Cloud-Based Task Management System with AWS Step Functions

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

D .Gayathri Aneesha

2210030177

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 Task Management System with AWS Step Functions**”** which is a Experimental work carried out by D. Gayathri Aneesha (2210030177), 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

Course Coordinator

Dr. Arpita Gupta

Head of the Department

Ms. P. Sree Lakshmi

Course Instructor

CONTENTS

1. INTRODUCTION

The Cloud-Based Task Management System is a modern, scalable solution that uses AWS Step Functions to automate and manage business processes through serverless architecture. It replaces manual coordination with automated workflows, significantly enhancing operational efficiency and accuracy. This system leverages AWS Lambda to process business logic, while Step Functions orchestrate the flow, allowing seamless communication between components.

Built with a focus on modularity and fault tolerance, the system ensures each task is independently executed and tracked, making it ideal for enterprises needing high reliability and flexibility [4]. The system provides visual workflows, error handling, and parallel execution support, enabling developers and operations teams to visualize task sequences and manage complex scenarios efficiently.

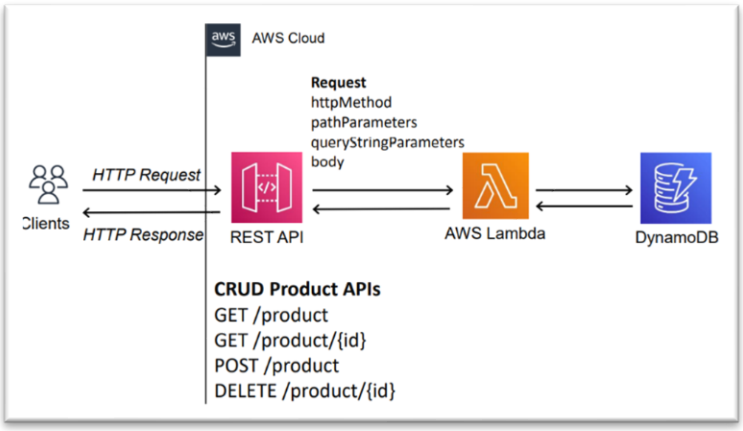
By integrating services such as Amazon DynamoDB for data management, IAM for secure authentication , and CloudWatch for monitoring, the system supports real-time task tracking and secure execution. This serverless approach ensures low operational overhead, rapid deployment, and effortless scalability, making the Cloud-Based Task Management System a powerful tool for organizations looking to optimize their backend processes using cloud-native technologies.

2. AWS Services Used as part of the project

The project utilizes several AWS services to build a serverless architecture:

1. AWS Lambda:
    - Executes business logic for each step.
    - Processes input/output and updates task states
    - Interacts with DynamoDB and other services [1].
    - Automatically scales to meet workload demands.

2. Amazon API Gateway:
    - Exposes RESTful endpoints for task management .
    - Enables secure communication between clients and AWS services.
    - Supports throttling and monitoring of API requests.
    - Acts as an entry point to invoke backend Lambda functions.



3. Amazon DynamoDB:
    - A NoSQL database used to store task details such as task ID, status, and owner.
    - Provides scalable and low-latency data access.
    - Supports efficient querying and indexing for rapid results.
    - Ensures high availability and fault tolerance.

4. AWS Identity and Access Management (IAM):
    - Defines permissions and access roles for Lambda and Step Functions .
    - Ensures secure, role-based resource access.
    - Provides fine-grained access control for various services.
    - Logs API calls using AWS CloudTrail for auditing purposes.

5. Amazon S3 :
    - Stores task attachments, logs, and exports.
    - Provides scalable and reliable storage.
    - Offers built-in versioning and lifecycle rules.
    - Supports encryption and access control policies

6. Amazon CloudWatch:
    - Monitors AWS services and application logs [5].
    - Sends alerts in case of anomalies or execution failures.
    - Helps in setting up metrics and dashboards for tracking.
    - Assists in debugging and system optimization.

