AWS Step Functions for Automated Order Processing in E-commerce

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

Thatiparti Taniswi

2210030402

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 **"AWS Step Functions for Automated Order Processing in E-commerce"** which is a experimental &/ Simulation work carried out by Thatiparti Taniswi (2210030402), 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

1. INTRODUCTION

In today's fast-paced digital environment, automation is crucial for improving operational efficiency, particularly in e-commerce. Manual processing of orders introduces delays and errors, leading to customer dissatisfaction. To solve this, we implement a serverless, event-driven architecture using AWS services to automate the order lifecycle.

This project uses AWS Step Functions to coordinate a series of tasks: order validation, payment processing, and storage. Each task is performed using AWS Lambda, providing scalability and fault tolerance. This approach simulates a real-world backend for online shopping platforms.

The main objectives of this project:

- Design a workflow using AWS Step Functions.
- Use Lambda functions to perform isolated tasks.
- Automate order validation, payment, and storage.
- Monitor and debug with CloudWatch.

2. AWS Services Used as part of the project

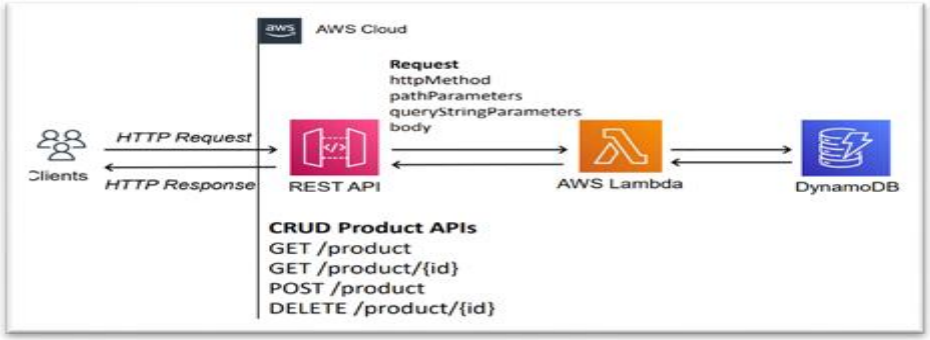
The project utilizes several AWS services to build a serverless architecture:

1. AWS Lambda:

- Executes business logic for each step.
- Processes input/output and updates task states
- Interacts with DynamoDB and other services
- 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 [3].
- 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 [4]. Ensures secure, role-based resource access. o Provides fine-grained access control for various services. o Logs API calls using AWS CloudTrail for auditing purposes.

3.Steps Involved in solving project problem statement

3.1 Introduction to SQS

To enable asynchronous communication between services and ensure scalable order handling, this project integrates Amazon Simple Queue Service (SQS) into the e-commerce order processing pipeline. SQS acts as a buffer between the order placement system and backend processors, improving reliability and performance.

3.2 Queue Type Selection

The implementation began by accessing the AWS Management Console and navigating to the SQS Dashboard. A Standard Queue was selected over a FIFO queue, as it offers higher throughput and is well-suited for scenarios where exact message order is less critical than performance and scalability. This type ensures at-least-once message delivery and supports multiple consumers.

3.3 Queue Configuration

A queue named PurchaseQueue was created with the following settings:

- Visibility Timeout: Configured to prevent multiple consumers from processing the same message simultaneously.
- Message Retention Period: Retained at the default value of 4 days, allowing sufficient time for message processing.
- Delivery Delay: Disabled for real-time message handling.
- Access Policy: Left at default but can be modified to restrict access to specific AWS services or IAM roles.

These configurations ensure that the system is fault-tolerant and capable of handling varying workloads.

3.4 Integrating the Queue into the System

Once the queue was created, the application's frontend was configured to push new order messages into PurchaseQueue. Each message followed a standardized JSON format, including fields like:

- orderId
- customerId
- itemList
- paymentStatus

This format simplifies message parsing and ensures seamless integration with downstream services like AWS Lambda and AWS Step Functions.

