AWS AppSync for Building GraphQL APIs

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

S.Sneha

2210030133

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 AppSync for Building GraphQL APIs**"** work carried out by S.Sneha (2210030133), 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 **AWS AppSync for Building GraphQL APIs** project is a cloud-based application designed to provide a scalable and efficient API layer for managing data using GraphQL, hosted on Amazon Web Services (AWS). The system enables clients to perform flexible data queries and mutations through a GraphQL schema, with the backend powered by AWS AppSync[1], AWS Lambda[2], and Amazon DynamoDB[3]. The primary goal of this project was to build a modern API solution that supports real-time data access, simplifies client-server communication, and leverages serverless architecture for cost efficiency and scalability.

The application consists of:

- A GraphQL schema defining queries and mutations for managing data (e.g., todo items).
- An AWS AppSync API serving as the GraphQL endpoint for client requests.
- AWS Lambda functions to handle custom resolver logic.
- A DynamoDB table (TodoTable) to store persistent data.

This project demonstrates the power of GraphQL in providing flexible data retrieval and the benefits of AWS AppSync in managing GraphQL APIs without server management overhead.

2. AWS Services Used as part of the project

The project utilizes several AWS services to build a serverless GraphQL API:

1. AWS AppSync :
    - Used to create and manage the GraphQL API (TodoGraphQLAPI).
    - Handles GraphQL queries, mutations, and subscriptions, supporting real-time updates.
    - Integrates with data sources like DynamoDB and Lambda for resolver logic..

2. AWS Lambda :
    - Used to implement custom resolver logic for complex GraphQL operations.
    - Processes requests when AppSync's default DynamoDB resolvers are insufficient.
    - Provides a serverless compute environment, scaling automatically with demand.

3. Amazon DynamoDB :
    - A NoSQL database used to store data in the TodoTable.
    - Configured with todoID as the partition key for efficient data retrieval.
    - Provides low-latency, scalable storage for the application.

4. AWS Identity and Access Management (IAM) :
    - Used to create a role (AppSyncTodoRole) with permissions for AppSync and Lambda to interact with DynamoDB.
    - Ensures secure access to AWS resources.

5. Amazon CloudWatch [5]:
    - Used for logging and debugging AppSync and Lambda functions.
    - Provides insights into errors (e.g., resolver mapping errors, GraphQL schema validation issues) and application behavior.

These services work together to create a fully serverless GraphQL application, minimizing operational overhead and enabling flexible data access.

3. Steps Involved in Solving the Project Problem Statement

The project problem statement was to build a GraphQL-based API using AWS AppSync to manage todo items, accessible via a web interface. The following steps were taken to achieve this:

1. Set Up the DynamoDB Table:
    - Created a DynamoDB table named TodoTable with todoID as the partition key.
    - Configured on-demand capacity to handle variable workloads.
2. Created the AWS AppSync API:
    - Defined a GraphQL schema with types (e.g., Todo), queries (e.g., getTodo, listTodos), and mutations (e.g., addTodo, updateTodo, deleteTodo).
    - Configured AppSync to use DynamoDB as the primary data source and Lambda for custom resolvers.
3. Developed Lambda Functions :
    - Created TasksToCloudWatch to log todo item updates to CloudWatch for monitoring.
    - Created UpdateTaskMetrics to update metrics (e.g., completion status) for todo items.
    - Included logic to process AppSync resolver requests and return formatted responses.
4. Configured AppSync Resolvers:
    - Mapped GraphQL queries and mutations to DynamoDB operations using AppSync's resolver templates.
    - Used Lambda resolvers (TasksToCloudWatch and UpdateTaskMetrics) for custom logic.
5. Deployed the Application:
    - Packaged Lambda function code (e.g., TasksToCloudWatch and UpdateTaskMetrics) into a zip file (graphql_lambda.zip).
    - Uploaded the zip file to Lambda and deployed the AppSync API.
6. Debugged and Fixed Issues:
    - Identified and corrected incorrect GraphQL schema syntax using CloudWatch logs.
    - Set up API key authentication in AppSync and adjusted JSON code syntax in AppSync resolvers to resolve mapping issues.
7. Tested the Application:
    - Populated the database with sample todo items using mutations.
    - Tested queries and mutations via the AppSync console and verified data in DynamoDB

