Serverless Real-Time Stock Portfolio Tracker Using AWS

Course Name: Software Development & Construction

Instructor Name: Dr. Zunurain Hussain

Institution Name: Information Technology University

Group Members :

Names	Roll No.
Muhammad Daniyal Hammad	BSSE23406
Ch. Mohsin Raza Aujla	BSSE23040
Eashal Yaseen	BSSE23026

Table of Contents

List Of Table:

List Of Equations:

List Of Figures:

1 Project Proposal, Executive Summary & Problem Definition:

1.1 Executive Summary:

The rapid growth of digital trading has increased the need for accurate, real-time stock portfolio management systems that are secure, scalable, and cost-efficient. This project proposes the development of a Serverless Real-Time Stock Portfolio Tracker using Amazon Web Services (AWS). The system allows users to create and manage their investment portfolio, track live stock prices, and monitor profit and loss in real time through a web-based dashboard. The proposed solution is built on a fully serverless architecture, utilizing AWS services such as Amazon S3 for frontend hosting, API Gateway for secure communication, AWS Lambda for backend processing, DynamoDB for database storage, and CloudWatch for monitoring and logging. The system eliminates the need for traditional servers, ensuring automatic scalability, high availability, and low operational cost. The expected outcome is a reliable, secure, and user-friendly cloud-based stock tracking platform suitable for small investors and students.

1.2 Introduction:

With the increasing popularity of online stock trading, individuals are actively investing in financial markets more than ever before. However, many small investors and students still rely on multiple applications, manual calculations, and scattered platforms to track their stock investments. These traditional methods are inefficient, time-consuming, and lack real-time accuracy. Cloud computing has transformed the way modern applications are developed and deployed. Serverless computing, in particular, allows developers to focus on application logic without worrying about server management. AWS provides powerful, scalable, and secure cloud services that are ideal for developing real-time financial applications. This project aims to bridge the gap between manual stock tracking and modern cloud based automation by introducing a Serverless Real-Time Stock Portfolio Tracker that provides live updates, secure data storage, and effortless scalability using AWS,

1.3 Problem Statement:

Many investors face difficulties in tracking their stock investments efficiently due to:

- Lack of a centralized portfolio management system
- Manual calculation of profit and loss
- Delayed stock price updates
- Data security concerns
- Limited scalability of traditional systems

Most existing solutions either require paid subscriptions or lack proper automation, security, and real-time accuracy. As a result, users are unable to make quick and informed financial decisions.

Therefore, there is a strong need for a secure, automated, real-time, and cloud-based stock portfolio tracking system that is cost-efficient, scalable, and easily accessible. This project addresses these challenges by using a fully serverless AWS architecture.

Problem	AWS-Based Solution
Manual portfolio tracking	Automated serverless system
Delayed price updates	Real-time API fetching
Poor scalability	AWS auto-scaling
Security issues	IAM + API Gateway
High infrastructure cost	Pay-per-use serverless

Table 1 Problem vs Solution

1.4 Aim & Objectives:

1.4.1 Aim:

To design and implement a Serverless Real-Time Stock Portfolio Tracking System using AWS that enables users to securely manage their stock investments and view real time market performance.

1.4.2 Objectives:

1. To develop a cloud-based web application for managing stock portfolios.
2. To fetch and display real-time stock prices using external stock APIs.
3. To store portfolio data securely using AWS DynamoDB.
4. To implement backend logic using AWS Lambda.
5. To establish secure communication using Amazon API Gateway.
6. To host the frontend using Amazon S3.
7. To monitor system performance and logs using Amazon CloudWatch.
8. To ensure system security using proper IAM roles, permissions, and access policies.
9. To provide a scalable, cost-efficient, and highly available serverless solution.

2 Equations & Formulae

2.1 Average Buy Price (Cost Basis)

2.1.1 Purpose:

Calculates the average price at which a stock was purchased.

2.1.2 Formula:

$$\text{Average Buy Price} = \frac{\text{Total Amount Invested}}{\text{Total Quantity of Shares}}$$

Equation 1 Average Buy Price

2.1.3 Where:

$$\text{Total Amount Invested} = \sum(\text{Buy Price} \times \text{Quantity})$$

Equation 2 Total Amount Invested

2.2 Market Value of Holdings

2.2.1 Purpose:

Calculates the current worth of the stock holdings.

2.2.2 Formula:

$$\text{Market Value} = \text{Current Market Price} \times \text{Total Quantity of Shares}$$

Equation 3 Market Value

2.3 Total Investment (Cost)

2.3.1 Purpose:

Represents the total money spent on purchasing shares.

2.3.2 Formula:

$$\text{Total Investment} = \text{Average Buy Price} \times \text{Total Quantity}$$

Equation 4 Total Investment

2.4 Unrealized Profit / Loss (P&L)

2.4.1 Purpose:

Calculates profit or loss for holdings that are not yet sold.

2.4.2 Formula:

$$\text{Unrealized P\&L} = \text{Market Value} - \text{Total Investment}$$

Equation 5 Unrealized P&L

2.5 Profit / Loss Percentage

2.5.1 Purpose:

Shows gain or loss relative to the invested amount.

2.5.2 Formula:

$$\text{P\&L Percentage} = \left(\frac{\text{Unrealized P\&L}}{\text{Total Investment}}\right) \times 100$$

Equation 6 P&L Percentage

2.6 Realized Profit / Loss

2.6.1 Purpose:

Calculated when shares are sold.

2.6.2 Formula:

$$\text{Realized P\&L} = (\text{Sell Price} - \text{Average Buy Price}) \times \text{Quantity Sold}$$

Equation 7 Realized P&L

Equation No	Name	Formula	Purpose
1	Avg Buy Price	Total Cost / Quantity	Cost basis
2	Market Value	Price × Quantity	Current worth
3	Total Investment	Σ(Buy Price × Qty)	Total cost
4	Unrealized P/L	(CP − BP) × Qty	Open profit
5	P/L %	(P/L ÷ Investment) × 100	Performance

Table 2 Formula Summary

3 AWS Architecture Design, Implementation Steps &amp Screenshots;

3.1 AWS Architecture Diagram:

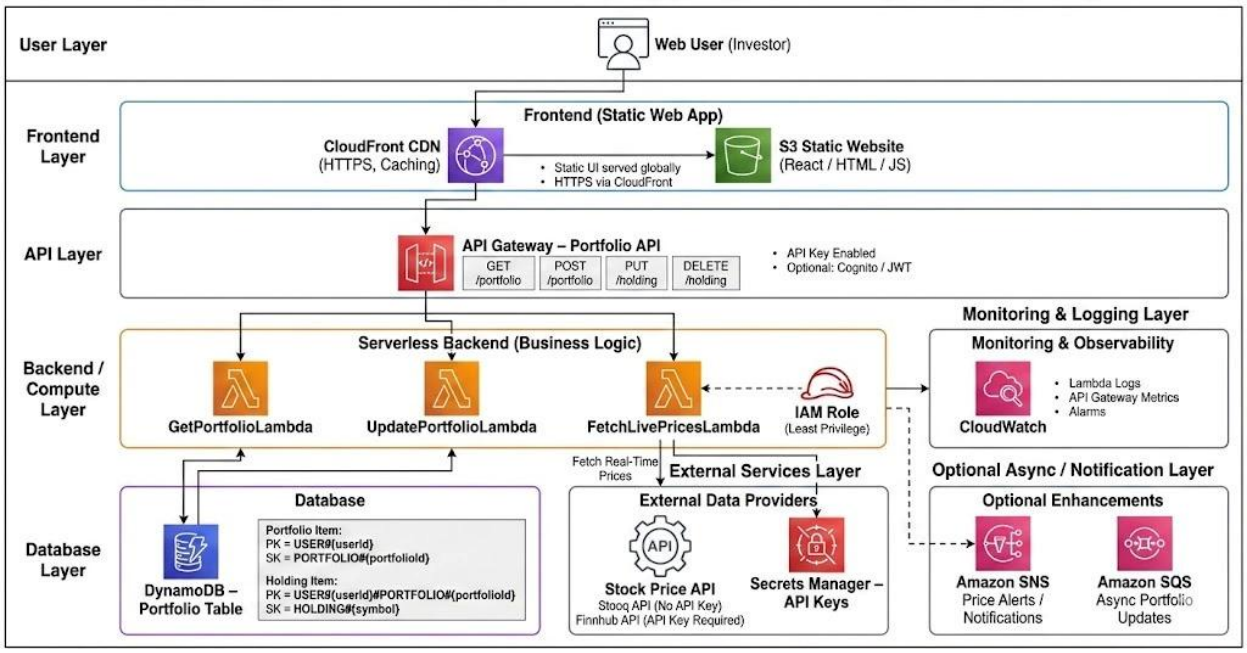


Figure 1 AWS Architecture Diagram

3.2 Implementation Steps:

Workflow Steps:

Step	Component	Action
1	User	Sends request
2	S3	Loads frontend
3	API Gateway	Routes request
4	Lambda	Processes logic
5	DynamoDB	Stores data
6	Frontend	Displays result

Table 3 Workflow Steps

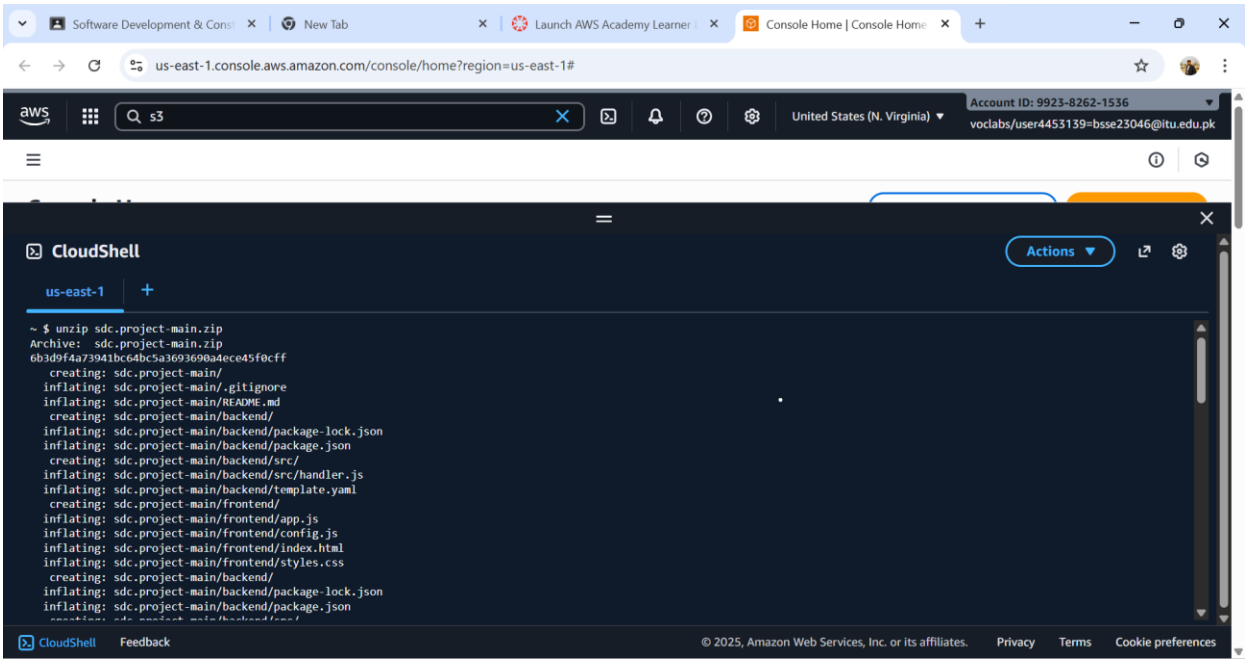


Figure 2 Uploaded my Sdc.project main .zip in the Aws Console Via Cloud Shell

```
sdc.project-main $ ls
backend  frontend  README.md
sdc.project-main $ cd backend
backend $ ls
package.json  package-lock.json  src  template.yaml
backend $ cd
~ $ cd sdc.project-main/backend
```

Figure 3 Verified that the files are uploaded

```
backend $ npm install

added 87 packages, and audited 88 packages in 4s

2 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
npm notice
npm notice New major version of npm available! 10.8.2 -> 11.7.0
npm notice Changelog: https://github.com/npm/cli/releases/tag/v11.7.0
npm notice To update run: npm install -g npm@11.7.0
npm notice
backend $ sam build

        SAM CLI now collects telemetry to better understand customer needs.

        You can OPT OUT and disable telemetry collection by setting the
        environment variable SAM_CLI_TELEMETRY=0 in your shell.
        Thanks for your help!
```