3.5 Benefits of Using SQS

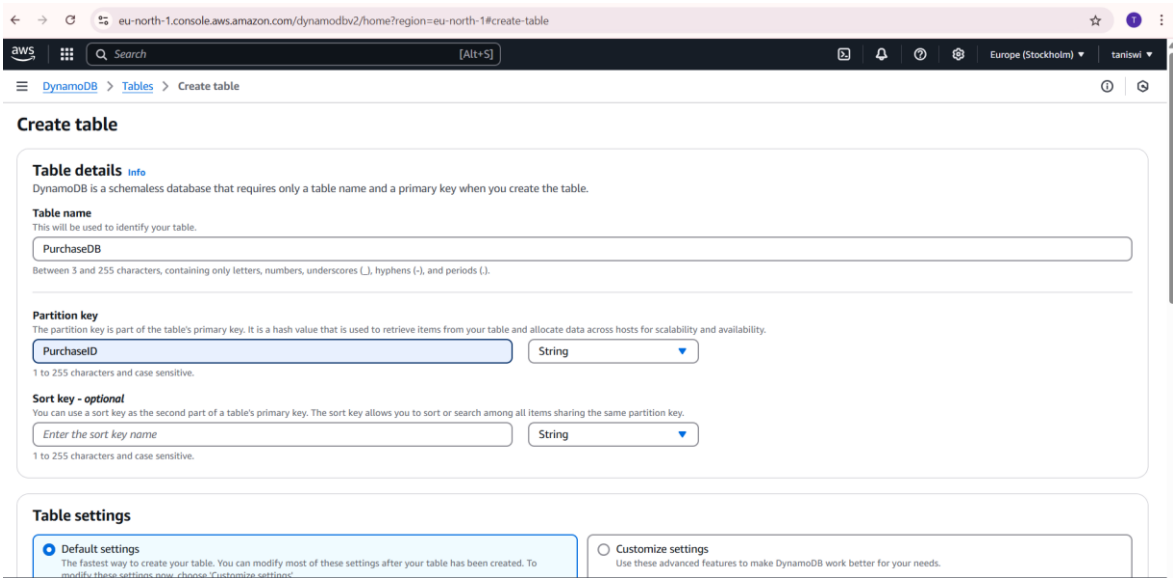
Integrating Amazon SQS into the order processing system offers the following advantages:

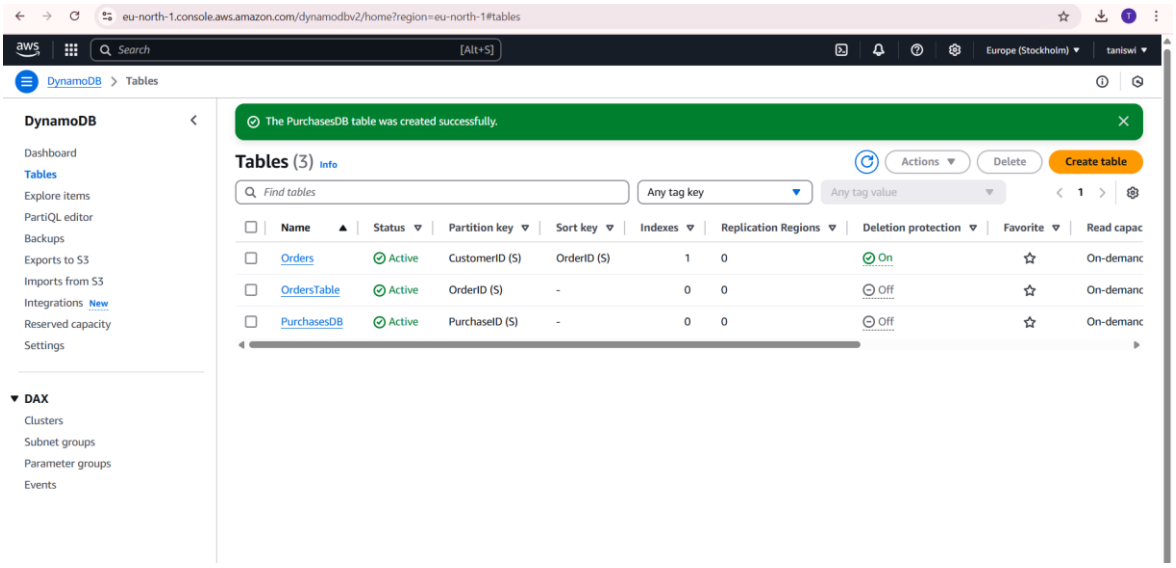
- Decoupled Architecture: Frontend and backend services operate independently.
- Improved Scalability: Allows multiple Lambda functions or microservices to consume messages in parallel.
- High Availability: Built-in redundancy and fault-tolerance provided by AWS.
- Event-Driven Workflow: Enables smooth orchestration of order processing tasks via AWS Step Functions.