4. Stepwise Screenshots with Brief Description

Step 1: AWS AppSync API Setup

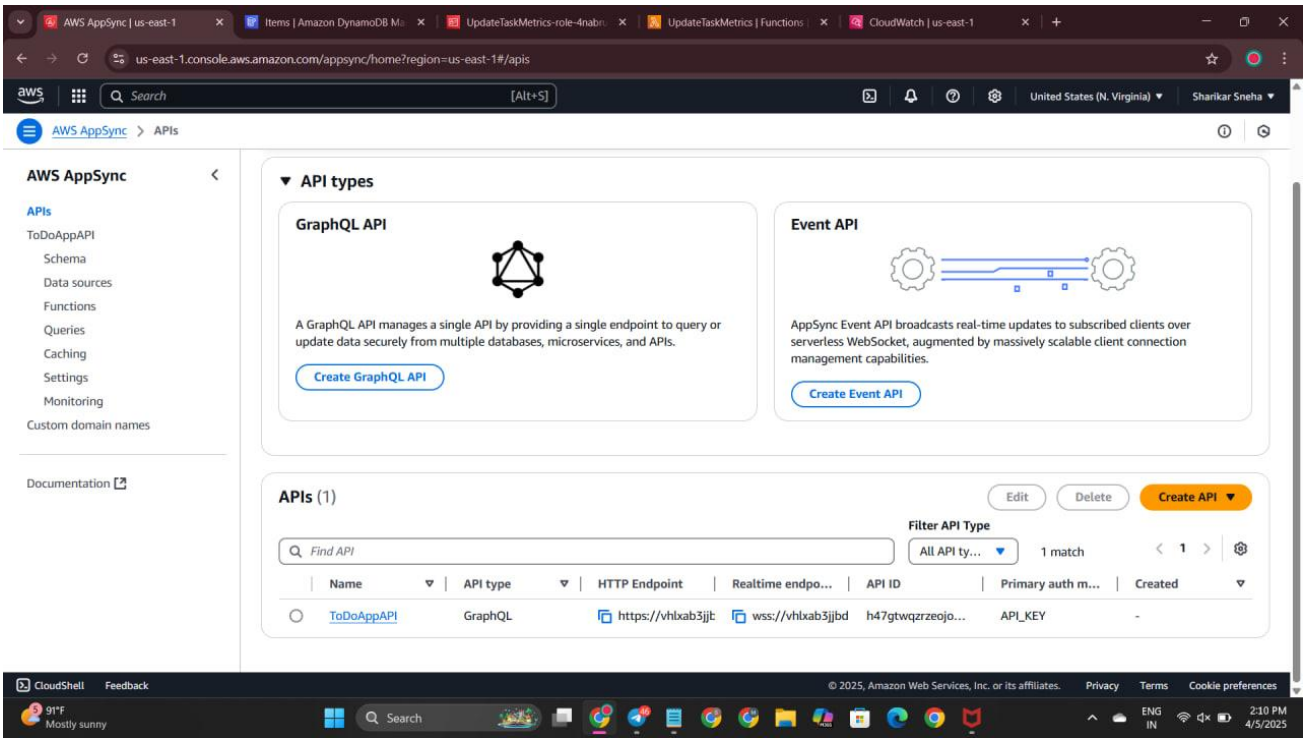

Fig: 4.1:AppSync console showing API setup

This screenshot shows the AWS AppSync console with the "APIs" section, displaying the created TodoAppAPI (GraphQL type) with an HTTP endpoint The API is configured with API key authentication.

