Cloud-Based Expense Splitting App using AWS Lambda and RDS

A Course Project Report Submitted in partial fulfillment of the course requirements for the award of grades in the subject of

CLOUD BASED AIML SPECIALITY
(22SDCS07A)

by

Namavarapu Subrahmanya Sai Pankaj

2210030450

Under the esteemed guidance of

Ms. P. Sree Lakshmi
Assistant Professor,
Department of Computer Science and Engineering

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

K L Deemed to be UNIVERSITY

Aziznagar, Moinabad, Hyderabad,
Telangana, Pincode: 500075

April 2025

K L Deemed to be UNIVERSITY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Certificate

This is Certified that the project entitled **Cloud-Based Expense Splitting App using AWS Lambda and RDS** which is a experimental &/ theoretical &/ Simulation&/ hardware work carried out by NSS Pankaj **2210030450** in partial fulfillment of the course requirements for the award of grades in the subject of **CLOUD BASED AIML SPECIALITY**, during the year **2024-2025**. The project has been approved as it satisfies the academic requirements.

Ms.P.Sree Lakshmi	**Dr. Arpita Gupta**
Course Coordinator	**Head of the Department**

Ms. P. Sree Lakshmi

Course Instructor

CONTENTS

Page No.

1. INTRODUCTION

The Cloud-Based Expense Splitting App using AWS Lambda and RDS is designed to efficiently manage shared expenses through a modern, serverless architecture. The backend leverages AWS Lambda, which runs serverless functions triggered by HTTP API requests without requiring traditional server management [1][2]. Application configurations such as database credentials are securely managed through Lambda environment variables [3].

Amazon RDS for MySQL provides the scalable and reliable relational database backend, enabling structured storage and easy querying of expense data [4][5]. For frontend integration, AWS API Gateway is used to expose Lambda functions via RESTful APIs, handling routing, authorization, and CORS settings [6][7].

The project is built and deployed entirely through AWS CLI, offering full automation, speed, and control during the development and deployment processes [8]. Meanwhile, Amazon S3 serves as the hosting platform for static frontend content like HTML forms, ensuring high availability and fast access to users globally [9].

By combining these AWS services, the project demonstrates how serverless technologies can simplify development, reduce operational overhead, and offer scalable, real-world applications suited for modern cloud-native solutions.

2. AWS Services Used as part of the project

AWS Lambda:

Serverless Compute – Acts as the backend logic for inserting user-submitted expense data into the RDS MySQL database.
Pay-per-Use Model – Charges only for the compute time consumed during function invocation, making it cost-efficient.
Scalability – Automatically scales based on incoming HTTP requests via API Gateway.
Easy Integration – Seamlessly integrates with API Gateway, RDS, CloudWatch, and S3.

Amazon RDS (Relational Database Service):

MySQL Database Hosting – Hosts the persistent database (Pexpenses) with a structured table (Ptransactions) for storing expense data.
Managed Service – AWS handles patching, backups, and maintenance, ensuring high availability and reliability.
Security – Credentials and network access are managed using IAM and VPC settings.
Query Support – Supports SQL operations for data insertion and future reporting use cases.

Amazon S3 (Simple Storage Service):

Static Website Hosting – Hosts the HTML frontend form for users to input expense data.
Public Access Control – Configured via bucket policy and ACLs to allow form submission securely.
Durability & Availability – Ensures reliable access to the frontend with 99.999999999% durability.
API Integration – The form integrates with API Gateway endpoint to trigger Lambda functions.

Amazon API Gateway:

HTTP Endpoint Creation – Provides RESTful endpoints that act as triggers for the Lambda function.
Security & Authorization – Supports method-level control and CORS for browser-based frontend access.
Throttling & Monitoring – Manages traffic load and logs API usage through CloudWatch integration.
Frontend Linking – Acts as the middleware between the S3-hosted form and the backend Lambda function.