4. Stepwise Screenshots with brief description

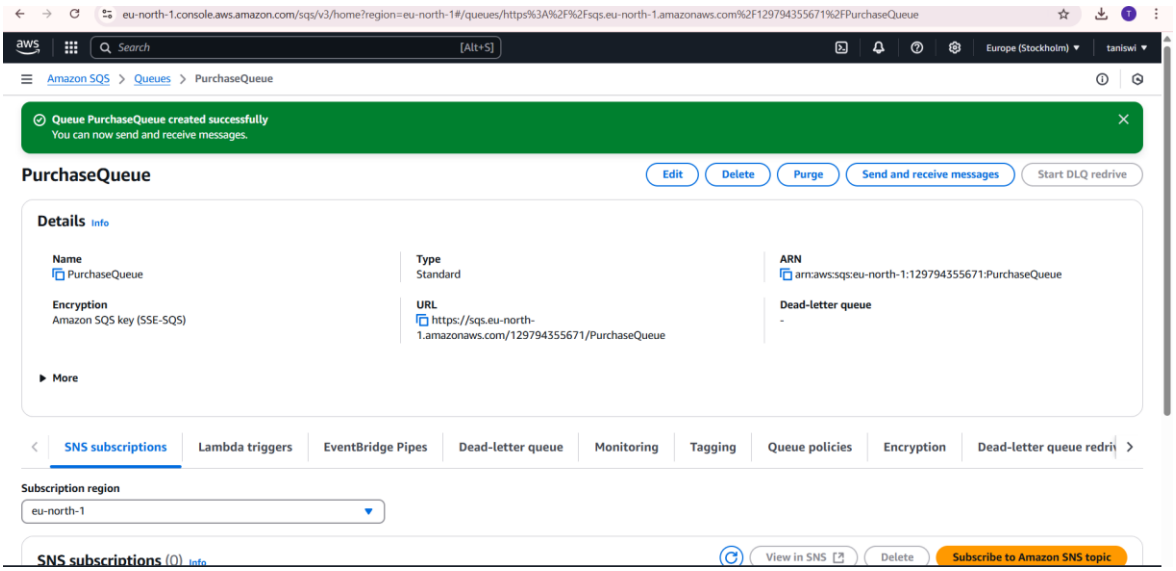
Creating and Connecting DynamoDB Table

- Go to DynamoDB and create a table with PurchaseID as the partition key.
- This table tracks each order's state and metadata throughout the workflow.

