 characters and case sensitive.' 3. 'Sort key - optional': A text input field containing 'timestamp' and a dropdown menu set to 'Number'. A note below states: 'You can use a sort key as the second part of a table's primary key. The sort key allows you to sort or search among all items sharing the same partition key. 1 to 255 characters and case sensitive.'

The screenshot shows the AWS IAM console configuration for a new user. At the top, there are tabs for 'Provisioned' and 'On-demand'. The 'Provisioned' tab is selected, with a description: 'Manage and optimize your costs by allocating read/write capacity in advance.' The 'On-demand' tab is also visible, with a description: 'Simplify billing by paying for the actual reads and writes your application performs.'

Below the tabs, there are two sections: 'Read capacity' and 'Write capacity'. Each section has an 'Auto scaling' toggle set to 'On' and an 'Info' link. The description for both is 'Dynamically adjusts provisioned throughput capacity on your behalf in response to actual traffic patterns.'

For 'Read capacity', the 'Minimum capacity units' is 1, 'Maximum capacity units' is 10, and 'Target utilization (%)' is 70. For 'Write capacity', the 'Minimum capacity units' is 1, 'Maximum capacity units' is 10, and 'Target utilization (%)' is 70.

## Step 2: Setting up the S3 Bucket

Create a new S3 bucket:

Bucket Name: user-data-processing-bucket  
Region: Asia Pacific (Sydney) ap-southeast-2

1. Access: Private
2. Enable versioning for data consistency

## Create bucket [Info](#)

Buckets are containers for data stored in S3.

### General configuration

AWS Region

Asia Pacific (Sydney) ap-southeast-2

Bucket name [Info](#)

user-data-processing-bucket

Bucket name must be unique within the global namespace and follow the bucket naming rules. [See rules for bucket naming](#) [↗](#)

Copy settings from existing bucket - *optional*

Only the bucket settings in the following configuration are copied.

[Choose bucket](#)

Format: s3://bucket/prefix

### Bucket Versioning

Versioning is a means of keeping multiple variants of an object in the same bucket. You can use versioning to preserve, retrieve, and restore every version of every object stored in your Amazon S3 bucket. With versioning, you can easily recover from both unintended user actions and application failures. [Learn more](#) [↗](#)

Bucket Versioning

☐ Disable

☒ Enable

## Step 3: Creating the Lambda Function

- Create a new Lambda function:

aws

Services

Search

[Alt+S]

presets for common use cases.

### Basic information

**Function name**  
Enter a name that describes the purpose of your function.

ProcessS3JSONToDynamoDB

Function name must be 1 to 64 characters, must be unique to the Region, and can't include spaces. Valid characters are a-z, A-Z, 0-9, hyphens (-), and underscores (\_).

**Runtime** [Info](#)  
Choose the language to use to write your function. Note that the console code editor supports only Node.js, Python, and Ruby.

Node.js 20.x

↻

**Architecture** [Info](#)  
Choose the instruction set architecture you want for your function code.

☒ x86\_64

☐ arm64

**Permissions** [Info](#)  
By default, Lambda will create an execution role with permissions to upload logs to Amazon CloudWatch Logs. You can customize this default role later when adding triggers.

► Change default execution role

Add the code to Lambda function:

✔ Successfully updated the function ProcessS3JSONToDynamoDB.

### Code source

[Info](#)

Upload from ▼

File Edit Find View Go Tools Window Test Deploy

Go to Anything (Ctrl-P)

Environment

ProcessS3JSONToDynamoDB

index.mjs

```
1 const AWS = require('aws-sdk');
2 const dynamoDB = new AWS.DynamoDB.DocumentClient();
3 const s3 = new AWS.S3();
4
5 exports.handler = async (event) => {
6   try {
7     // Get the S3 bucket and file details from the event
8     const bucket = event.Records[0].s3.bucket.name;
9     const key = decodeURIComponent(event.Records[0].s3.object.key.replace(/\+/g, ' '));
10
11     // Get the JSON file from S3
12     const response = await s3.getObject({ Bucket: bucket, Key: key }).promise();
13     const jsonData = JSON.parse(response.Body.toString());
14
15     // Prepare DynamoDB item
16     const item = {
17       userID: jsonData.userID,
18       timestamp: Date.now(),
19       data: jsonData.data || {},
20       source: key
21     };
22
23     // Write to DynamoDB
24     await dynamoDB.put({
25       TableName: 'UserDataTable',
26       Item: item
27     }).promise();
28
29     return {
30       statusCode: 200,
31       body: JSON.stringify({ message: 'Data processed successfully' })
32     };
33   } catch (error) {
34     console.error('Error:', error);
35     throw error;
36   }
37 }
```