3. Steps Involved in Solving the Project Problem Statement

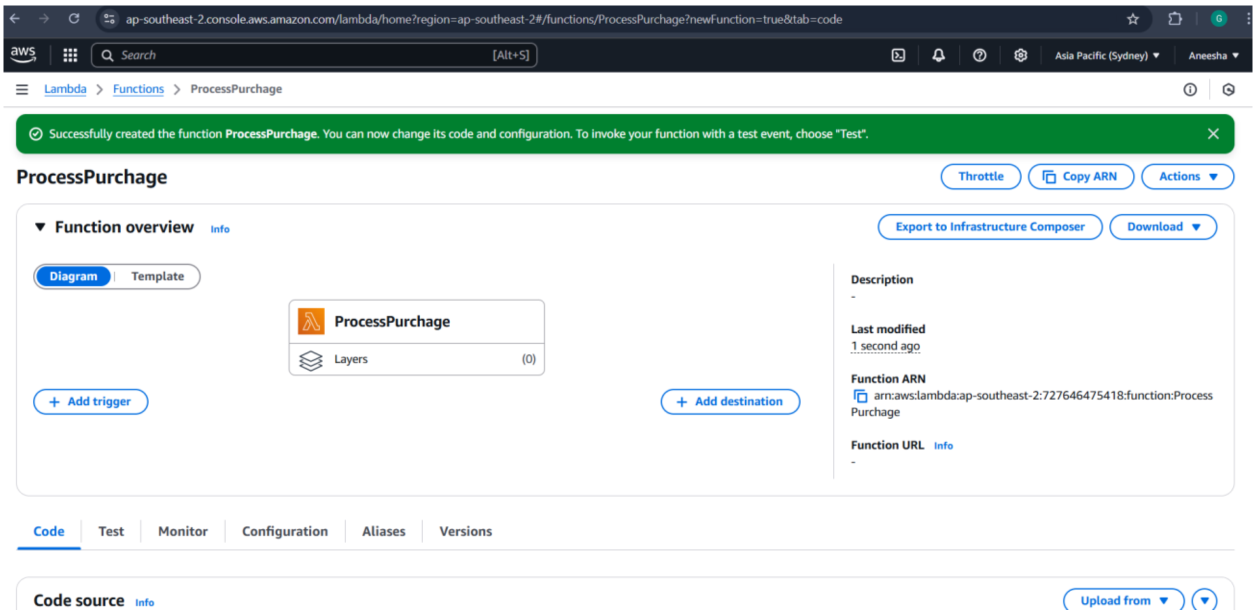
The project aimed to build a Cloud-Based Task Management System using AWS Step Functions with a serverless architecture to automate the processing of tasks. The following steps were taken to implement the solution:

1. Create Lambda Functions:
   - In the AWS Console, go to Lambda > Functions > Create Function .
   - Name the first function ProcessPurchage.
   - Write and upload the source code.
   - Edit runtime settings as needed and deploy.
   - Repeat the steps to create a second function called ProcessRefund with appropriate logic.
2. Configure IAM Role:
   - Navigate to IAM > Dashboard > Roles .
   - Create a new role with the name StepFunctionLambdaRole[7].
   - Assign necessary permissions for Step Functions and Lambda.
   - Ensure the role includes trust relationships for service invocation.
3. Create Step Functions State Machine:
   - Go to Step Functions > State Machines > Create State Machine .
   - Define the workflow using Amazon States Language (ASL).
   - Choose the StepFunctionLambdaRole as the execution role.
   - Successfully create and deploy the state machine and test it.
4. Execute Workflow:
   - Start execution of the state machine.
   - Pass appropriate input parameters as JSON.
   - Monitor real-time status via the console and inspect transitions.
   - Validate output results to ensure logic flows correctly .

5. Validate Output:
   - Check CloudWatch logs for both refund and purchase executions
   - Confirm successful processing and correct outputs.
   - Review metrics such as execution time and resource usage.
   - Ensure that all edge cases are handled appropriately.

4. Stepwise Screenshots with Brief Description

Step 1: Creating Lambda function ProcessPurchage


ap-southeast-2.console.aws.amazon.com/lambda/home?region=ap-southeast-2#/functions/ProcessPurchage?newFunction=true&tab=code

Lambda > Functions > ProcessPurchage

Successfully created the function **ProcessPurchage**. You can now change its code and configuration. To invoke your function with a test event, choose "Test".

ProcessPurchage

Function overview Info

Description
-

Last modified
1 second ago

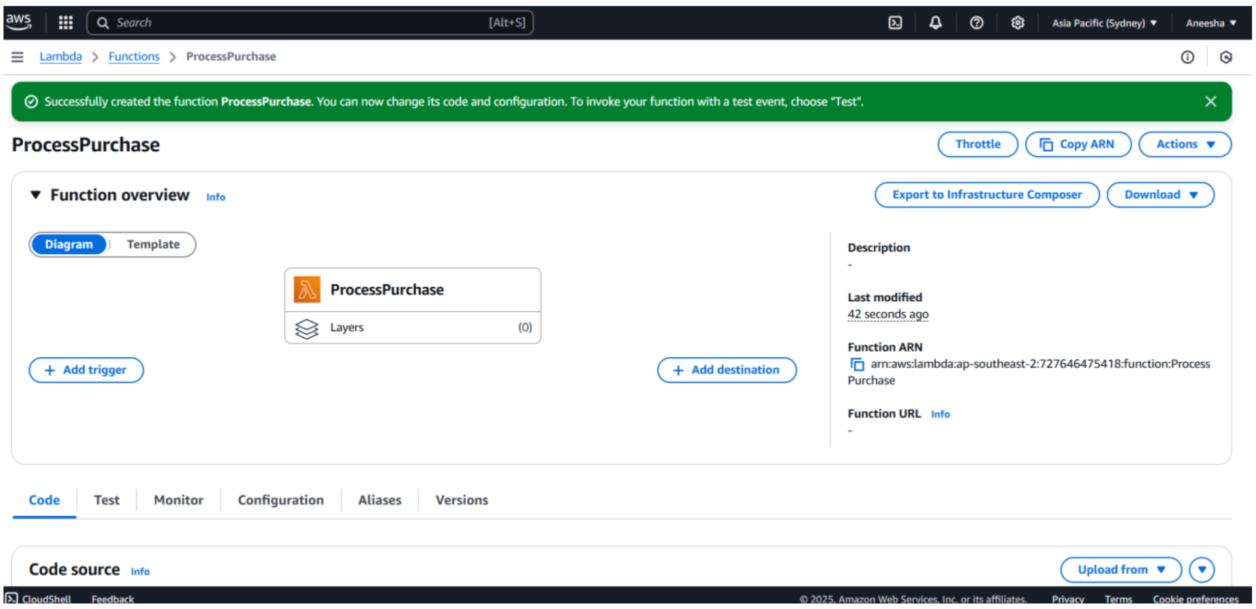
Function ARN
arn:aws:lambda:ap-southeast-2:727646475418:function:ProcessPurchage