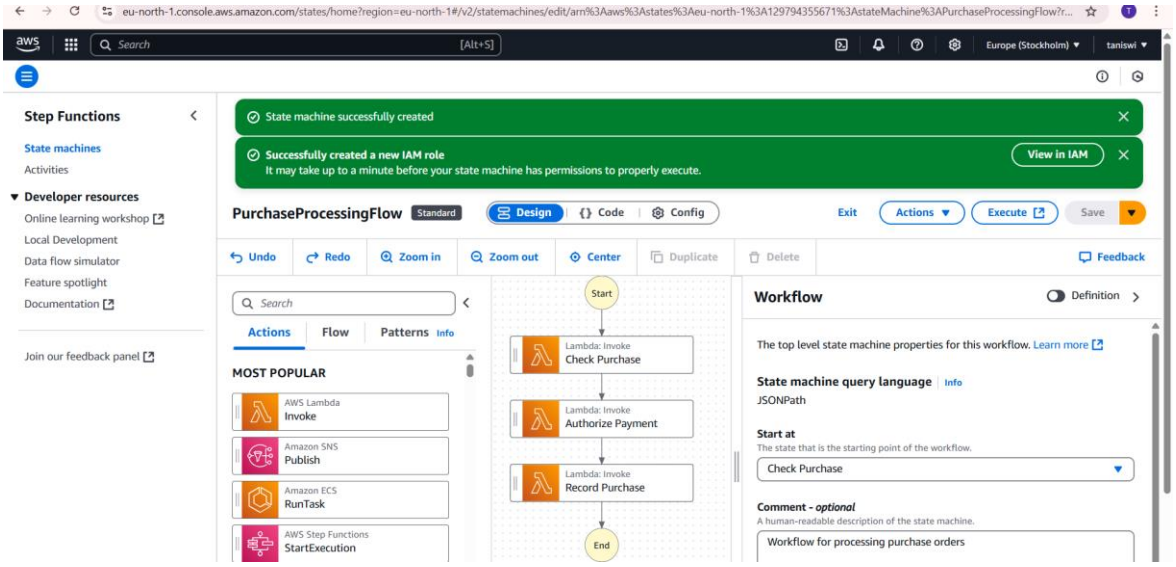




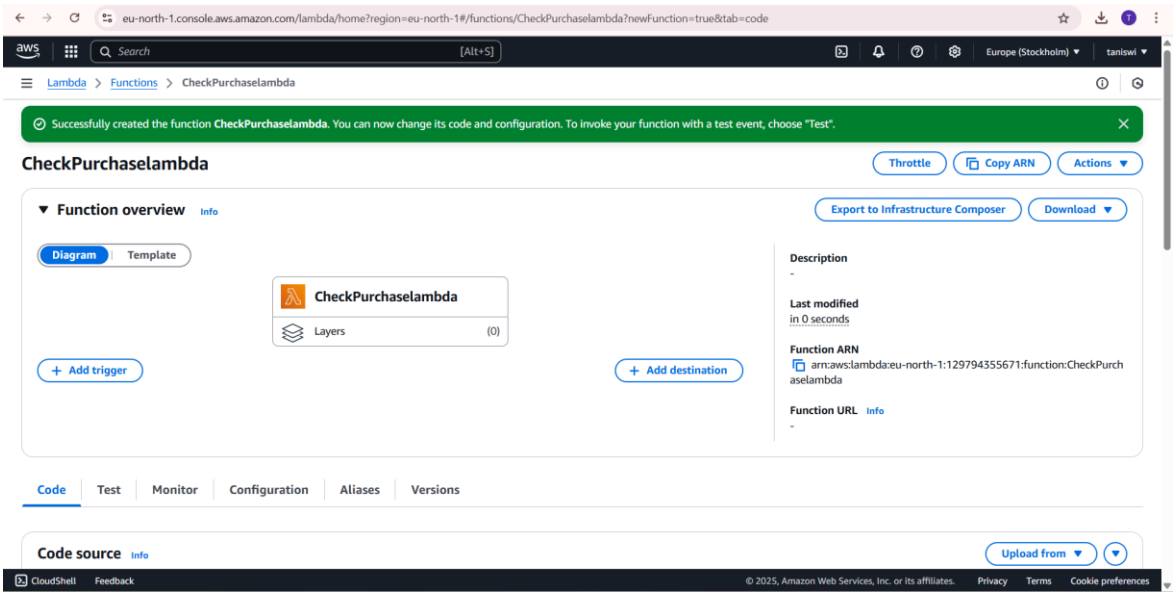
Creating the Purchase Queue with Amazon SQS



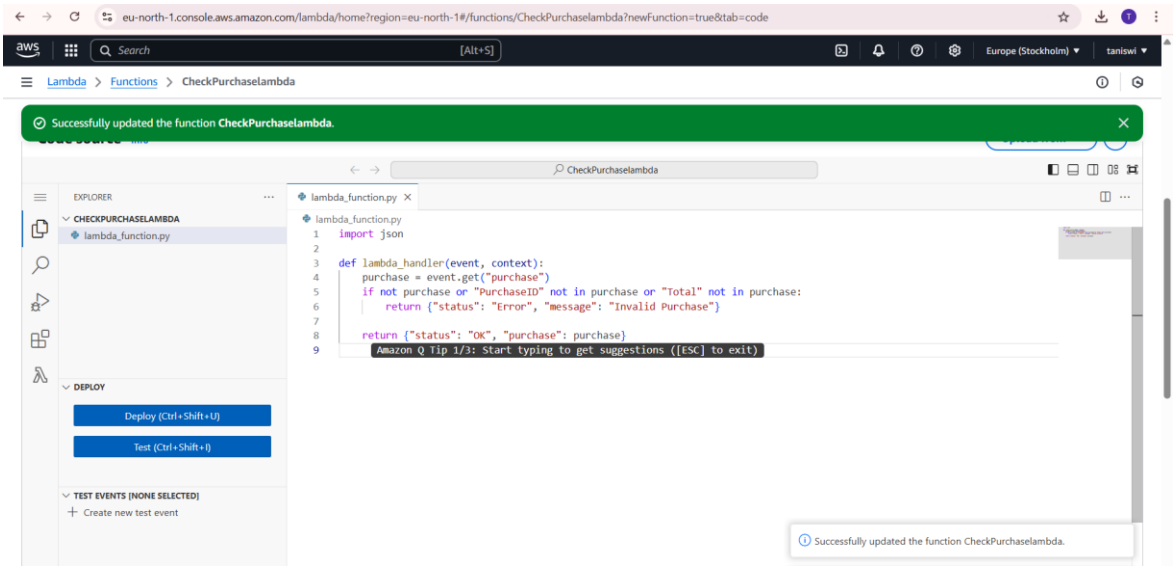
Visualizing the State Machine in AWS Step Functions