## Step 4: Configuring Permissions

1. Create IAM Role for Lambda with policies:
  - AWSLambdaBasicExecutionRole
  - Custom policy for S3 access
  - Custom policy for DynamoDB access

The screenshot shows the AWS IAM console 'Create role' page, specifically Step 3: Name, review, and create. The left sidebar shows the navigation path: IAM > Roles > Create role. The main content area is titled 'Name, review, and create'. Under 'Role details', the 'Role name' field is filled with 'LambdaS3DynamoDBRole'. The 'Description' field contains the text 'Allows Lambda functions to call AWS services on your behalf.' Below this, 'Step 1: Select trusted entities' is shown with an 'Edit' button. The 'Trust policy' section displays a JSON snippet: 

```
1 {
2   "Version": "2012-10-17",
3   "Statement": [
4     {
5       "Effect": "Allow",
```

This section is titled 'Existing role'. It instructs the user to 'Choose an existing role that you've created to be used with this Lambda function. The role must have permission to upload logs to Amazon CloudWatch Logs.' A dropdown menu shows 'LambdaS3DynamoDBRole' with a downward arrow. To the right is a refresh button (circular arrow icon). Below the dropdown, a link states: 'View the LambdaS3DynamoDBRole role on the IAM console.'

## Step 5: Setting up S3 Trigger

1. Add S3 trigger to Lambda function
2. Configure for ObjectCreated events
3. Specify .json file suffix filter

### Destination

**i** Before Amazon S3 can publish messages to a destination, you must grant the Amazon S3 principal the necessary permissions to call the relevant API to publish messages to an SNS topic, an SQS queue, or a Lambda function. [Learn more](#)

Destination

Choose a destination to publish the event. [Learn more](#)

☒ **Lambda function**  
Run a Lambda function script based on S3 events.

☐ **SNS topic**  
Fanout messages to systems for parallel processing or directly to people.

☐ **SQS queue**  
Send notifications to an SQS queue to be read by a server.

Specify Lambda function

☒ Choose from your Lambda functions

☐ Enter Lambda function ARN

Lambda function

ProcessS3JSONToDynamoDB ▼

CancelSave changes

## 3. Demonstration Preparation

### Key Points to Cover

1. Architecture Overview
  - Explain serverless benefits
  - Describe event-driven workflow
2. Implementation Details
  - Show DynamoDB table structure
  - Explain Lambda function code
  - Demonstrate S3 trigger configuration
3. Testing Process
  - Upload sample JSON file
  - Show real-time processing
  - Verify DynamoDB entries

## Sample Test Data

Json

```
{
  "userID": "user123",
  "data": {
    "name": "Viven Hotwani",
    "email": "viven.hotwani@example.com",
    "action": "login"
  }
}
```

## Potential Questions and Answers

1. Q: How does the system handle concurrent uploads? A: Lambda automatically scales to handle multiple concurrent executions.
2. Q: What happens if the JSON is malformed? A: Error handling in Lambda catches parsing errors and logs them in CloudWatch.
3. Q: How can we monitor processing failures? A: Through CloudWatch Logs and Metrics, and by setting up CloudWatch Alarms.

## 4. Certification Requirements

### AWS Certification Deadline

- Submission deadline: October 21st, 2024
- Required certification: AWS Certified Developer Associate

### Importance of Certification

- Validates understanding of AWS services
- Demonstrates practical implementation skills
- Required for project evaluation
- Enhances career prospects in cloud computing

### Preparation Tips

1. Complete AWS practice exams
2. Review serverless architecture patterns
3. Understand DynamoDB design principles
4. Practice Lambda function development