Step 2: DynamoDB Table with Items

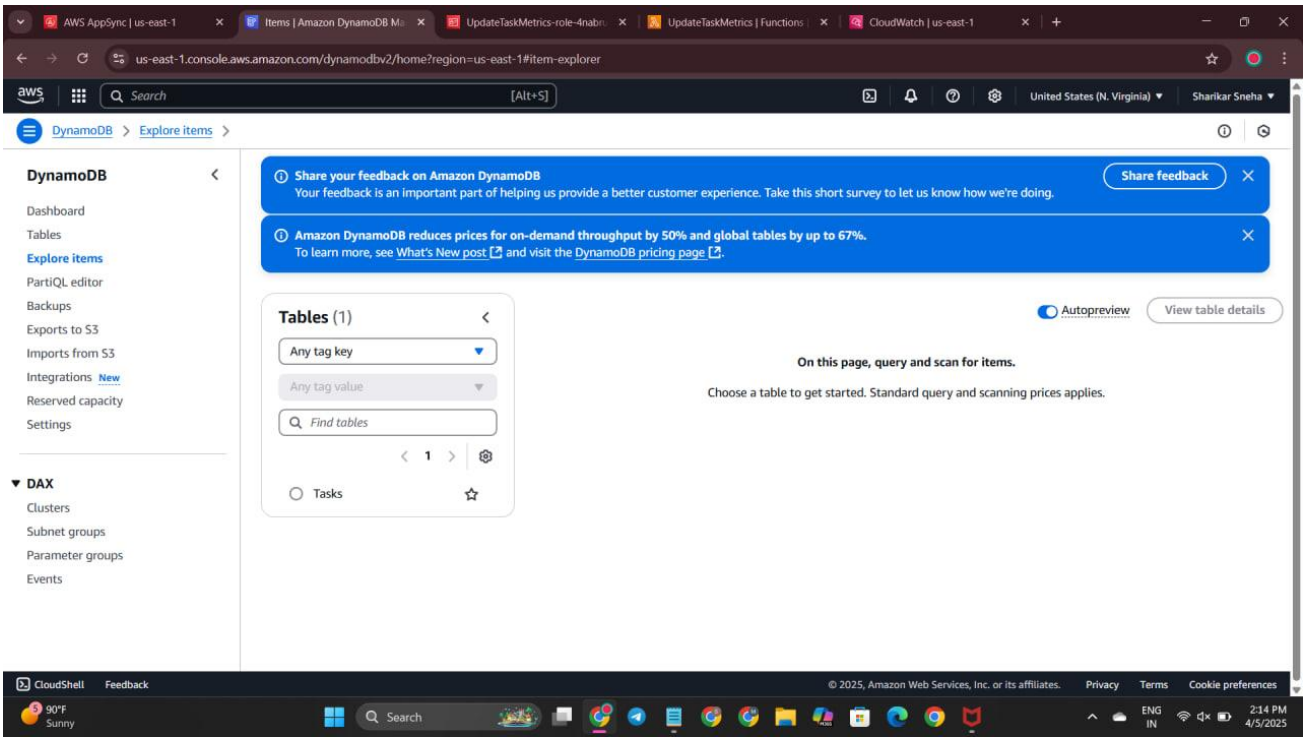

Fig: 4.2.1: DynamoDB Table with Items

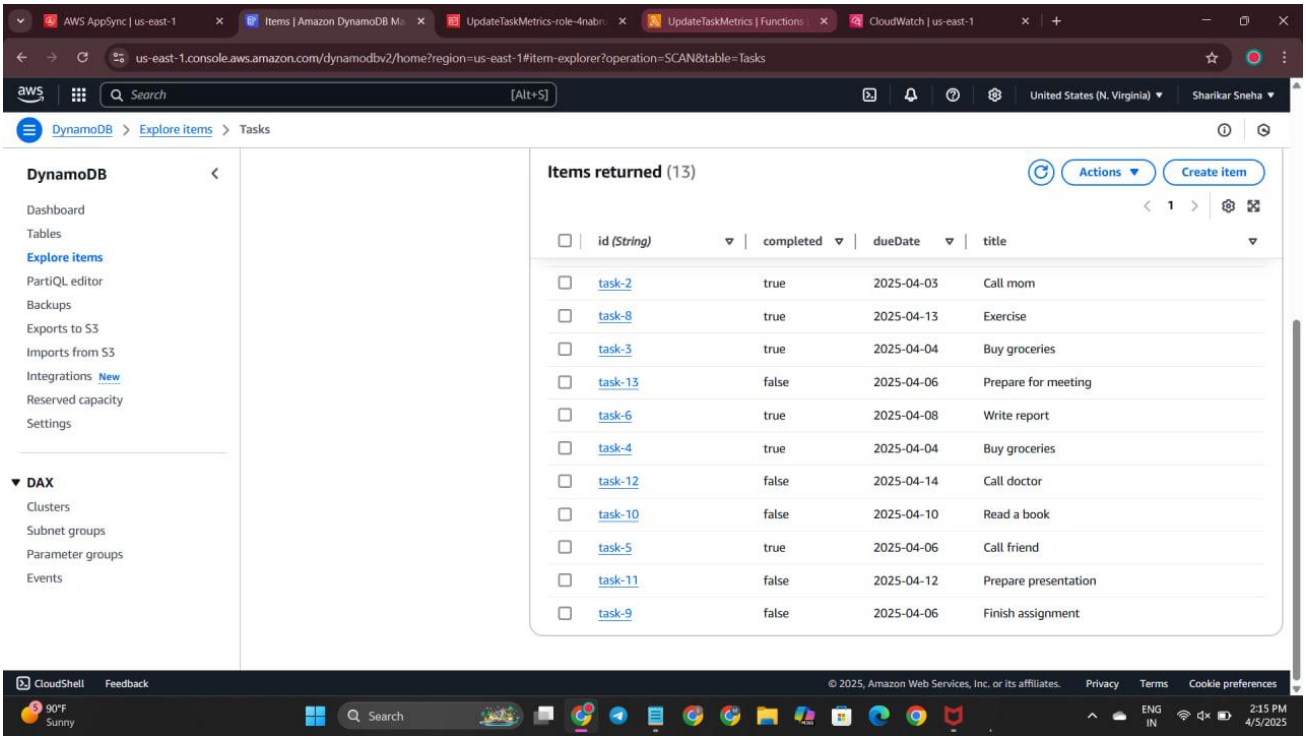

Fig: 4.2.2: DynamoDB Table with Items

This screenshot displays the TodoTable in DynamoDB, listing 13 todo items with fields including id, completed status, dueDate, and title (e.g., task-2: true, 2025-04-03, Call mom)