Function URL Info
-

Fig: 4.1: Creating Lambda function ProcessPurchage

This shows the initial setup in AWS Lambda with selected runtime and basic configuration. This function handles purchase logic and is integrated with Step Functions for orchestration [2].

Step 2: Editing source code and runtime settings.


Lambda > Functions > ProcessPurchase

Successfully created the function **ProcessPurchase**. You can now change its code and configuration. To invoke your function with a test event, choose "Test".

ProcessPurchase

Function overview Info

Description
-

Last modified
42 seconds ago

Function ARN
arn:aws:lambda:ap-southeast-2:727646475418:function:ProcessPurchase

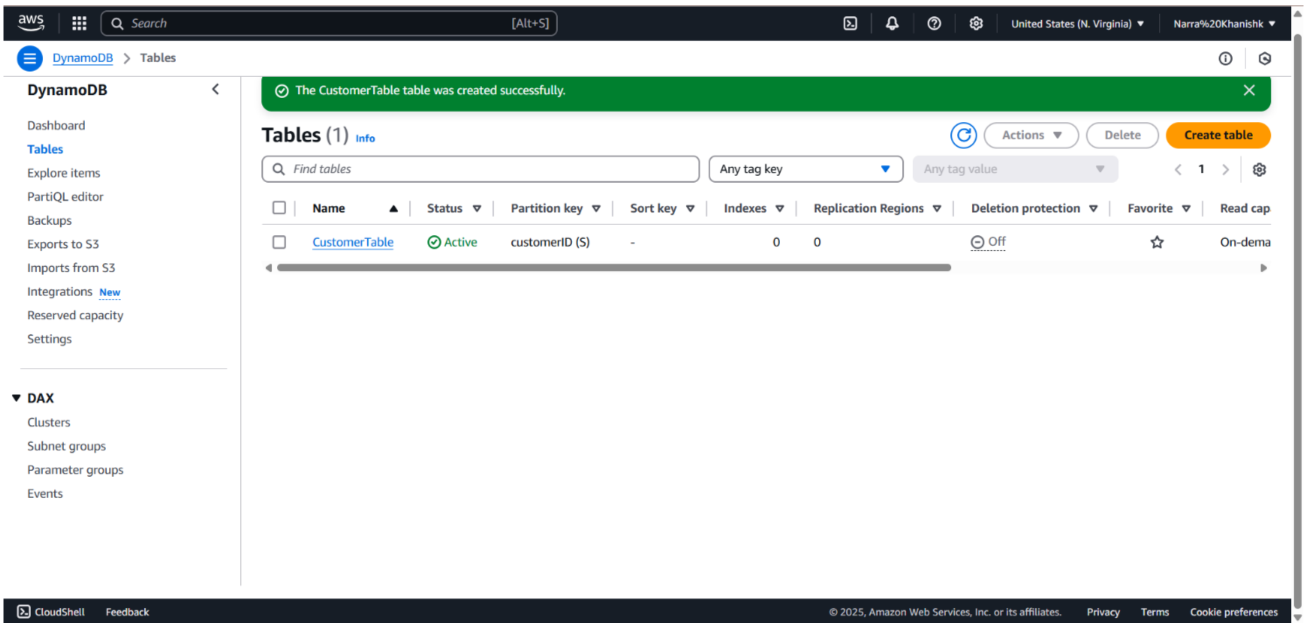
Function URL Info
-

© 2025, Amazon Web Services, Inc. or its affiliates.

Fig: 4.2: Editing source code and runtime settings.

Editing source code and runtime settings – displays how code is uploaded or edited directly and how environment variables are set. This enables proper configuration of logic and environment within the Lambda function .

Step 3: Creating Lambda function ProcessRefund.



The CustomerTable table was created successfully.