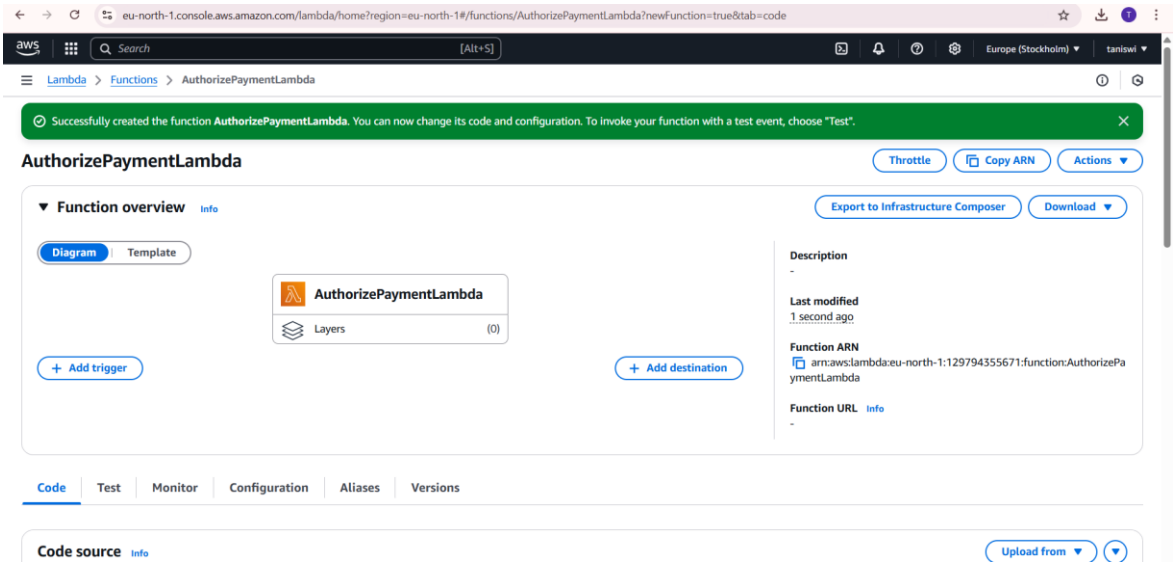
Creating Lambda Function for Order Validation