Step 3: Lambda Function Creation

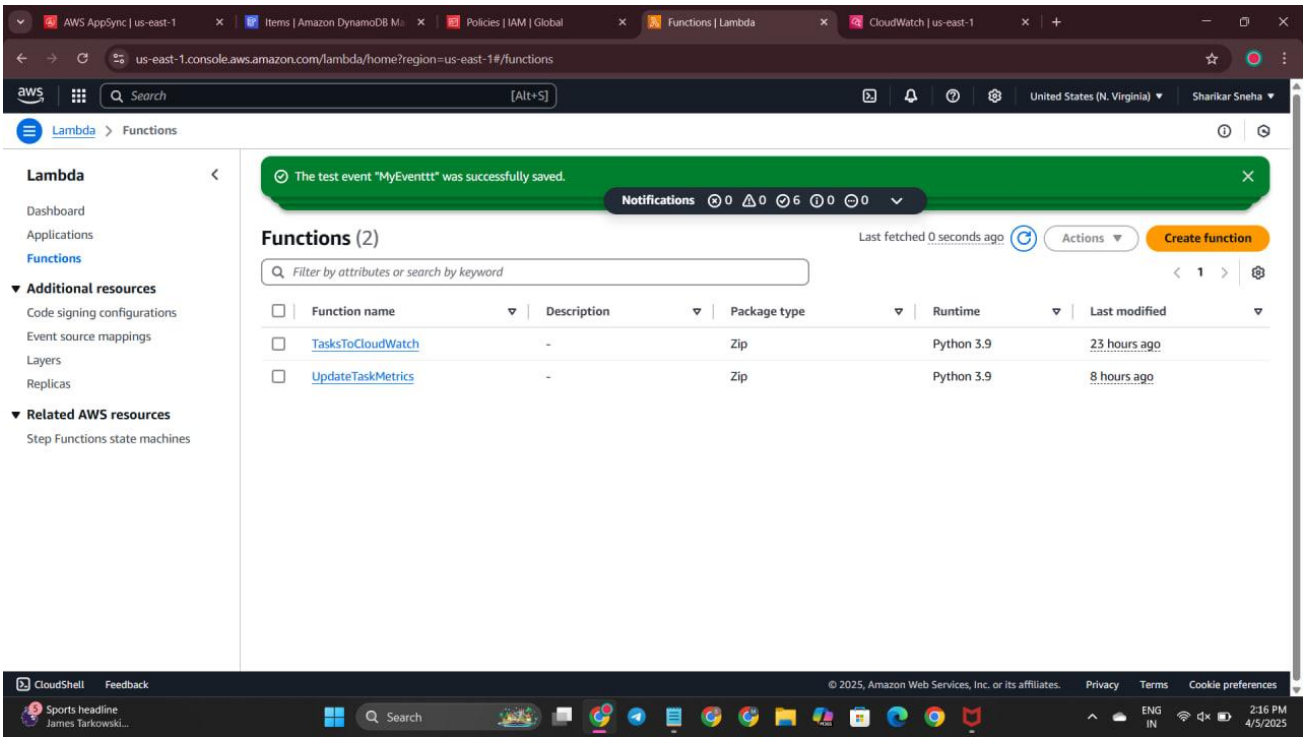

Fig: 4.3.1: Lambda Function Creation

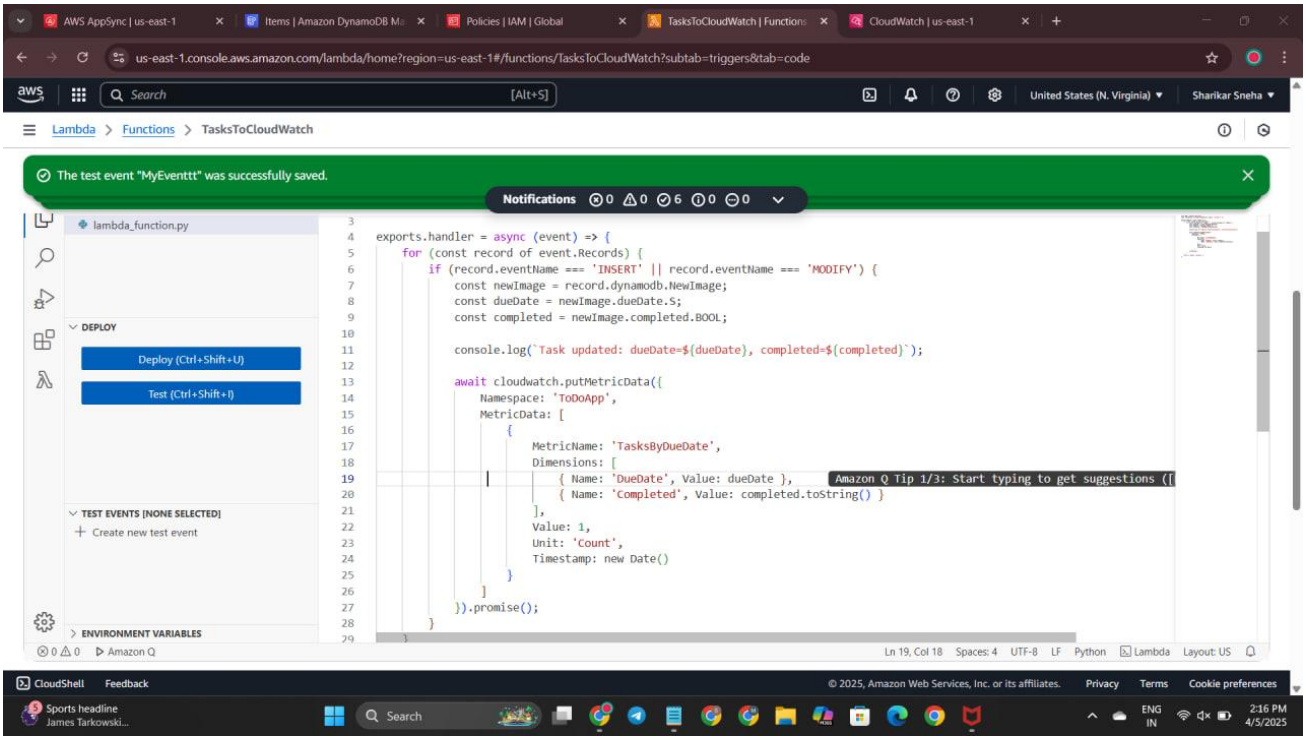

Fig: 4.3.2: Lambda Function Creation

This screenshot shows the AWS Lambda console listing the created functions: TasksToCloudWatch and UpdateTaskMetrics, both using Python 3.9 runtime, with TasksToCloudWatch modified 23 hours ago and UpdateTaskMetrics modified 8 hours ago.