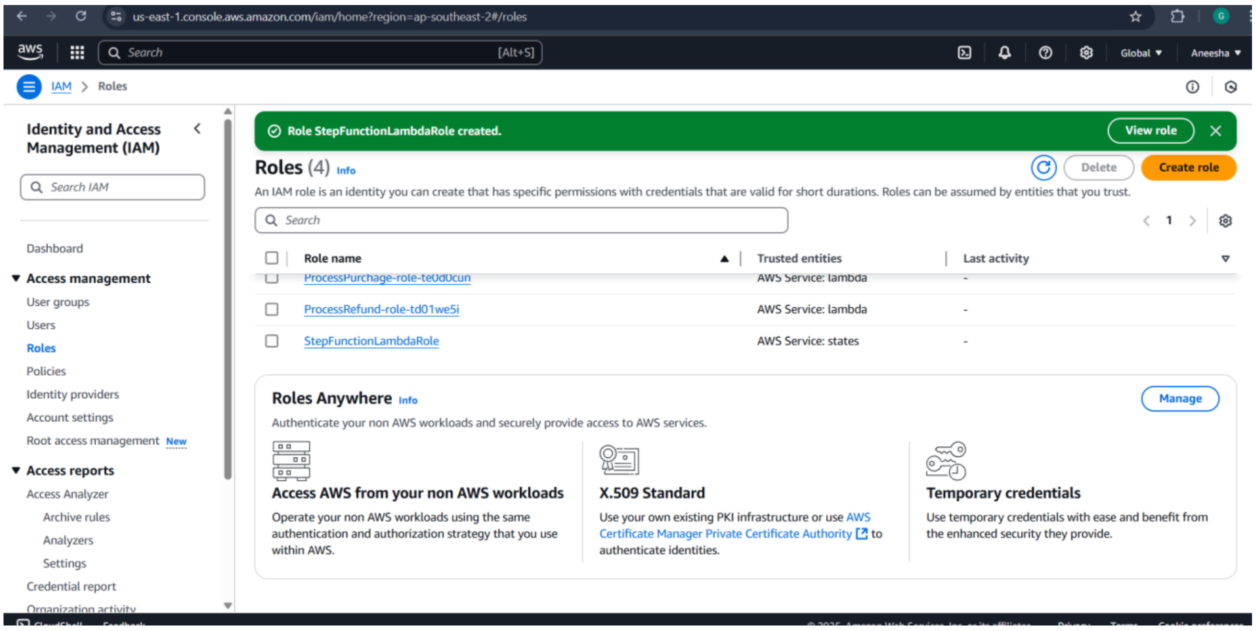
Tables (1) Info

Name	Status	Partition key	Sort key	Indexes	Replication Regions	Deletion protection	Favorite	Read cap
CustomerTable	Active	customerID (S)	-	0	0	Off	☆	On-dema

Fig: 4.3: Lambda Function Creation

Creating Lambda function ProcessRefund – shows the process of setting up another function to handle refund logic. This function is linked with the state machine for conditional execution based on input.

Step 4: Creating IAM role StepFunctionLambdaRole.



Role StepFunctionLambdaRole created.

Roles (4) Info

An IAM role is an identity you can create that has specific permissions with credentials that are valid for short durations. Roles can be assumed by entities that you trust.

Role name	Trusted entities	Last activity
ProcessPurchage-role-te0d0cun	AWS Service: lambda	-
ProcessRefund-role-td01we5i	AWS Service: lambda	-
StepFunctionLambdaRole	AWS Service: states	-

Roles Anywhere Info

Authenticate your non AWS workloads and securely provide access to AWS services.

Access AWS from your non AWS workloads
Operate your non AWS workloads using the same authentication and authorization strategy that you use within AWS.

X.509 Standard
Use your own existing PKI infrastructure or use AWS Certificate Manager Private Certificate Authority to authenticate identities.

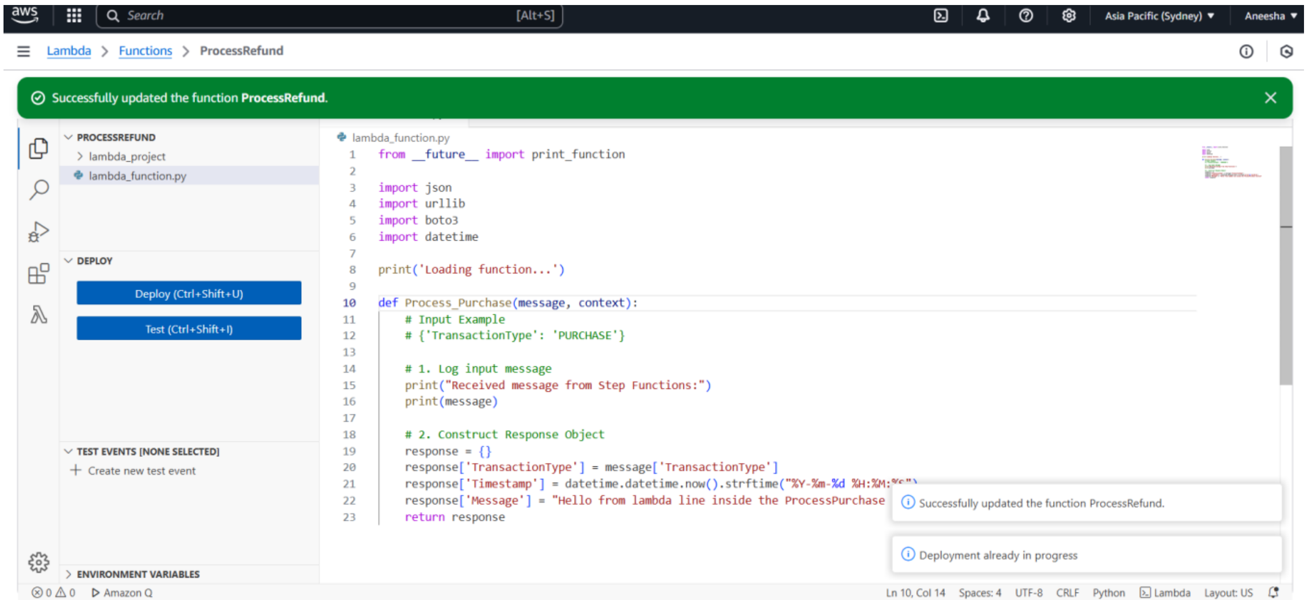
Temporary credentials
Use temporary credentials with ease and benefit from the enhanced security they provide.