AWS IAM (Identify and Access Management):

Role-Based Access – IAM roles ensure that Lambda has permission to connect to RDS and log to CloudWatch.
Policy Enforcement – JSON-based policies define access control for each service used.
Security Compliance – Ensures only authorized services and users can perform operations.
Custom Execution Role – A specific role (PLambdaRDSRole) is assigned to Lambda for managing secure backend operations.

3. Steps involved in solving project problem statement

Step 1: Design Database Schema in Amazon RDS

- Create an S3 bucket, e.g., expense-tracker-data-bucket
- Define a table Ptransactions with the following fields:
- id (INT, AUTO_INCREMENT, Primary Key)
- name (VARCHAR)
- amount (DECIMAL)

Step 2: Upload Data to S3

- Create a Lambda function (PExpenseLambda) with runtime set to Python 3.9.
- Package and deploy code that connects to RDS using PyMySQL and inserts form data into Ptransactions.
- Include required dependencies inside a ZIP file using CLI and upload via:

```
aws lambda update-function-code \
--function-name PExpenseLambda \
 --zip-file fileb://function.zip\
```

Step 3: Create REST API with API Gateway (via CLI)

- Set up a REST API (`PExpenseAPI`) using: aws apigateway create-rest-api --name "PExpenseAPI"
- Create resources and methods (e.g., `/submit`) and integrate them with the Lambda function using CLI.

Step 4: Host HTML Form on Amazon S3

- Design a static index.html form to collect user input: Fields: Name, Amount, Phone Number, Due Date
- Create an S3 bucket (e.g., pexpensesplitter-html) and enable static website hosting:

aws s3api create-bucket --bucket pexpensesplitter-html --region us-east-1

aws s3 website s3://pexpensesplitter-html/ --index-document index.html aws

s3 cp index.html s3://pexpensesplitter-html/index.html --acl public-read

Step 5: Link Form to API Gateway Endpoint

- Design a static index.html form to collect user input:

<form method="POST" action="https://<api-id>.execute-api.us-east-1.amazonaws.com/prod/submit">

- Ensure CORS is configured for POST requests in API Gateway.

Step 6: Test and Verify End-to-End Flow

- Design a static index.html form to collect user input:
- Enter details and submit the form.
- Verify the data is correctly inserted into the RDS table Ptransactions.
- Use Lambda logs in CloudWatch to debug or validate function execution

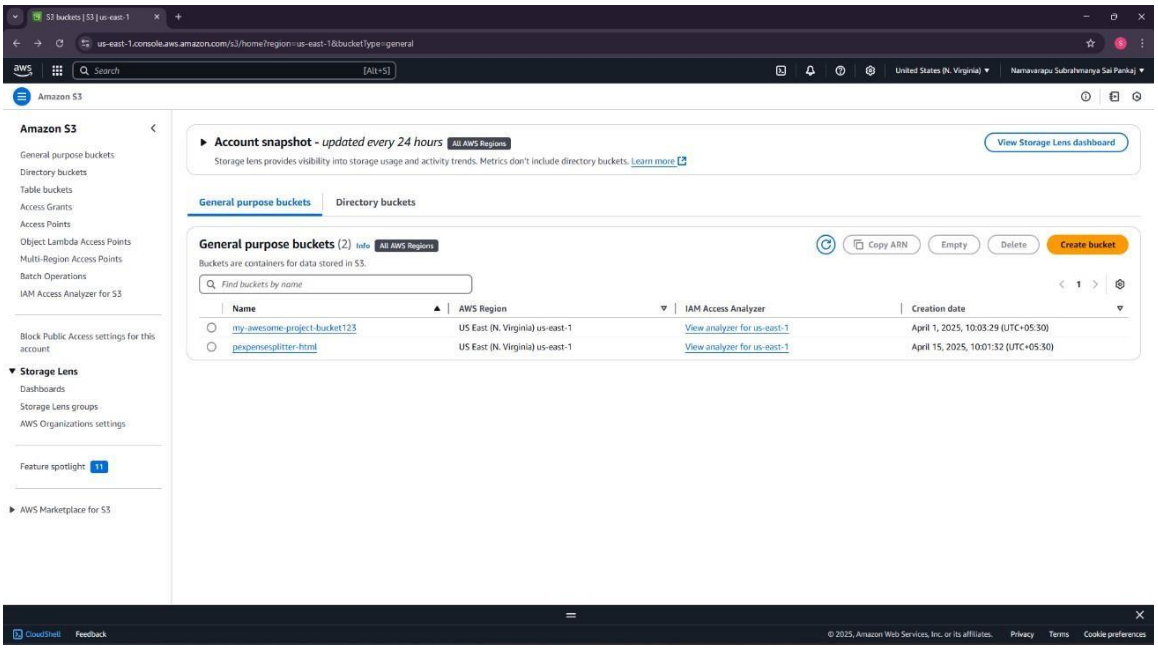
Step 7: Extend to Multiple Users (Optional Enhancement