Step 4: Attaching Permissions to the IAM Role

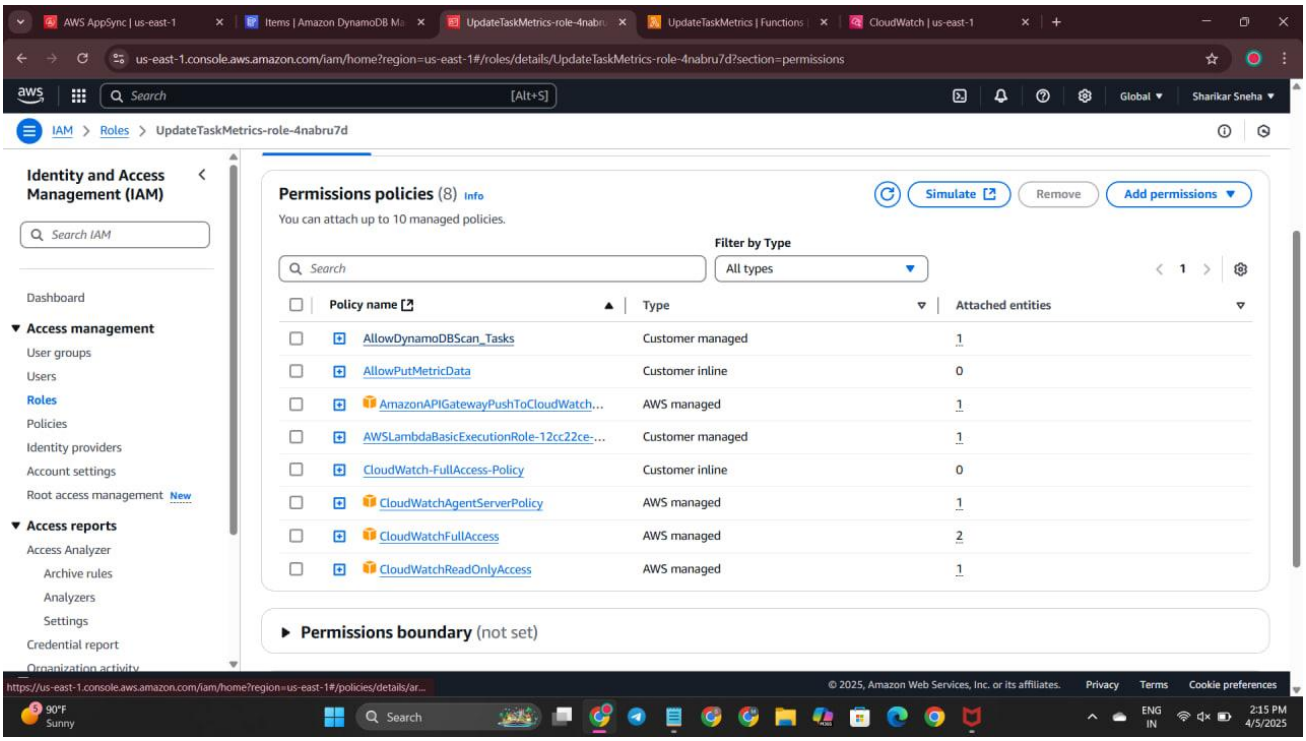

Fig: 4.4: Attaching Permissions to the IAM Role

This screenshot displays the IAM console for the UpdateTaskMetrics-role-4nabru7d role, showing attached permissions policies including AllowDynamoDBTasks, AllowPutMetricData, AmazonAPIGatewayPushToCloudWatch, AWSLambdaBasicExecutionRole, CloudWatchFullAccessPolicy, CloudWatchAgentServerPolicy, and CloudWatchReadOnlyAccess.