Fig: 4.4: Creating IAM role StepFunctionLambdaRole.

This outlines role creation and policy attachment to allow Step Functions to invoke Lambda functions securely. It follows least-privilege access principles for secure service interaction.

Step 5: Defining Step Function state machine and Selecting execution role and creating state machine.

Code:



Lambda > Functions > ProcessRefund

Successfully updated the function **ProcessRefund**.

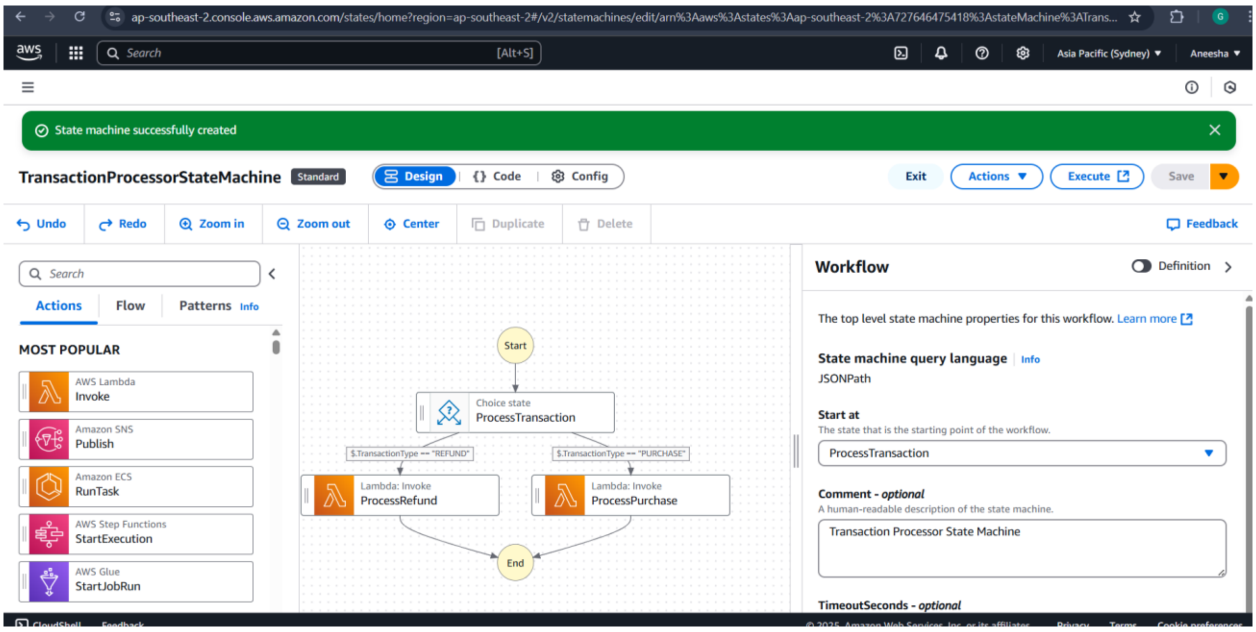
lambda_function.py

```python
from __future__ import print_function

import json
import urllib
import boto3
import datetime

print('Loading function...')

def Process_Purchase(message, context):
    # Input Example
    # {'TransactionType': 'PURCHASE'}

    # 1. Log input message
    print("Received message from Step Functions:")
    print(message)

    # 2. Construct Response Object
    response = {}
    response['TransactionType'] = message['TransactionType']
    response['Timestamp'] = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")
    response['Message'] = "Hello from lambda line inside the ProcessPurchase
    return response
```

Successfully updated the function ProcessRefund.