- Allow the same system to accept inputs from multiple users.
- Filter or visualize entries user-wise using a dashboard in the future.

4. Stepwise Screenshots with brief description

Step 1: Creating the S3 Bucket

- Create an Amazon S3 bucket named expensetrackeramazon using the AWS Management Console or CLI.
- This bucket will act as the central storage for your expense dataset and related configuration files.
- Advantages: Scalability: Easily accommodates growing volumes of expense data.
- Reliability: Ensures durable and secure storage with 99.999999999% data availability.


S3 buckets | S3 | us-east-1
us-east-1.console.aws.amazon.com/s3/home?region=us-east-1&bucketType=general
Search [Alt+S]
United States (N. Virginia)
Namavarapu Subrahmanya Sai Pankaj
Amazon S3
Amazon S3
General purpose buckets
Directory buckets
Table buckets
Access Grants
Access Points
Object Lambda Access Points
Multi-Region Access Points
Batch Operations
IAM Access Analyzer for S3
Block Public Access settings for this account
Storage Lens
Dashboards
Storage Lens groups
AWS Organizations settings
Feature spotlight 11
AWS Marketplace for S3
View Storage Lens dashboard
General purpose buckets
Directory buckets
Info
All AWS Regions
Copy ARN
Empty
Delete
Create bucket
Find buckets by name
CloudShell
Feedback
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences

Account snapshot - *updated every 24 hours* All AWS Regions

Storage lens provides visibility into storage usage and activity trends. Metrics don't include directory buckets. Learn more

General purpose buckets (2)

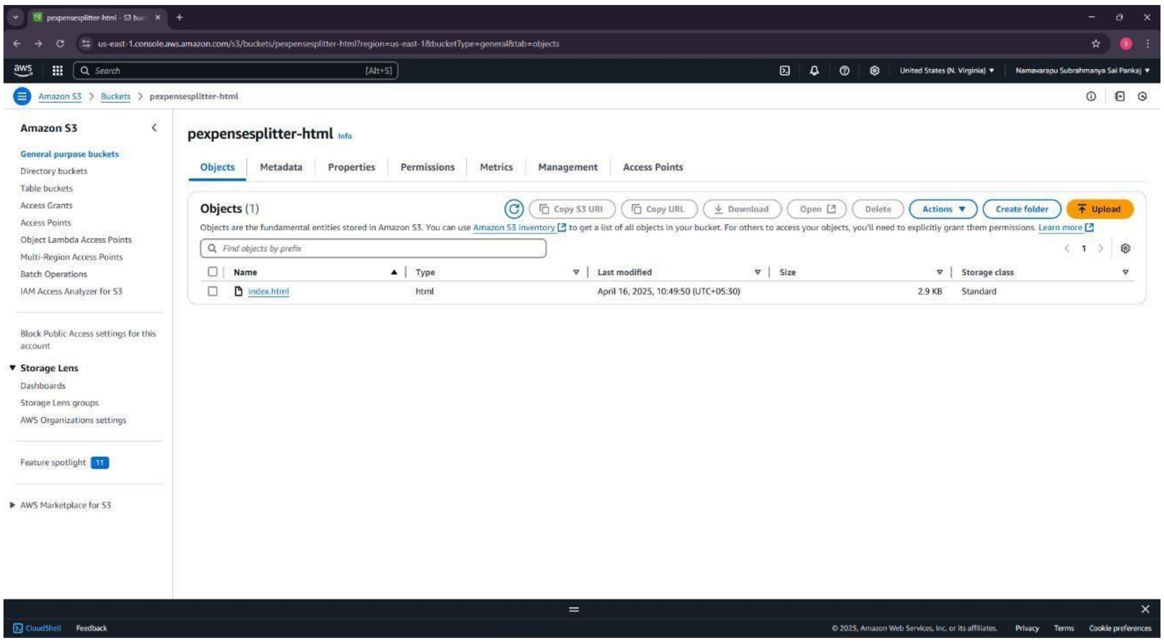
Buckets are containers for data stored in S3.