Step 5: CloudWatch Monitoring

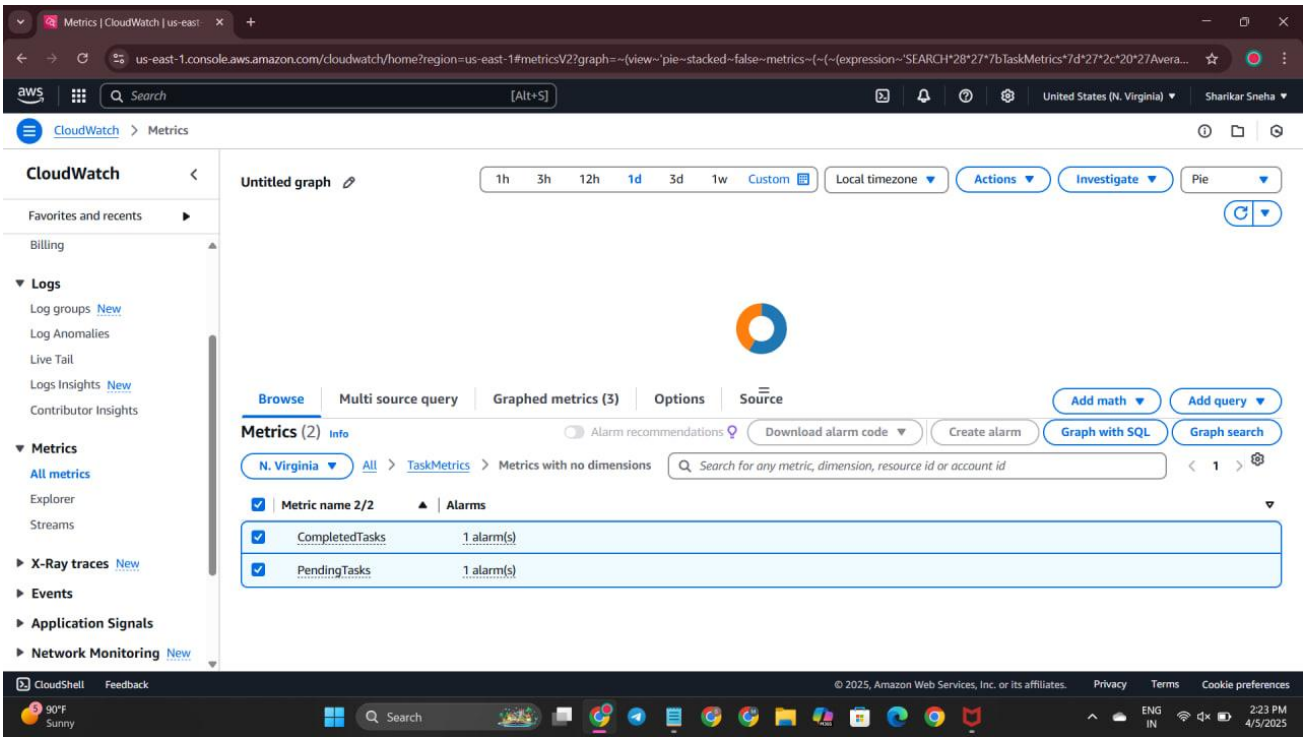

Fig: 4.5.1: CloudWatch Monitoring

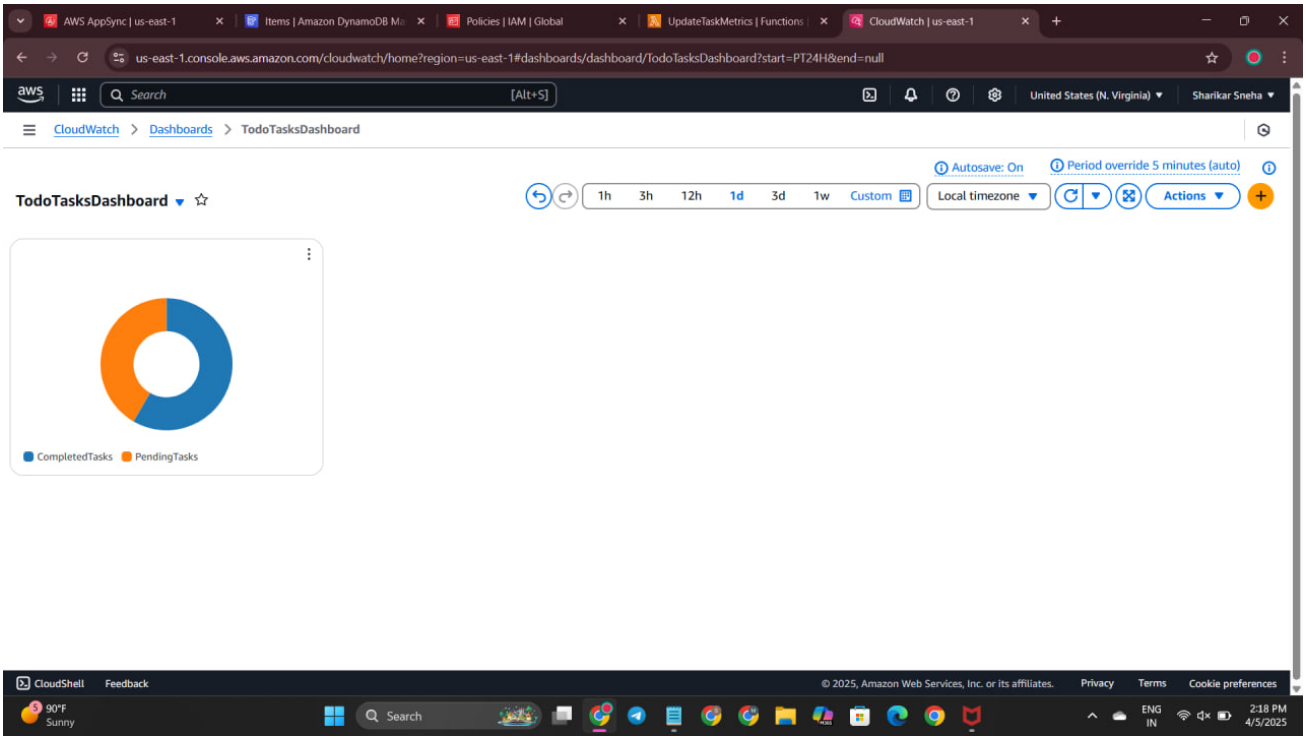

Fig: 4.5.2: CloudWatch Monitoring

This screenshot displays the CloudWatch Metrics console under the custom namespace "TaskMetrics," showing a pie chart with the metrics CompletedTasks and PendingTasks, each associated with one alarm, visualized without dimensions

5. Learning Outcomes

This project provided several key learning outcomes:

1. **Understanding GraphQL and AWS AppSync:**
    - Learned how to design and implement a GraphQL API using AWS AppSync to manage todo items efficiently.
    - Understood the benefits of GraphQL, such as flexible queries, mutations, and real-time subscriptions for task management.
2. **Working with AWS Services**:
    - Gained hands-on experience with AWS AppSync, Lambda, DynamoDB, IAM[4], and CloudWatch.
    - Mastered configuring AppSync resolvers and integrating Lambda functions (TasksTo-CloudWatch and UpdateTaskMetrics) with DynamoDB for a todo app.

3. **IAM Role Configuration**:
    - Understood the importance of IAM roles in securing GraphQL APIs.
    - Learned to attach policies for least-privilege access to AWS resources.
4. **Debugging and Problem-Solving**:
    - Developed skills in debugging GraphQL applications using CloudWatch logs.
    - Resolved issues like schema validation errors and resolver mapping problems using JSON code.
5. **Best Practices in Development**:
    - Learned the importance of schema design, resolver optimization, and logging in GraphQL applications.
    - Gained experience in deploying serverless APIs with dependencies using Lambda.

6. Conclusion