Deployment already in progress

Fig: 4.5.1: Code



State machine successfully created

TransactionProcessorStateMachine Standard

Workflow

The top level state machine properties for this workflow. Learn more

State machine query language: JSONPath

Start at: ProcessTransaction

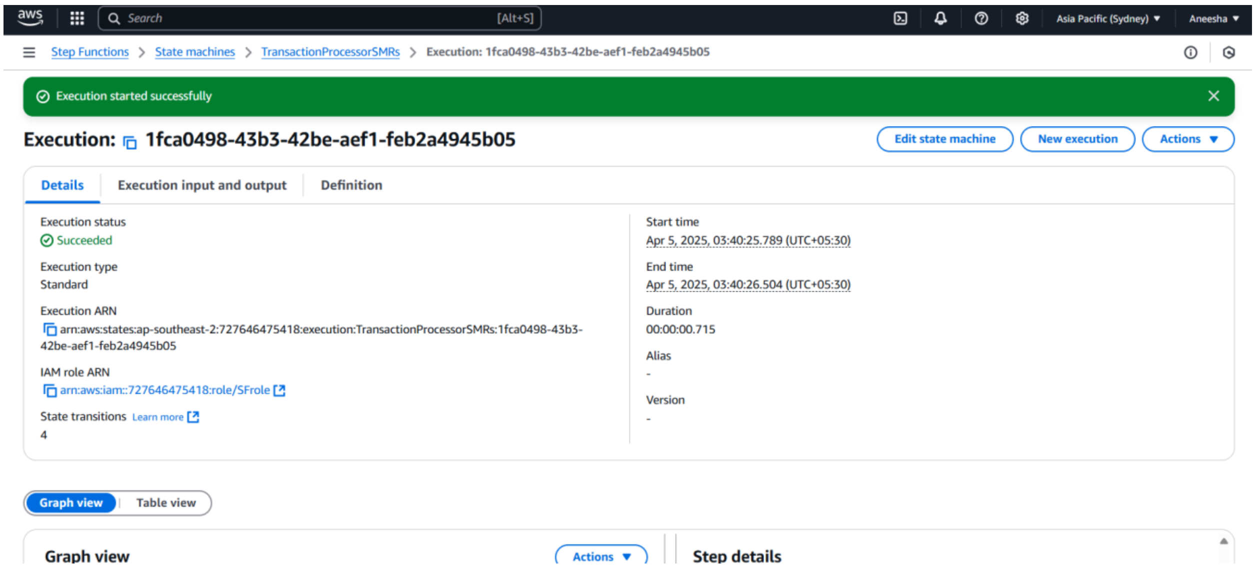
Comment - optional: Transaction Processor State Machine

Design
Code
Config
Exit
Actions
Execute
Save
Undo
Redo
Zoom in
Zoom out
Center
Duplicate
Delete
Feedback
Search
Actions
Flow
Patterns Info
MOST POPULAR
AWS Lambda Invoke
Amazon SNS Publish
Amazon ECS RunTask
AWS Step Functions StartExecution
AWS Glue StartJobRun
Start
Choice state ProcessTransaction
$.TransactionType == "REFUND"
$.TransactionType == "PURCHASE"
Lambda: Invoke ProcessRefund
Lambda: Invoke ProcessPurchase
End
Definition
The state that is the starting point of the workflow.
A human-readable description of the state machine.
TimeoutSeconds - optional

Fig: 4.5.2: Final Design

It demonstrates how workflow states are added and configured using Amazon States Language (ASL). The flow includes branching logic and error handling for robustness . Selecting execution role and creating state machine – shows the selection of the IAM role created earlier and the deployment of the final state machine. This enables the defined workflow to be executed securely and effectively.

Step 6: Output logs from ProcessPurchase.



Step Functions > State machines > TransactionProcessorSMRs > Execution: 1fca0498-43b3-42be-aef1-feb2a4945b05

Execution started successfully

Execution: 1fca0498-43b3-42be-aef1-feb2a4945b05

Execution status
Succeeded

Execution type
Standard

Execution ARN
arn:aws:states:ap-southeast-2:727646475418:execution:TransactionProcessorSMRs:1fca0498-43b3-42be-aef1-feb2a4945b05

IAM role ARN
arn:aws:iam::727646475418:role/SFrole

State transitions
4

Start time
Apr 5, 2025, 03:40:25.789 (UTC+05:30)

End time
Apr 5, 2025, 03:40:26.504 (UTC+05:30)