Name	AWS Region	IAM Access Analyzer	Creation date
my-awesome-project-bucket123	US East (N. Virginia) us-east-1	View analyzer for us-east-1	April 1, 2025, 10:03:29 (UTC+05:30)
pexpensesplitter-html	US East (N. Virginia) us-east-1	View analyzer for us-east-1	April 15, 2025, 10:01:32 (UTC+05:30)

Fig. 4.1: Creating S3 bucket

Step 2: Adding Files to the S3 Bucket

- Uploaded the following files to the S3 bucket: `index.html` – A simple frontend form with fields for Name, Amount, Phone Number, and Due Date.
- Granted public read access to the HTML file using S3 bucket policies. □ Verified deployment by accessing the form via the public S3 static website URL.



pexpensesplitter-html

Objects (1)

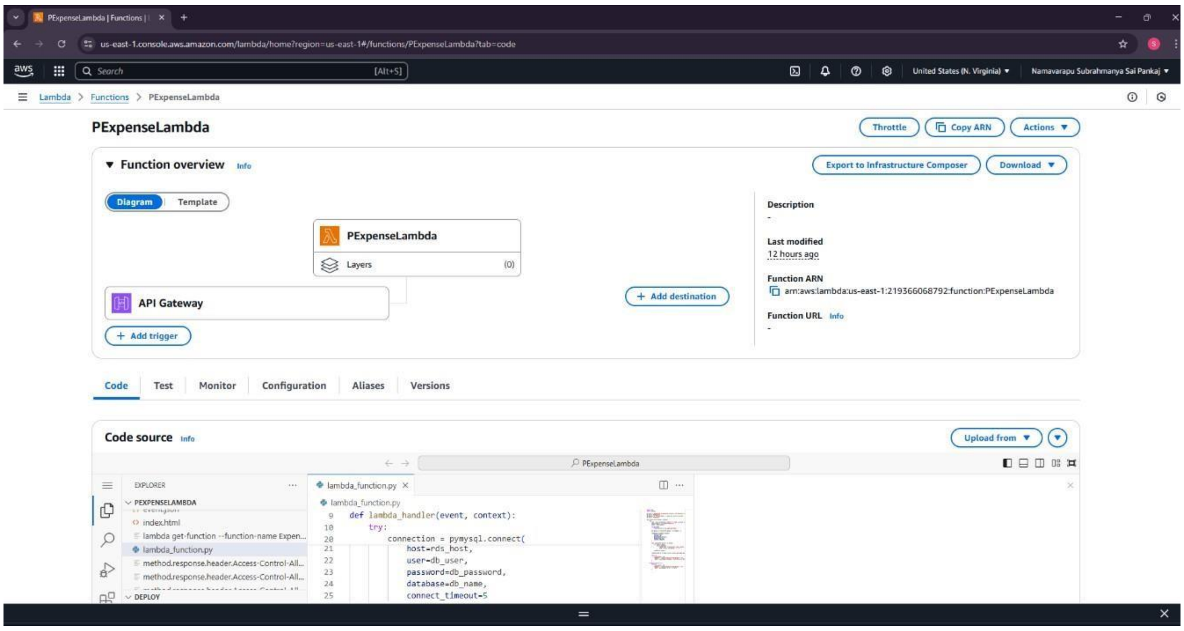
Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