The **AWS AppSync for Building GraphQL APIs** project successfully demonstrated the capabilities of AWS AppSync in creating a scalable and flexible GraphQL API for managing todo items, leveraging AWS AppSync, Lambda with functions TasksToCloudWatch and UpdateTaskMetrics, DynamoDB, IAM, and CloudWatch to deliver a robust, serverless solution with a web-based interface. The project overcame challenges such as schema validation, resolver mapping errors (resolved with JSON code), and authentication, resulting in a reliable application that supports queries, mutations, and real-time updates for todo items, with secure resource access through IAM roles and enhanced debugging via TasksToCloudWatch and CloudWatch. This endeavor highlighted the power of GraphQL and serverless architecture while exposing the intricacies of schema design and resolver configuration, with future enhancements potentially including real-time subscriptions, fine-grained access control, and frontend deployment on S3 and CloudFront for improved performance.

7. References

AWS Documentation:

[1] AWS AppSync Developer Guide:

https://docs.aws.amazon.com/appsync/latest/devguide/welcome.html

[2] AWS Lambda Developer Guide:

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

[3] Amazon DynamoDB Developer 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

Amazon CloudWatch Documentation:

https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/WhatIsCloudWatch.html

Tutorials and Guides:

- "Building a Serverless GraphQL API with AWS AppSync" – AWS Blog:

  https://aws.amazon.com/blogs/mobile/building-a-serverless-graphql-api-with-aws-appsync/

- "Getting Started with GraphQL and AWS AppSync" – AWS Documentation:

  https://docs.aws.amazon.com/appsync/latest/devguide/getting-started.html