Duration
00:00:00.715

Alias
-

Version
-

Fig: 4.6.1: Output of executed Successfully

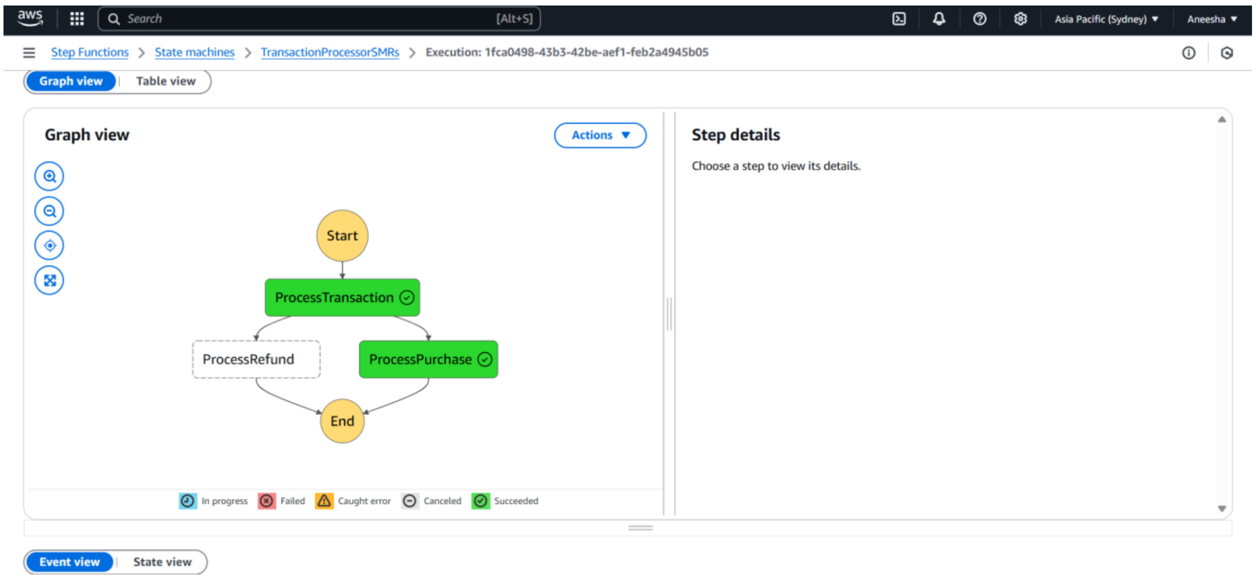
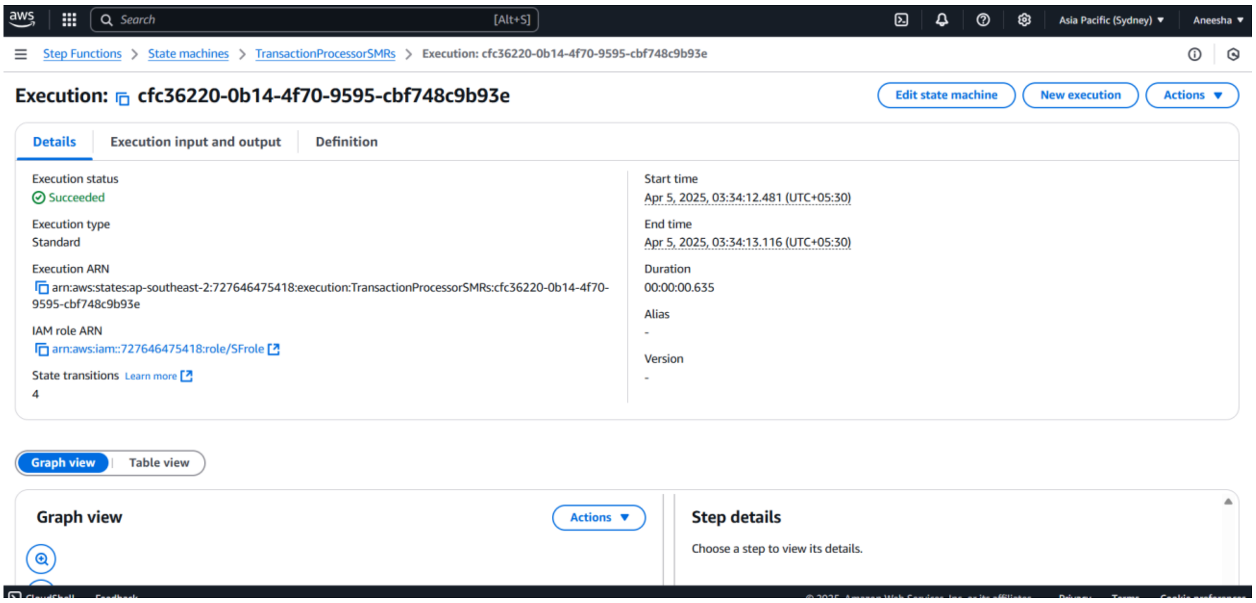


Fig: 4.6.2: Final Output

Output logs from ProcessPurchase – provides evidence of successful logic execution and data flow through CloudWatch logs. It ensures correctness of the workflow output and Lambda behavior .

Step 7: Output logs from ProcessRefund.


Step Functions > State machines > TransactionProcessorSMRs > Execution: cfc36220-0b14-4f70-9595-cbf748c9b93e

Execution: cfc36220-0b14-4f70-9595-cbf748c9b93e

Details | Execution input and output | Definition

Execution status
Succeeded

Execution type
Standard

Execution ARN
arn:aws:states:ap-southeast-2:727646475418:execution:TransactionProcessorSMRs:cfc36220-0b14-4f70-9595-cbf748c9b93e

IAM role ARN
arn:aws:iam::727646475418:role/SFrole

State transitions
4

Start time
Apr 5, 2025, 03:34:12.481 (UTC+05:30)

End time
Apr 5, 2025, 03:34:13.116 (UTC+05:30)