Name	Type	Last modified	Size	Storage class
index.html	html	April 16, 2025, 10:49:50 (UTC+05:30)	2.9 KB	Standard

Fig. 4.2: Uploading `index.html` to the S3 bucket

Step 3: Setting Up AWS Lambda and RDS Integration

- Created a Lambda function named PExpenseLambda using Python 3.9.
- Configured the function to: Parse incoming JSON from API Gateway. Insert data into the `Ptransactions` table in the RDS MySQL database.
- Used `PyMySQL` library bundled with function.zip and uploaded using AWS CLI.



PExpenseLambda

Function ARN: arn:aws:lambda:us-east-1:219366068792:function:PExpenseLambda

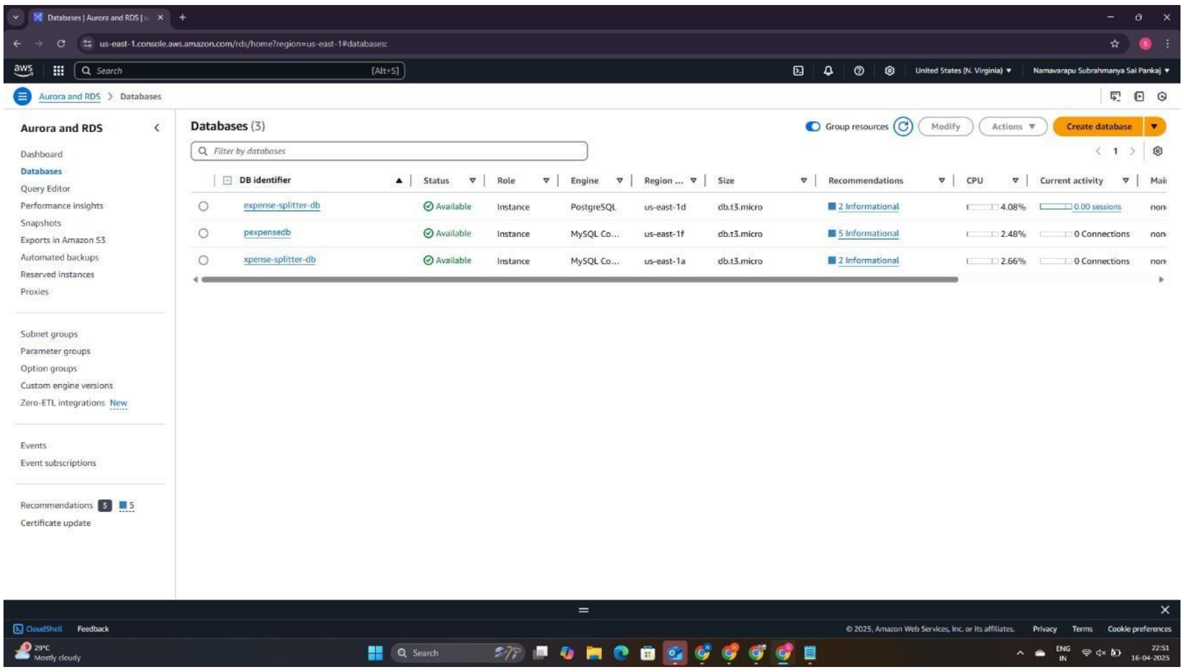
Last modified: 12 hours ago

```
def lambda_handler(event, context):
    try:
        connection = pymysql.connect(
            host=rds_host,
            user=db_user,
            password=db_password,
            database=db_name,
            connect_timeout=5
```

Fig. 4.3: Lambda function configuration to interact with RDS

Step 4: Creating RDS MySQL Database

- Provisioned an RDS instance named pexpensedb with credentials: Username: `Padminuser`, Password: `Ppassword123!`
- Created database `Pexpenses` and table `Ptransactions` with fields
- Used `PyMySQL` library bundled with function.zip and uploaded using AWS CLI.