Figure 4 Installed Npm dependencies on the Aws

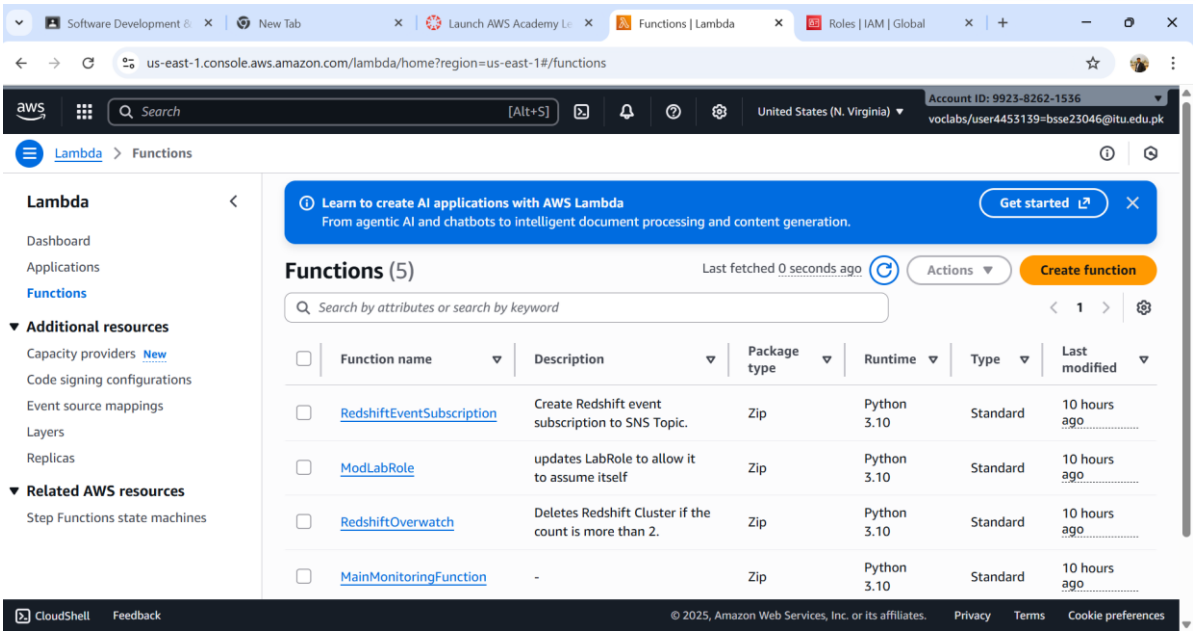


Figure 5 Lamda

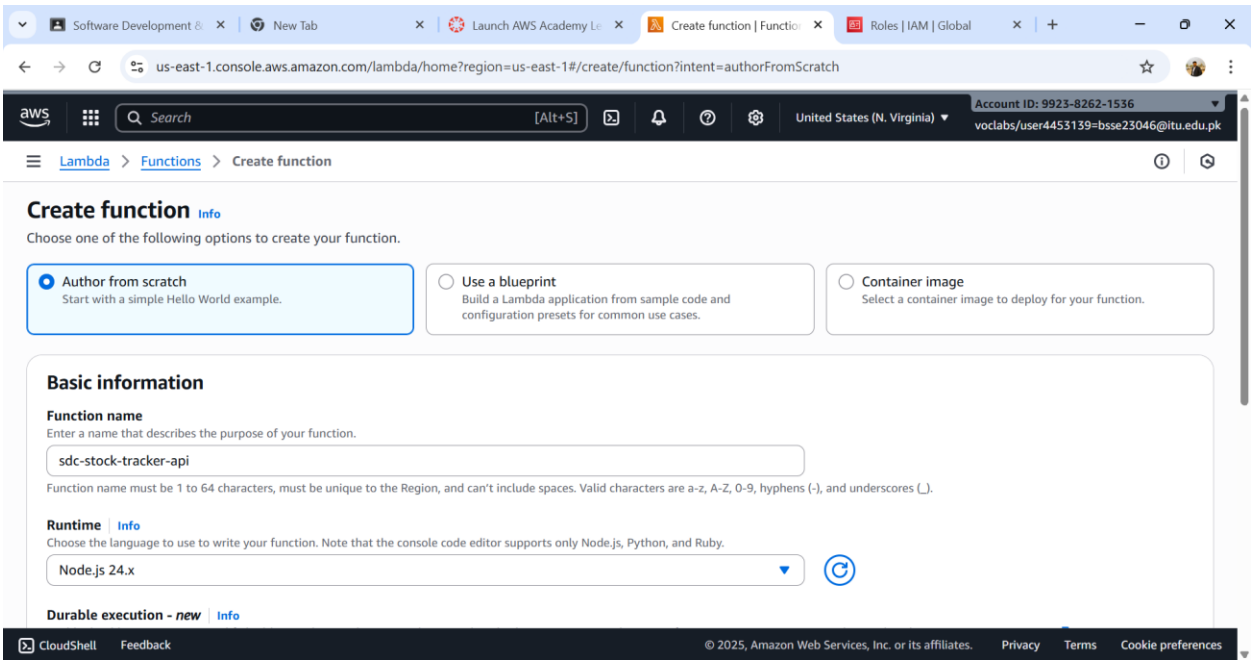


Figure 6 Lambda Creation

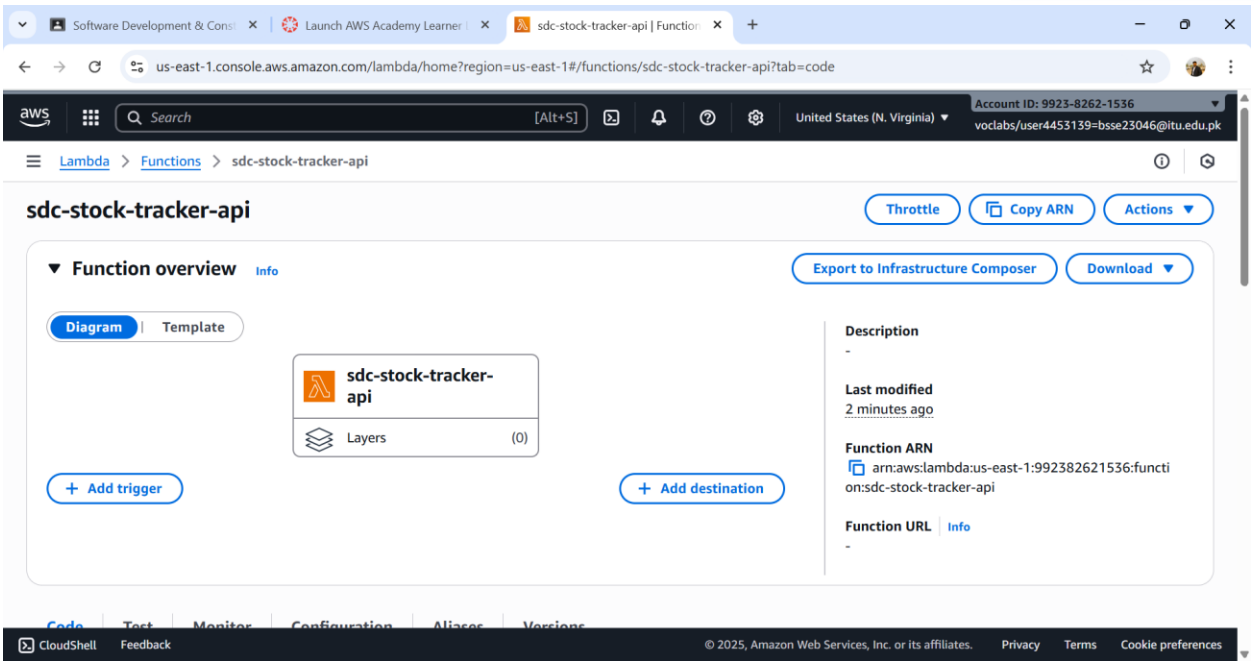


Figure 7 Lamda creation successful