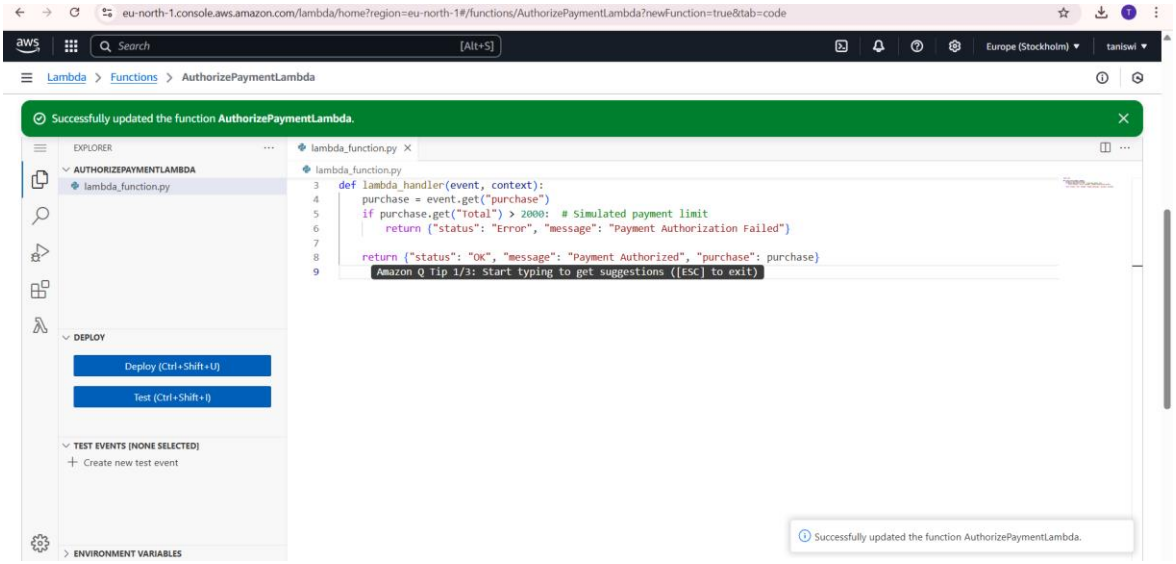


Creating CheckPurchaseLambda

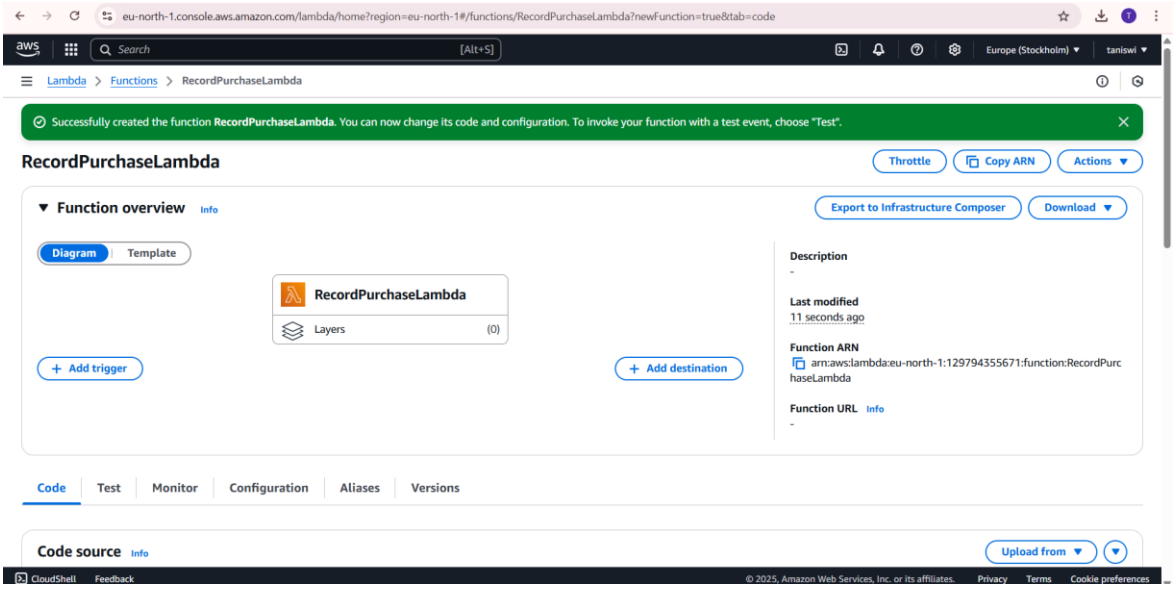


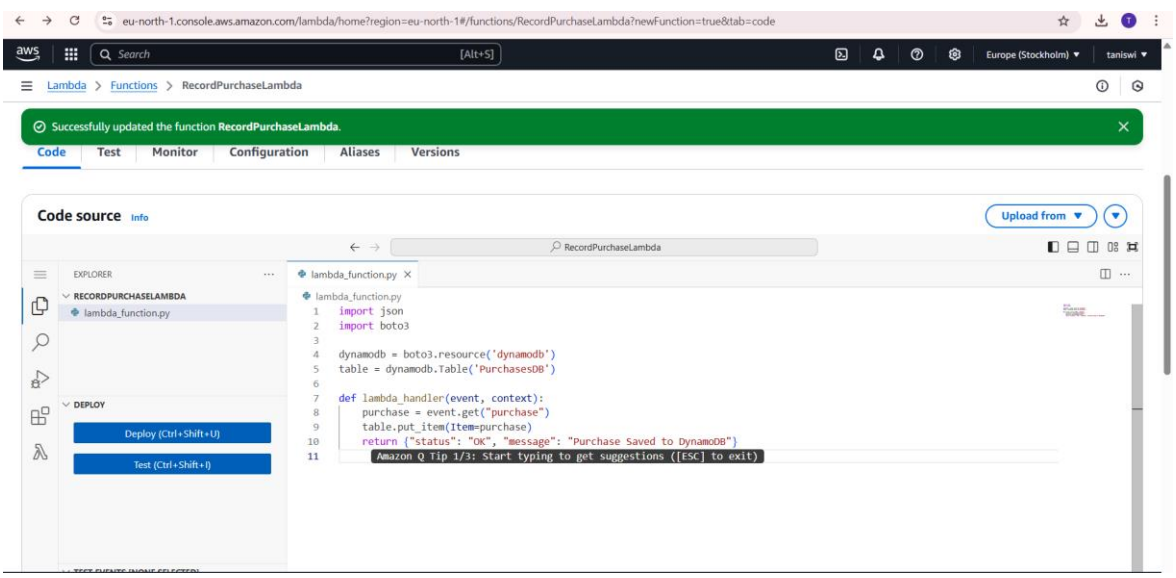
Creating AuthorizePaymentLambda



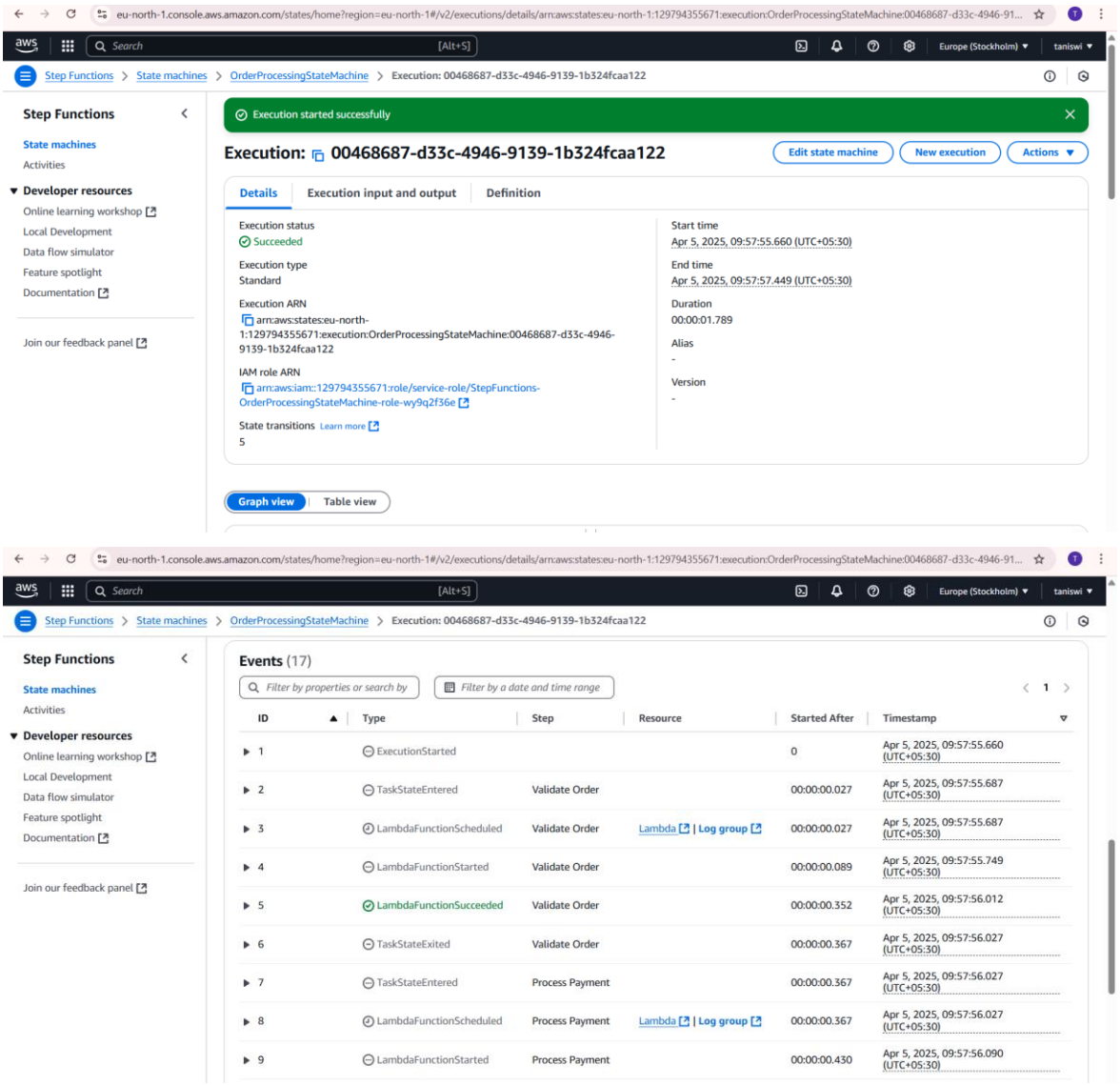


Creating RecordPurchaseLambda





Execution completed successfully



5.LEARNING OUTCOMES

Upon successful completion of this project, the following learning outcomes were achieved:

1. Gained hands-on experience in designing and implementing cloud-native workflows using AWS Step Functions for real-time orchestration.
2. Learned how to decompose a monolithic application into microservices using AWS Lambda to handle independent tasks such as payment validation, inventory updates, and shipping label generation.
3. Understood the use of Amazon SQS as a message queue to enable asynchronous communication between different services and decouple components in a serverless architecture.
4. Explored the integration of Amazon DynamoDB for managing and tracking the state of orders, user information, and inventory in a scalable, low-latency NoSQL database.
5. Implemented real-time notification systems using Amazon SNS to alert customers and administrators on order status updates and failures.