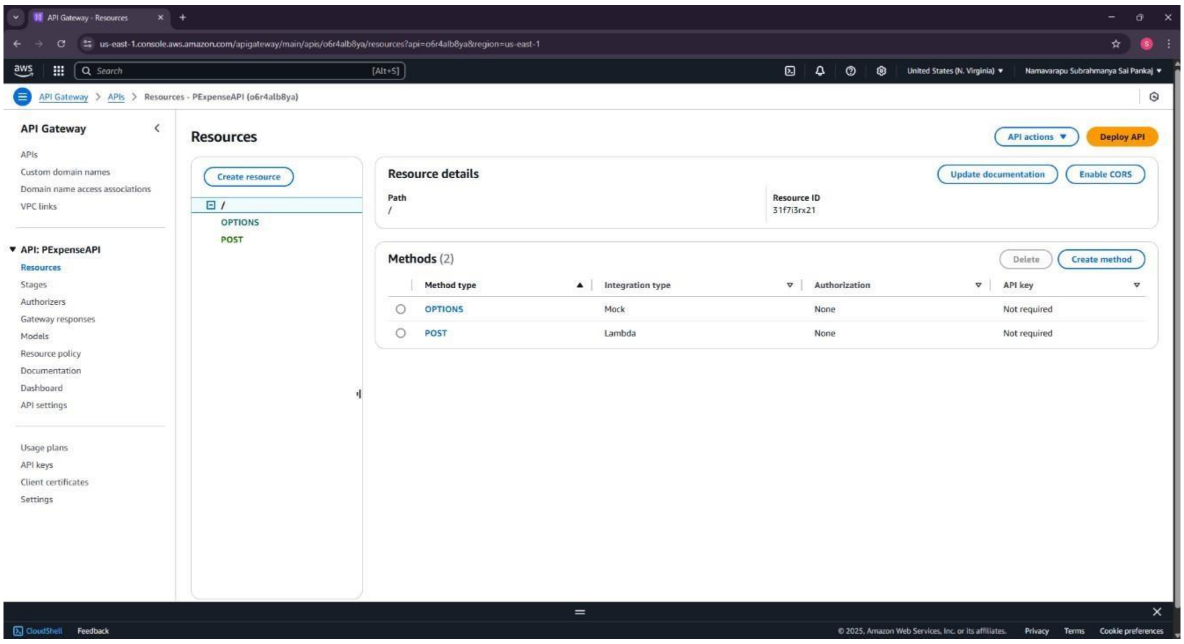
Databases (3)

DB identifier	Status	Role	Engine	Region ...	Size	Recommendations	CPU	Current activity	Mai
expense-splitter-db	Available	Instance	PostgreSQL	us-east-1d	db.t3.micro	2 Informational	4.08%	0.00 sessions	non
pexpensedb	Available	Instance	MySQL Co...	us-east-1f	db.t3.micro	5 Informational	2.48%	0 Connections	non
xpense-splitter-db	Available	Instance	MySQL Co...	us-east-1a	db.t3.micro	2 Informational	2.66%	0 Connections	non

Fig. 4.4: Table structure in RDS

Step 5: Deploying API Gateway

- Created a REST API named `PExpenseAPI` with a POST method at resource path `/submit`.
- Integrated the POST method with `PExpenseLambda` via Lambda Proxy integration.
   Used `PyMySQL` library bundled with function.zip and uploaded using AWS CLI.



Resource details

Path	Resource ID
/	31f7i3rx21

Methods (2)

Method type	Integration type	Authorization	API key
OPTIONS	Mock	None	Not required
POST	Lambda	None	Not required

Fig. 4.5: API Gateway deployment for Lambda integration

5. Learning Outcomes

Understood how to connect AWS Lambda with RDS and handle real-time data insertion

- Gained practical knowledge on writing Python scripts inside AWS Lambda to parse POST request data and insert it into a MySQL database hosted on Amazon RDS.
- Understood how to use the `PyMySQL` library, package dependencies into a ZIP file, and deploy the Lambda function using AWS CLI.
- Learned to troubleshoot Lambda errors using CloudWatch Logs and correct runtime issues like missing modules and schema mismatches.