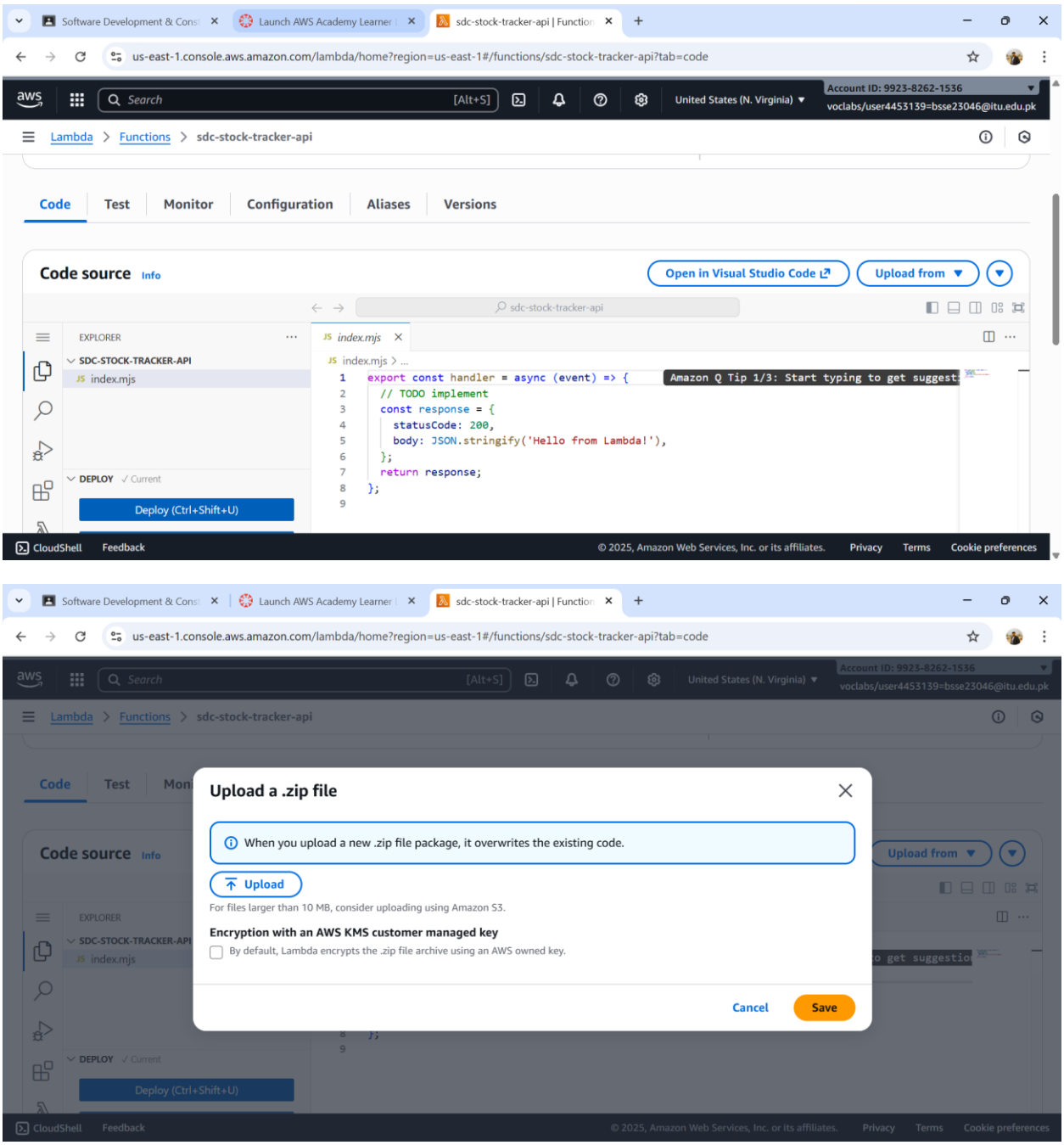


Figure 8 Uploading backed files

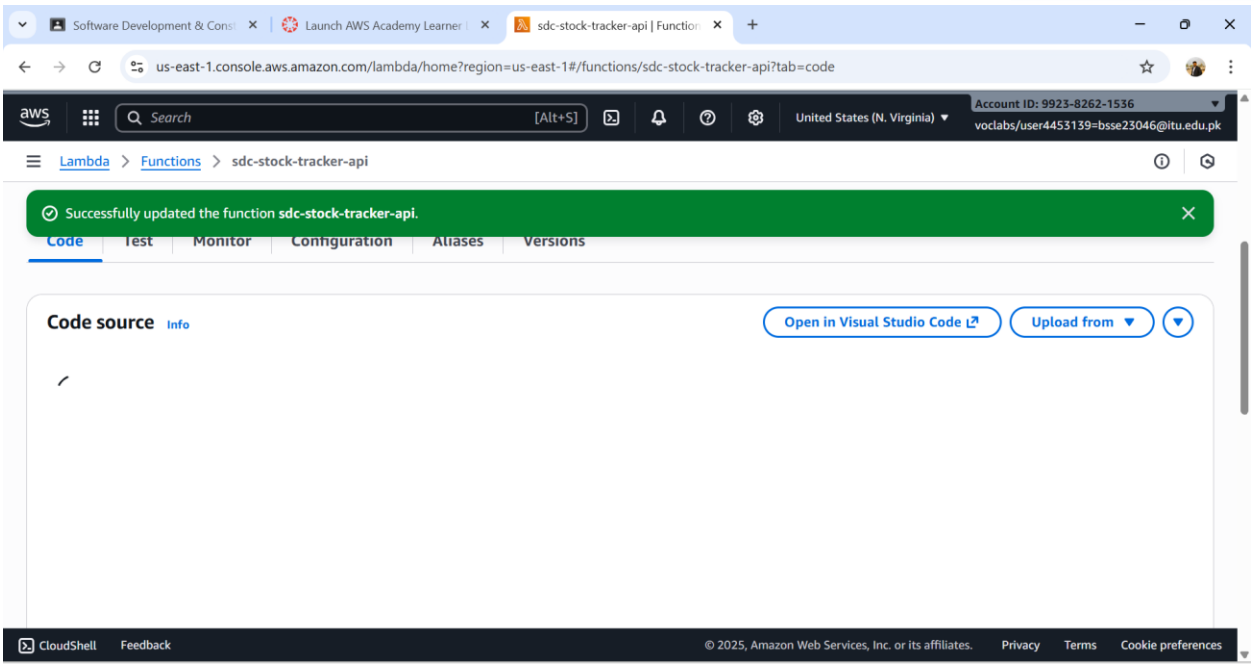
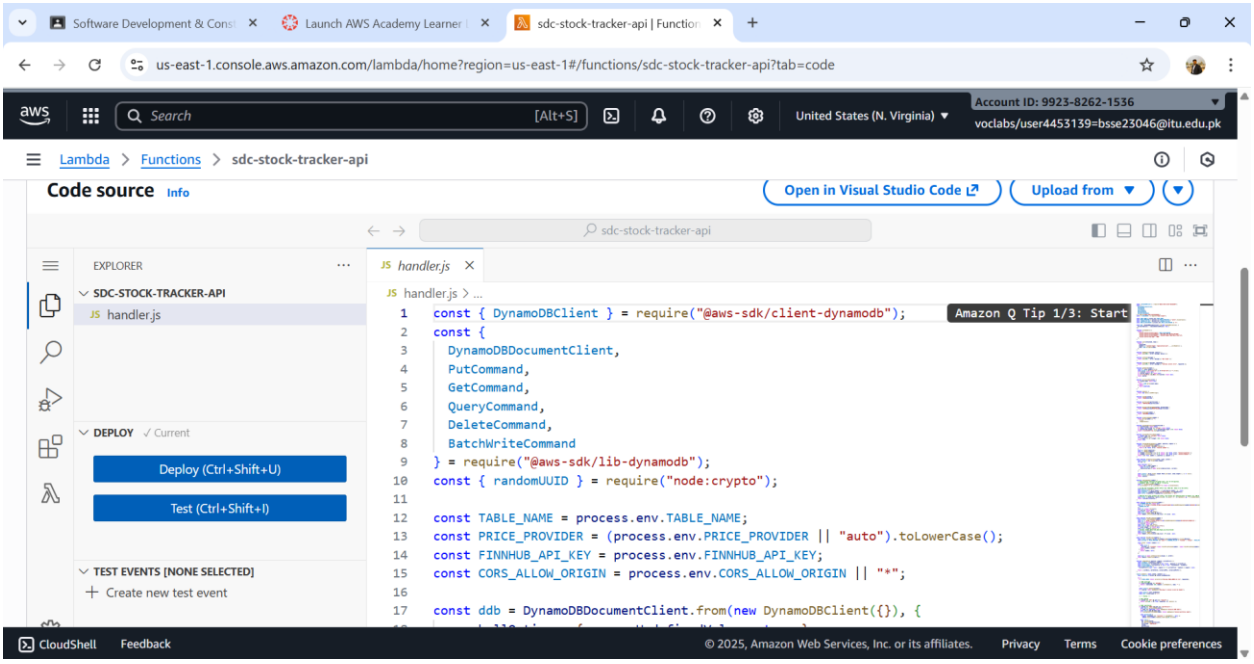


Figure 9 Handler.js uploaded



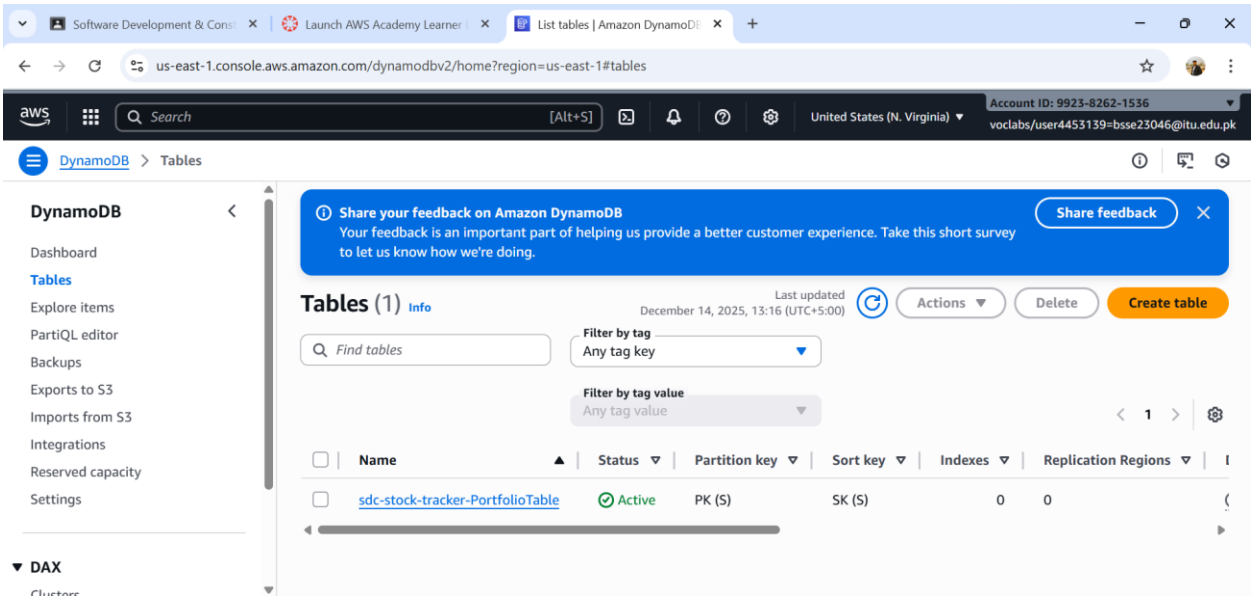
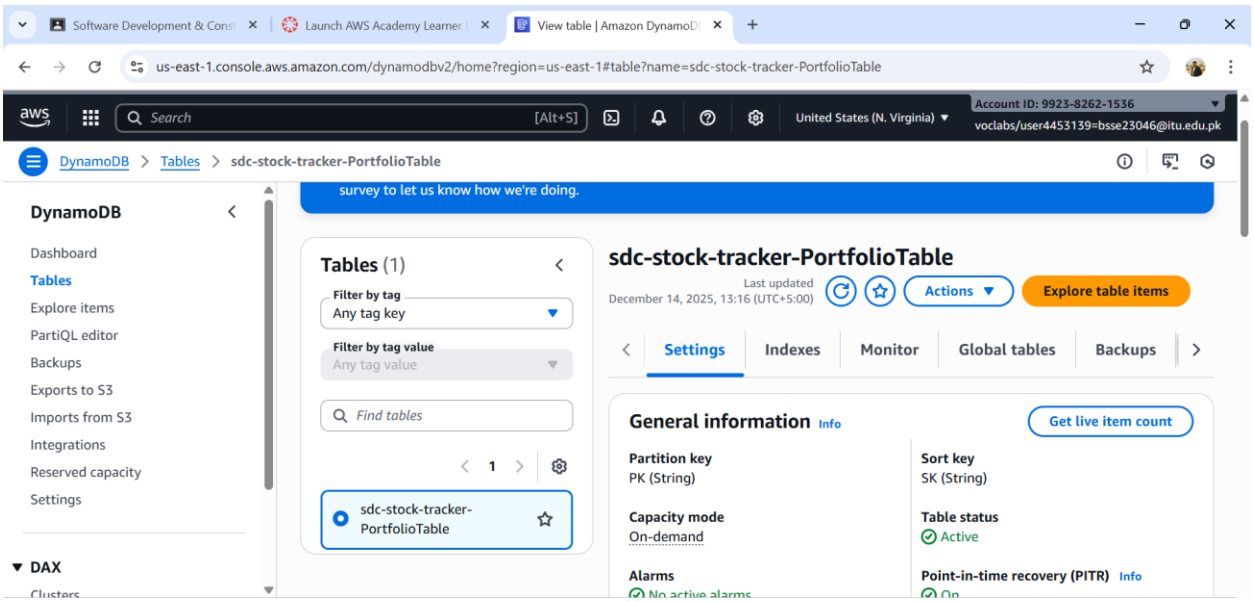