Duration
00:00:00.635

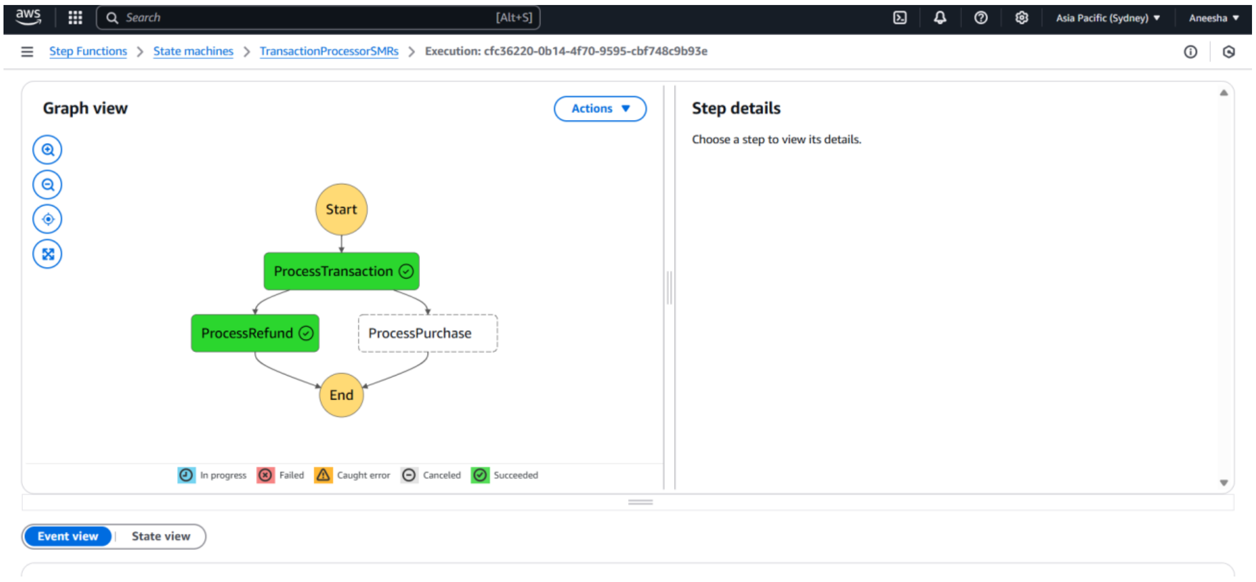
Alias
-

Version
-

Graph view

Step details
Choose a step to view its details.

Fig: 4.7.1: Output of executed Successfully


Step Functions > State machines > TransactionProcessorSMRs > Execution: cfc36220-0b14-4f70-9595-cbf748c9b93e

Step details
Choose a step to view its details.

Fig: 4.7.2: Final Output

Output logs from ProcessRefund – illustrates how execution logs are reviewed in CloudWatch for the refund logic. This helps validate output and debug any errors in logic.

5. Learning Outcomes

1. Understanding Serverless Architecture:
   Learned to build a serverless application using AWS Step Functions and Lambda. Understood the benefits such as cost efficiency and scalability in designing modern workflows.

2. Working with AWS Services:
   Gained practical experience with AWS Step Functions, Lambda, IAM, and CloudWatch. Understood how cloud services interact to build efficient applications.

3. IAM Role Configuration:
   Learned the importance of secure access management through IAM roles . Created and assigned policies to ensure proper role-based permissions for workflow execution.

4. Debugging and Problem-Solving:
   Used CloudWatch logs to identify and fix issues like execution errors. Applied debugging practices to refine workflow transitions and function outputs [6].

5. State Machine Design and Execution:

   Designed workflows using Amazon States Language (ASL) and monitored them during execution. Validated correct task sequencing and error handling during state transitions0.

6. Best Practices in Development:
   Followed cloud best practices including modular Lambda code, structured logs, and policy-based permissions. Learned the significance of maintaining stateless architecture for seamless cloud operations.

6. Conclusion

The Cloud-Based Task Management System demonstrates an efficient approach to automating task workflows using AWS Step Functions and related services [3]. It offers a highly scalable, maintainable, and cost-effective architecture, making it an ideal solution for businesses aiming to streamline their backend operations. The use of serverless technologies allows for modular updates, reduced infrastructure management, and rapid deployment cycles, significantly lowering the time to market. This system supports automated execution and error handling via AWS Step Functions, ensuring that each process follows a defined and reliable path. By leveraging Lambda functions to handle discrete tasks and DynamoDB for persistent data storage, the application achieves stateless, efficient processing.

Furthermore, the implementation of IAM roles ensures secure interaction between services, granting permissions with the principle of least privilege to enhance security posture. Monitoring tools like CloudWatch provide insight into execution metrics and logs, aiding in quick issue resolution and performance optimization. These features collectively contribute to operational stability and robustness of the system in a production-grade environment. These enhancements would increase interactivity and allow for more dynamic workflows. Overall, this project demonstrates the potential of serverless cloud-native architectures to transform traditional task management into intelligent, automated systems while maintaining agility, efficiency, and security.

7. References

[1] AWS Step Functions Documentation:
https://docs.aws.amazon.com/step-functions/latest/dg/welcome.html

[2] AWS Lambda Developer Guide:
https://docs.aws.amazon.com/lambda/latest/dg/welcome.html

[3] Amazon DynamoDB Guide:
https://docs.aws.amazon.com/dynamodb/latest/developerguide/Introduction.html

[4] AWS IAM Documentation:
https://docs.aws.amazon.com/IAM/latest/UserGuide/introduction.html

[5] Amazon CloudWatch Documentation:
https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/WhatIsCloudWatch.html

[6] Amazon API Gateway Guide:
https://docs.aws.amazon.com/apigateway/latest/developerguide/welcome.html

[7] AWS General Documentation:
https://docs.aws.amazon.com/