Learned how to configure API Gateway to securely trigger backend logic

- Gained hands-on experience in creating and deploying REST APIs using AWS API Gateway.
- Learned to enable CORS configuration for HTML-based clients to communicate with Lambda securely.
- Mastered the process of mapping HTTP POST requests to Lambda invocations with input payload parsing.

Built a fully serverless HTML-to-Database pipeline using AWS Services

- Designed a user-friendly HTML form hosted on Amazon S3 and connected it to the backend via API Gateway.
- Understood how to host static websites using S3 and configure public access using bucket policies.
- Connected the form inputs (Name, Amount, Phone, Due Date) directly to a real- time backend, simulating a fully functional expense tracking workflow.

Managed database access and API permissions using IAM roles

- Configured IAM roles with correct execution permissions for Lambda to access RDS securely without exposing credentials.

- Gained knowledge of cross-service trust relationships and minimal privilege principles in AWS.
- Understood the importance of fine-grained access policies in securing backend databases and APIs.

Deployed and Managed the Entire Application Using AWS CLI

- Gained experience in building a **production-ready serverless application** using only AWS CLI commands without using the AWS console.
- Mastered real-world DevOps tasks like service creation, configuration, deployment, and debugging from the command line.

6. Conclusion

This project demonstrates how AWS Lambda, Amazon RDS, and API Gateway can be combined to create a real-time, serverless Cloud-Based Expense Splitting Application. By utilizing AWS services like S3 for static website hosting, users can easily submit expense data via a user-friendly HTML form. This triggers a POST request through API Gateway, invoking a Lambda function that writes data into an RDS MySQL database using the PyMySQL library. The architecture is scalable, cost-effective, and maintenance-free, offering seamless integration for dynamic expense tracking and real-time collaboration. IAM roles ensure secure access control, while future enhancements could include generating summaries, sending notifications, and visualizing spending habits with Amazon QuickSight. The project showcases the power of AWS to build secure, reliable, and event-driven applications.

7. Future Enhancements

This project showcases how AWS Lambda, Amazon RDS, and API Gateway can be seamlessly integrated to build a real-time, serverless Cloud-Based Expense Splitting Application. Using Amazon S3 for static website hosting, users interact with an intuitive HTML form to submit expense data, which triggers a POST request via API Gateway and invokes a Lambda function. This function, leveraging PyMySQL, stores the data in an RDS MySQL database, enabling dynamic and efficient expense tracking. The architecture is lightweight, auto-scalable, cost-effective, and secured using IAM roles to manage access across services. Looking ahead, this project can be extended to include advanced features such as automated email/SMS notifications for due payments, visual dashboards using Amazon QuickSight, user authentication with Amazon Cognito, and integration with payment gateways to facilitate real-time settlements. These future enhancements will not only boost user experience but also establish a more comprehensive, intelligent, and collaborative financial management solution

8. References

1. https://docs.aws.amazon.com/lambda/latest/dg/welcome.html

2. https://docs.aws.amazon.com/lambda/latest/dg/python-handler.html

3. https://docs.aws.amazon.com/lambda/latest/dg/configuration-envvars.html

4. https://docs.aws.amazon.com/rds/latest/MySQLUserGuide/Welcome.html

5. https://docs.aws.amazon.com/rds/latest/UserGuide/CHAP_GettingStarted.html

6. https://docs.aws.amazon.com/apigateway/latest/developerguide/apigateway-rest

7. https://docs.aws.amazon.com/apigateway/latest/developerguide/how-to-cors.html

8. https://docs.aws.amazon.com/cli/latest/userguide/cli-services-lambda.html

9. https://docs.aws.amazon.com/AmazonS3/latest/userguide/Welcome.html