Figure 10 Dynamo.db Created



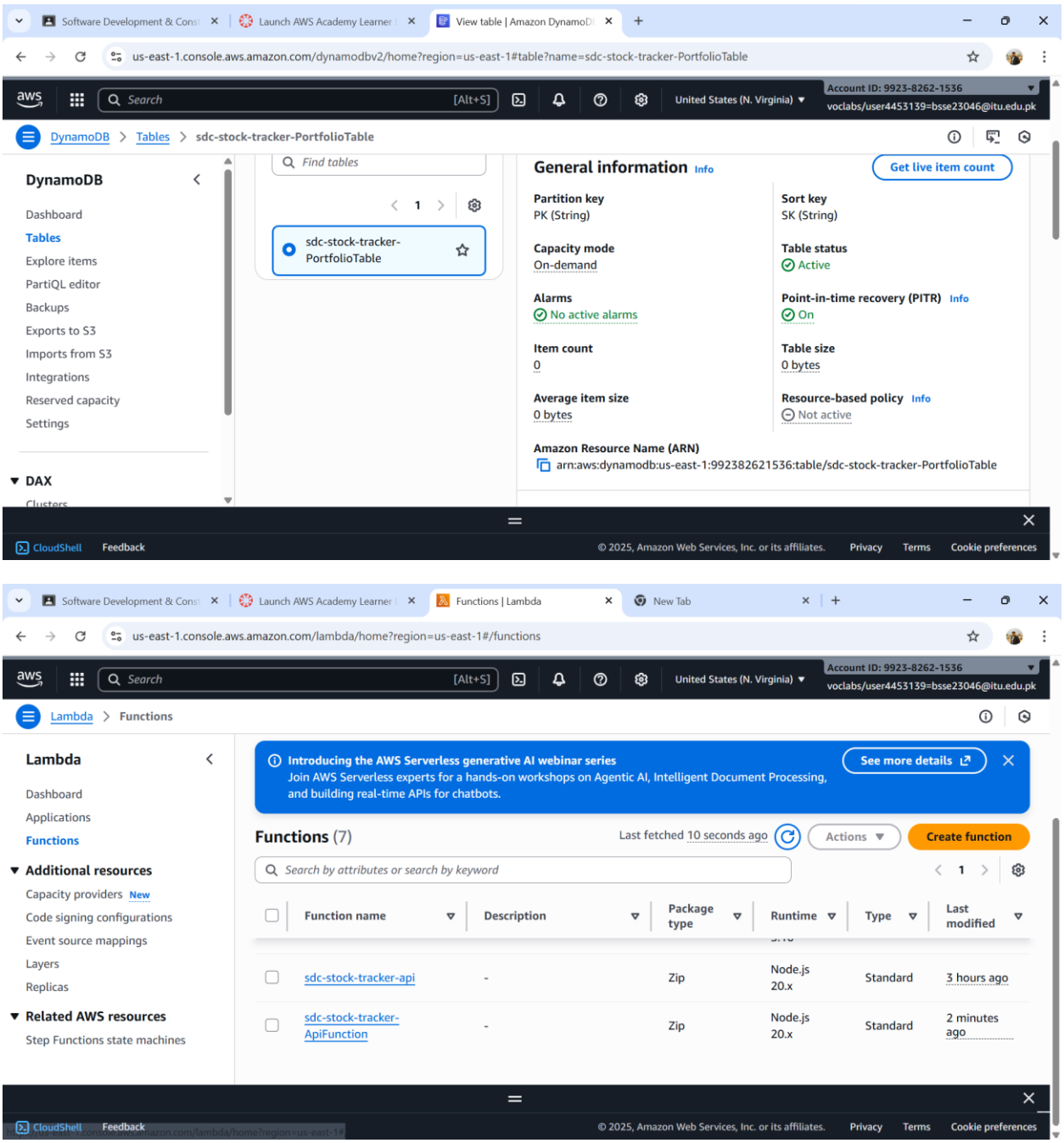


Figure 11 Aws Lambda Function Created

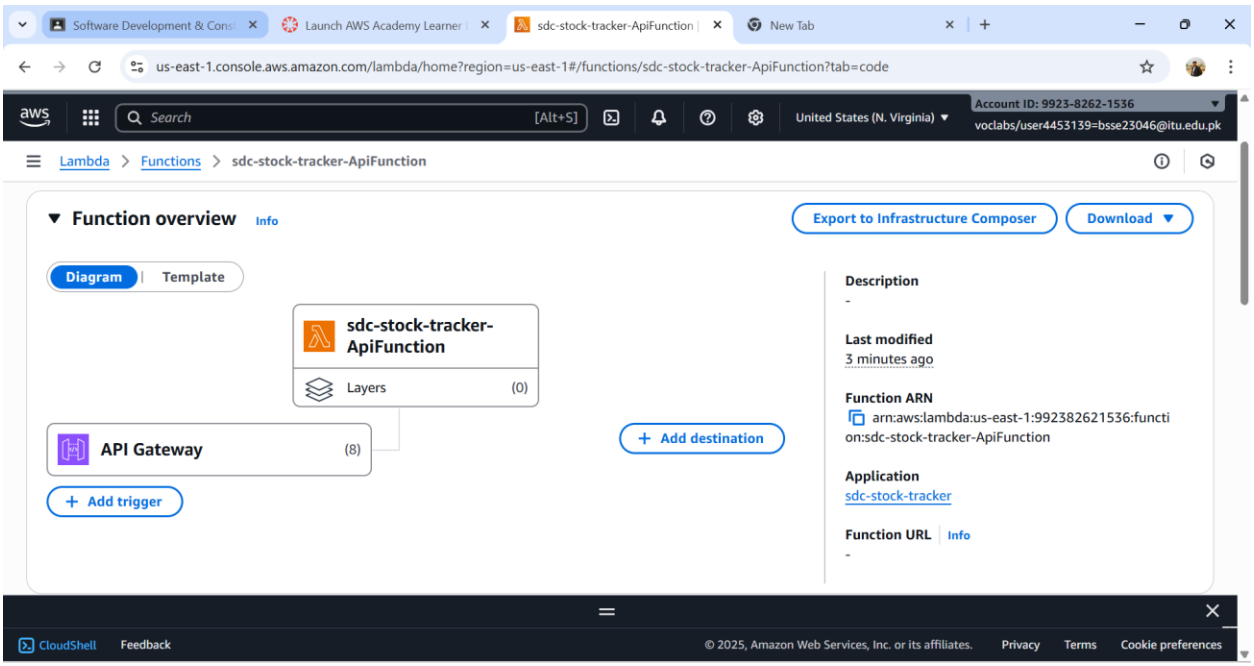


Figure 12 Api Gateways Attached

IAM Role Attached with lambda:

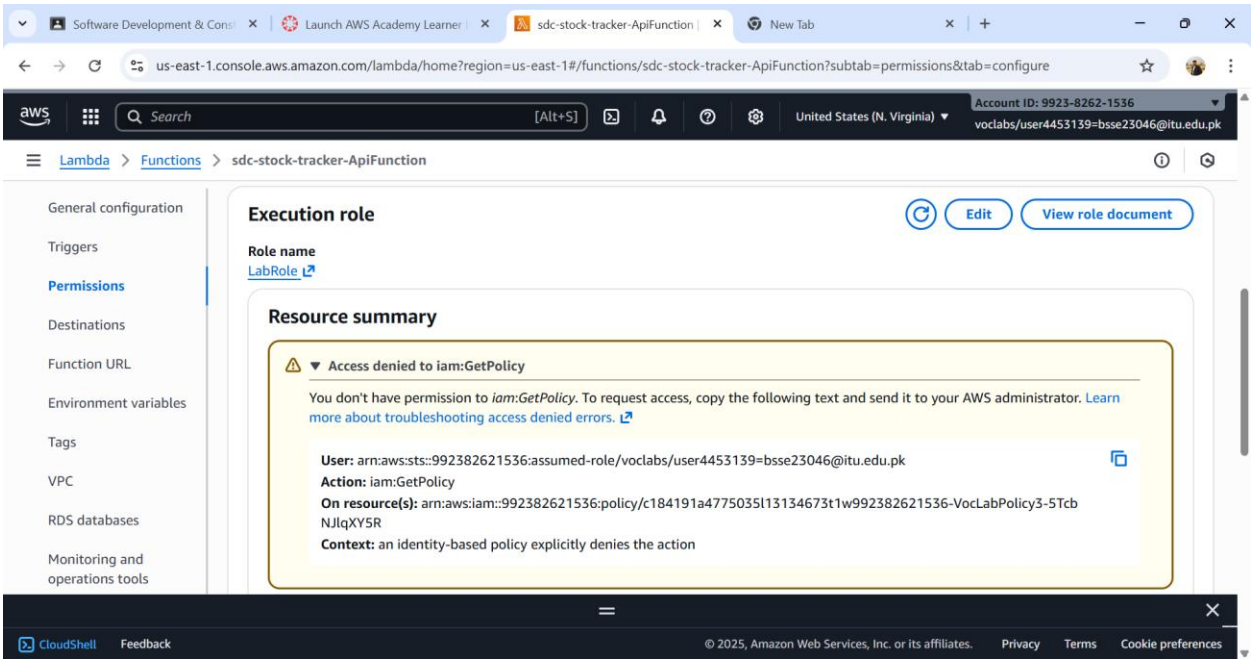


Figure 13 Lab Role Attached

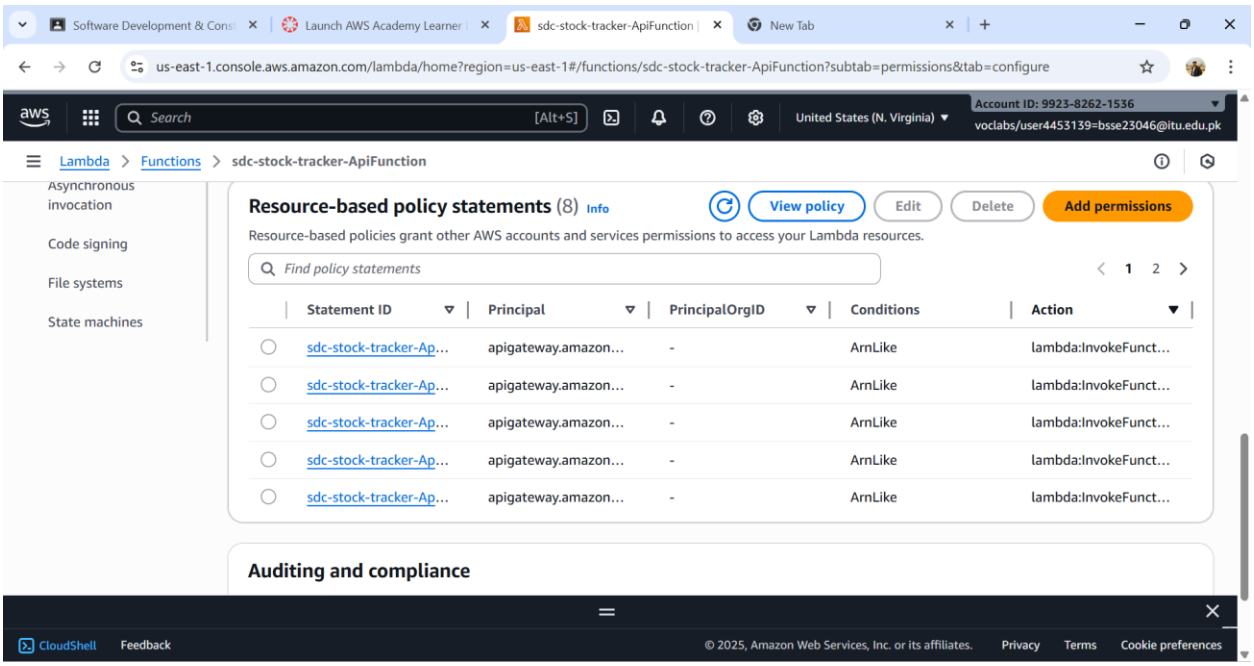


Figure 14 Resource-based policy statements

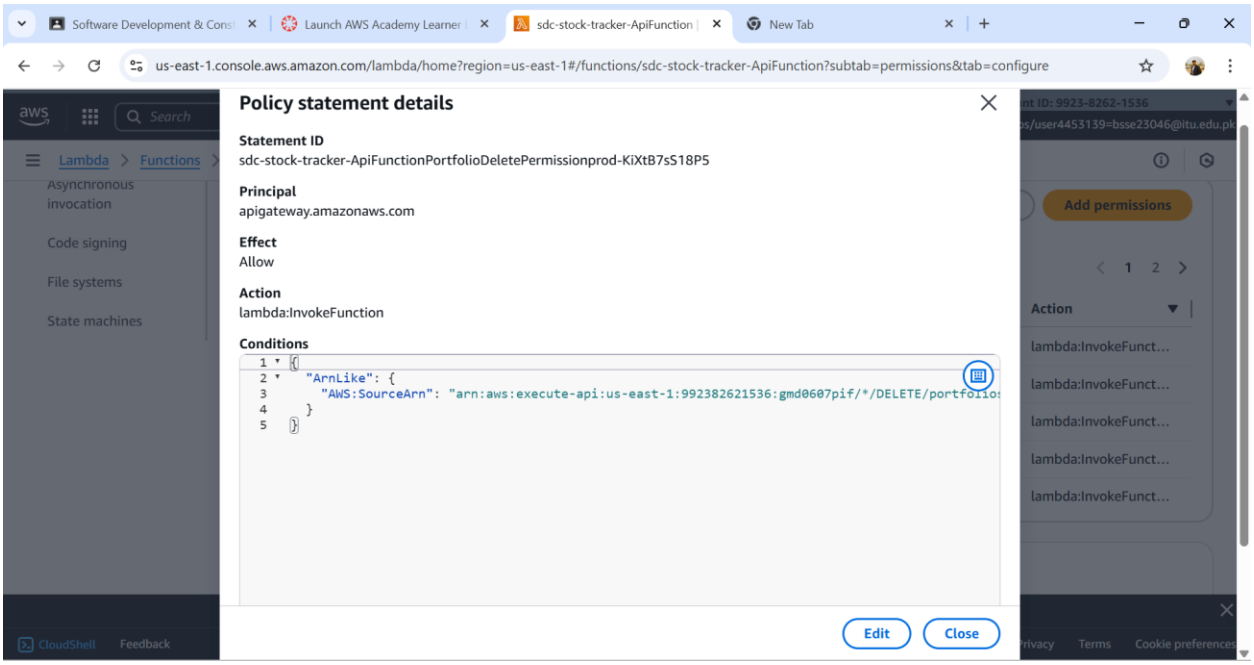


Figure 15 Delete Policy Statement

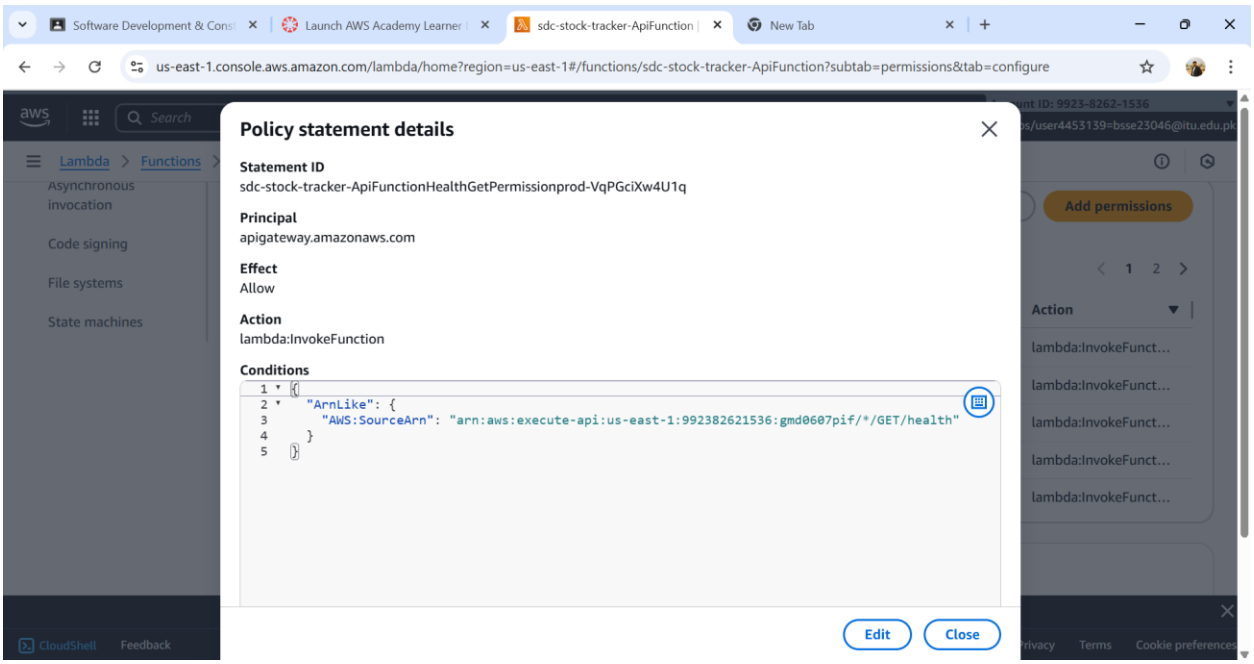


Figure 16 Get Health Policy Statement

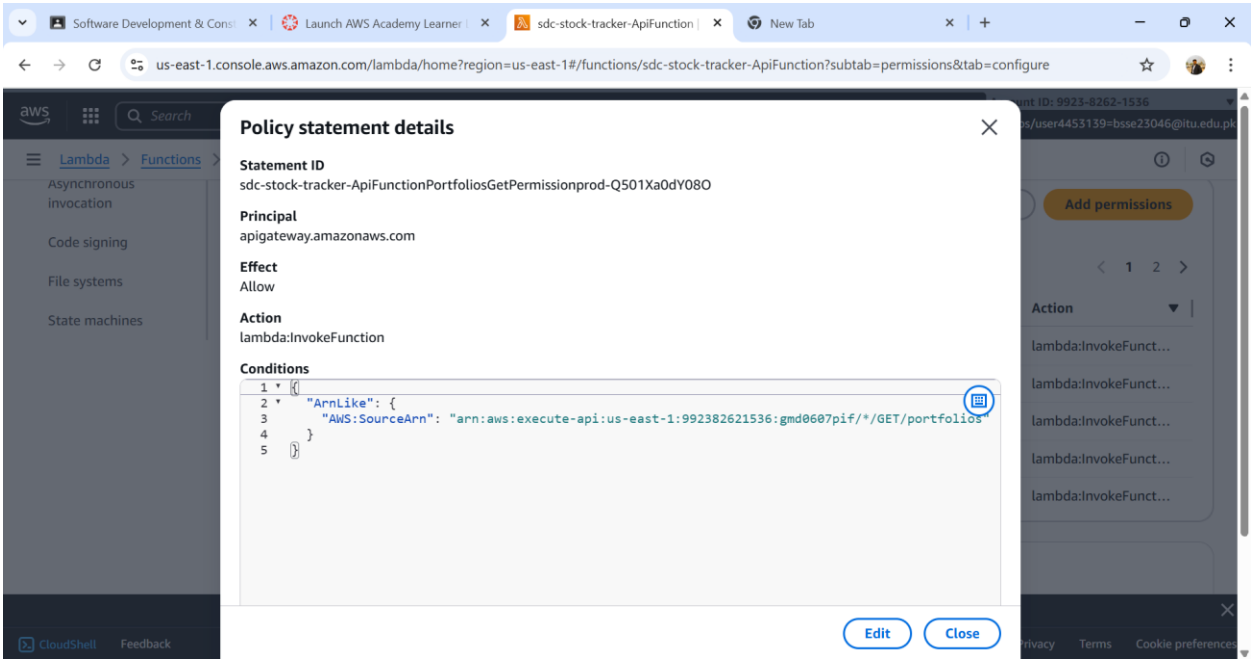


Figure 17 Get Portfolio Policy Statement

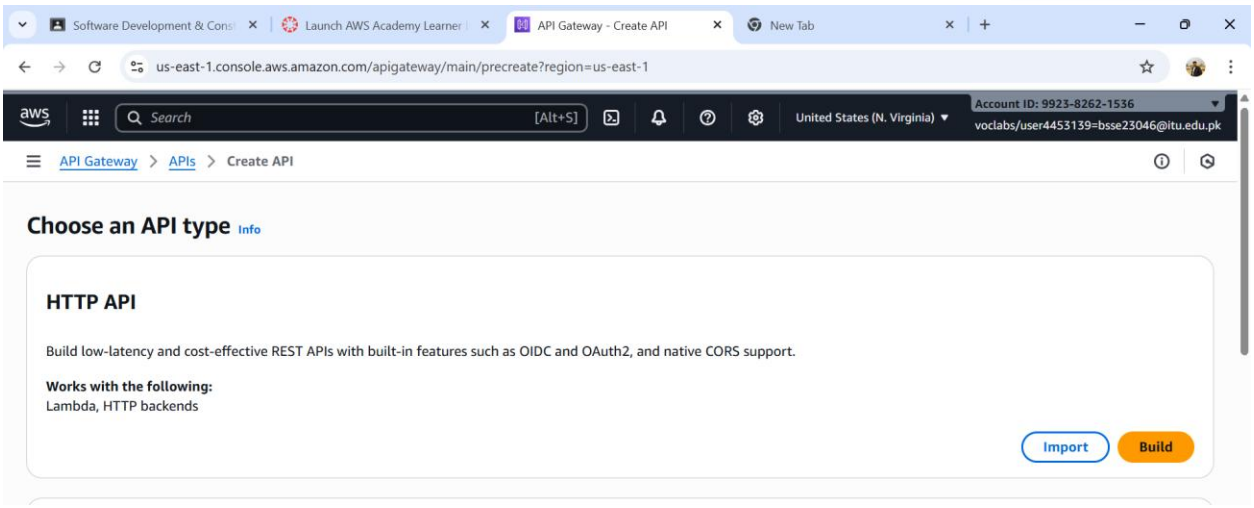


Figure 18 Api Created

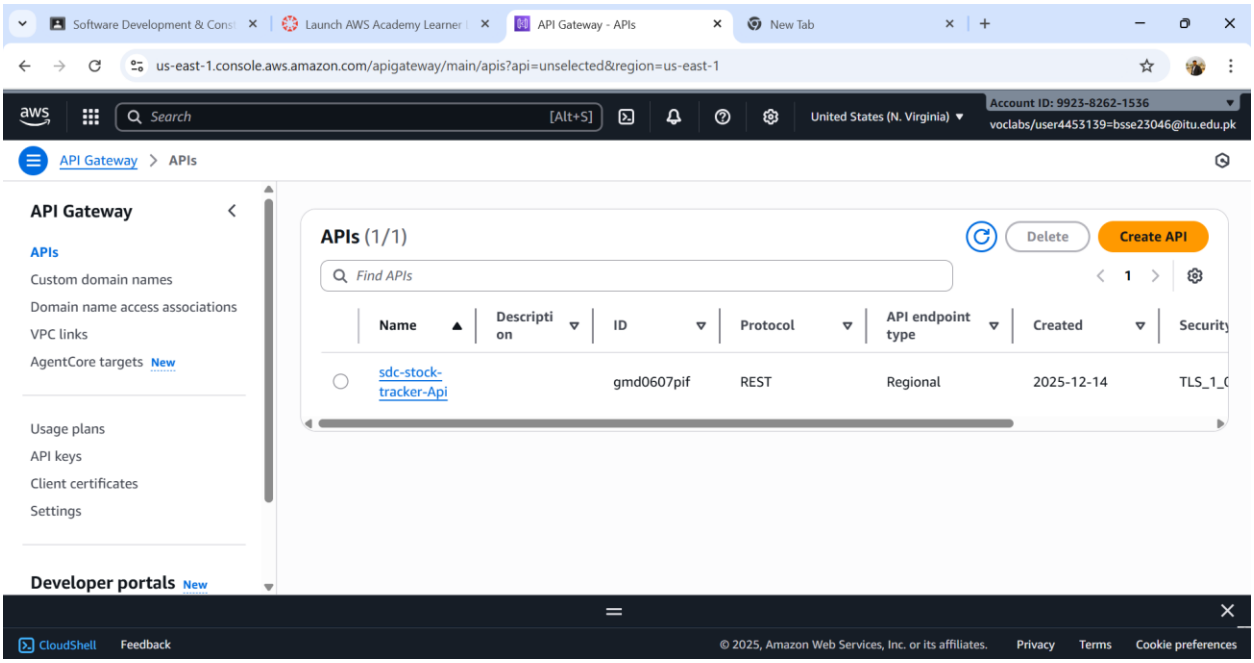


Figure 19 Sdc-stock-tracker-Api

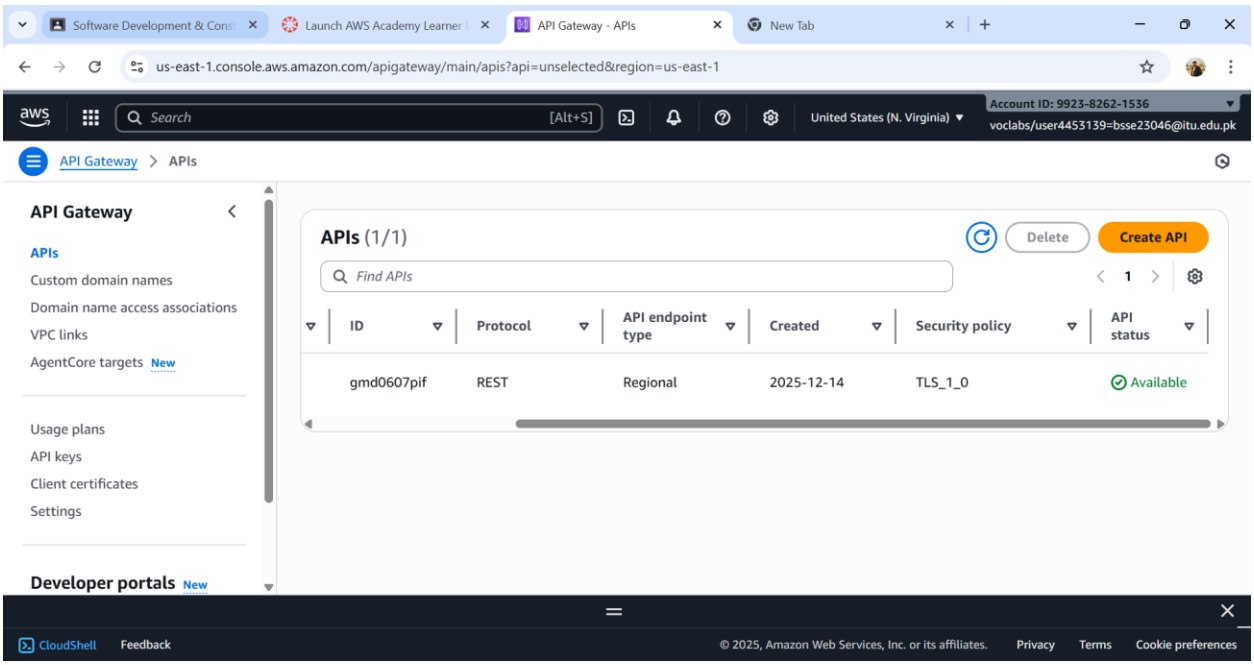


Figure 20 Status

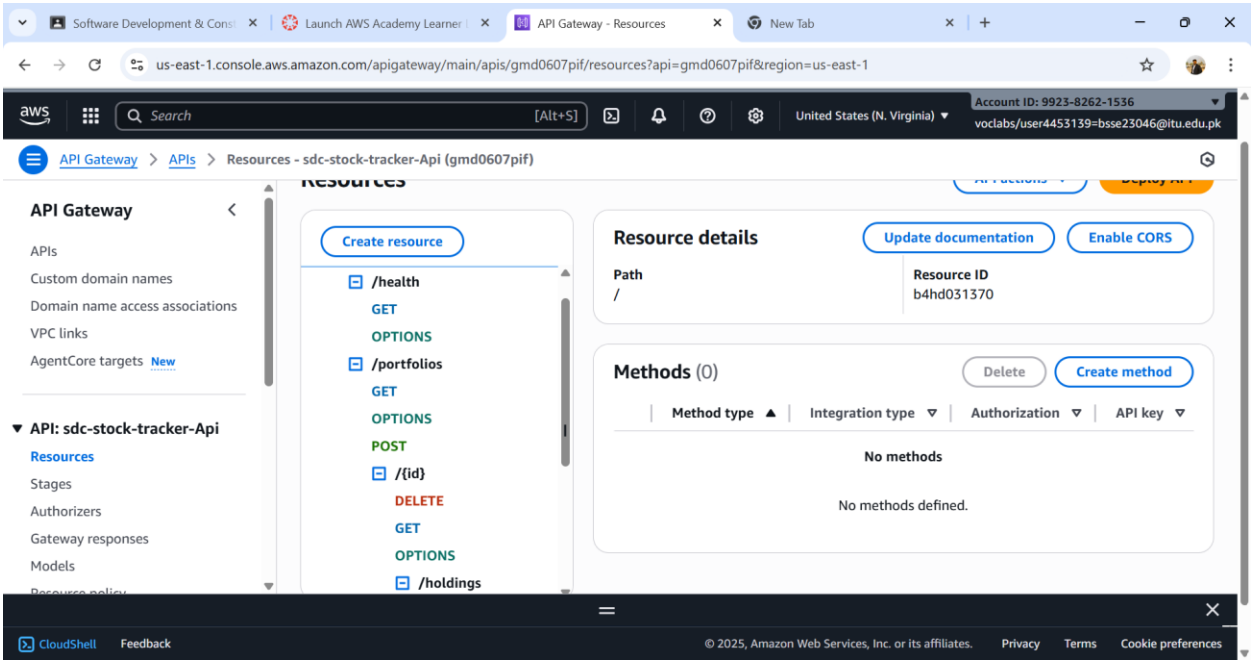
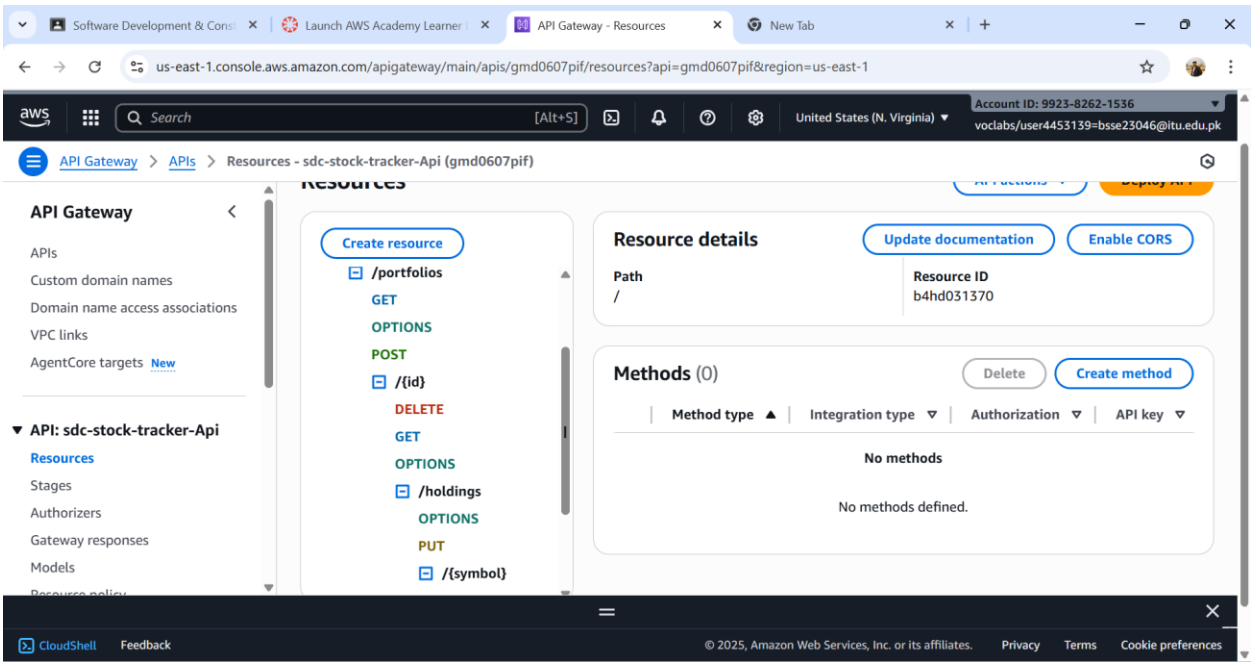


Figure 21 Resources In API Created



API's Tested via curl in aws cloud shell:

```
{"error":"Missing or invalid x-user-id header"}backend $ curl -X POST "https://gmd0607pif.execute-api.us-east-1.amazonaws.com/prod/portfolios" \