6. Developed a fault-tolerant, scalable, and event-driven system that follows best practices in serverless computing.
7. Improved problem-solving skills by handling exceptions, retries, and branching logic within the Step Function state machine.
8. Gained confidence in using IAM roles and policies to securely connect AWS services with precise access control.

6.CONCLUSION

This project successfully demonstrated the design and implementation of an automated order processing system using AWS Step Functions and a suite of serverless AWS services. By integrating Lambda functions, SQS, DynamoDB, S3, and SNS within a centralized state machine, we created a robust and scalable solution that can handle real-time e-commerce transactions efficiently. Each component was designed to perform a specific task, and the orchestration using Step Functions ensured smooth coordination between them with proper error handling and retries.

The system not only reduces manual intervention and processing time but also provides flexibility and resilience in handling varying workloads. It highlights the power of serverless architecture in modern cloud applications, enabling rapid development, cost efficiency, and easier maintenance. This project serves as a strong foundation for building more complex and production-ready e-commerce workflows using cloud-native tools.

7.REFERENCES:

[1] Amazon Web Services (AWS). (2024). AWS Step Functions – Overview. Available at https://aws.amazon.com/step-functions

[2] AWS Documentation. (2024). Automating Workflows with Step Functions. Available at https://docs.aws.amazon.com/step-functions

[3] Amazon Web Services (AWS). (2024). Step Functions and Order Processing. Available at https://aws.amazon.com/blogs/architecture/

[4] Cloud Performance Journal. (2023). Workflow Optimization in E-commerce. Available at https://cloudperformancejournal.com

[5] U.S. Department of Homeland Security. (2023). API Security Best Practices. Available at https://www.dhs.gov/api-security

[6] Amazon Web Services (AWS). (2024). Step Functions Scalability. Available at https://aws.amazon.com/blogs/architecture/

[7] AWS Developer Guide. (2024). Monitoring Step Functions with CloudWatch. Available at https://docs.aws.amazon.com/step-functions/latest/dg/

[8] Amazon Tech Blog. (2023). Scaling Order Processing with AWS Step Functions. Available at https://aws.amazon.com/blogs/

[9] Shopify Engineering. (2023). Automating Payments with Step Functions. Available at https://shopify.engineering

[10] Walmart Tech. (2023). Inventory Automation with AWS.
Available at https://walmarttech.