> -H "Content-Type: application/json" \
> -H "x-user-id: 123" \
> -d '{"name":"Tech Portfolio"}'
{"PK":"USER#123","SK":"PORTFOLIO#8e94c6e0-909a-4105-8e11-fab4ec543d3e","entityType":"PORTFOLIO","userId":"123","portfolioId":"8e94c6e0-909a-4105-8e11-fab4ec543d3e","name":"Tech Por
tfolio","createdAt":"2025-12-14T10:36:06.784Z","updatedAt":"2025-12-14T10:36:06.805Z"}backend $
```

Figure 22 User Created

```
backend $ curl -X GET "https://gmd0607pif.execute-api.us-east-1.amazonaws.com/prod/portfolios" \
> -H "x-user-id: 123"
{"items":[{"portfolioId":"8e94c6e0-909a-4105-8e11-fab4ec543d3e","entityType":"PORTFOLIO","userId":"123","updatedAt":"2025-12-14T10:36:06.805Z","createdAt":"2025-12-14T10:36:06.784Z
","PK":"USER#123","name":"Tech Portfolio","SK":"PORTFOLIO#8e94c6e0-909a-4105-8e11-fab4ec543d3e"}]}backend $
```

Figure 23 Get A specific Portfolio

```
backend $ curl -X PUT "https://gmd0607pif.execute-api.us-east-1.amazonaws.com/prod/portfolios/8e94c6e0-909a-4105-8e11-fab4ec543d3e/holdings" \
> -H "Content-Type: application/json" \
> -H "x-user-id: 123" \
> -d '{"symbol":"AAPL","quantity":10,"price":150,"avgCost":150}'
{"PK":"USER#123#PORTFOLIO#8e94c6e0-909a-4105-8e11-fab4ec543d3e","SK":"HOLDING#AAPL","entityType":"HOLDING","userId":"123","portfolioId":"8e94c6e0-909a-4105-8e11-fab4ec543d3e","symb
ol":"AAPL","quantity":10,"avgCost":150,"createdAt":"2025-12-14T10:53:35.995Z","updatedAt":"2025-12-14T10:53:35.995Z"}backend $
```

Figure 24 Add holdings

```
{"error":"Query param 'symbols' is required (comma-separated)"}backend $ curl -X GET "https://gmd0607pif.execute-api.us-east-1.amazonaws.com/prod/prices?symbols=AAPL" \
> -H "x-user-id: 123"
{"prices":{"AAPL":null},"ts":"2025-12-14T10:54:43.411Z"}backend $
```

Figure 25 Check stock price

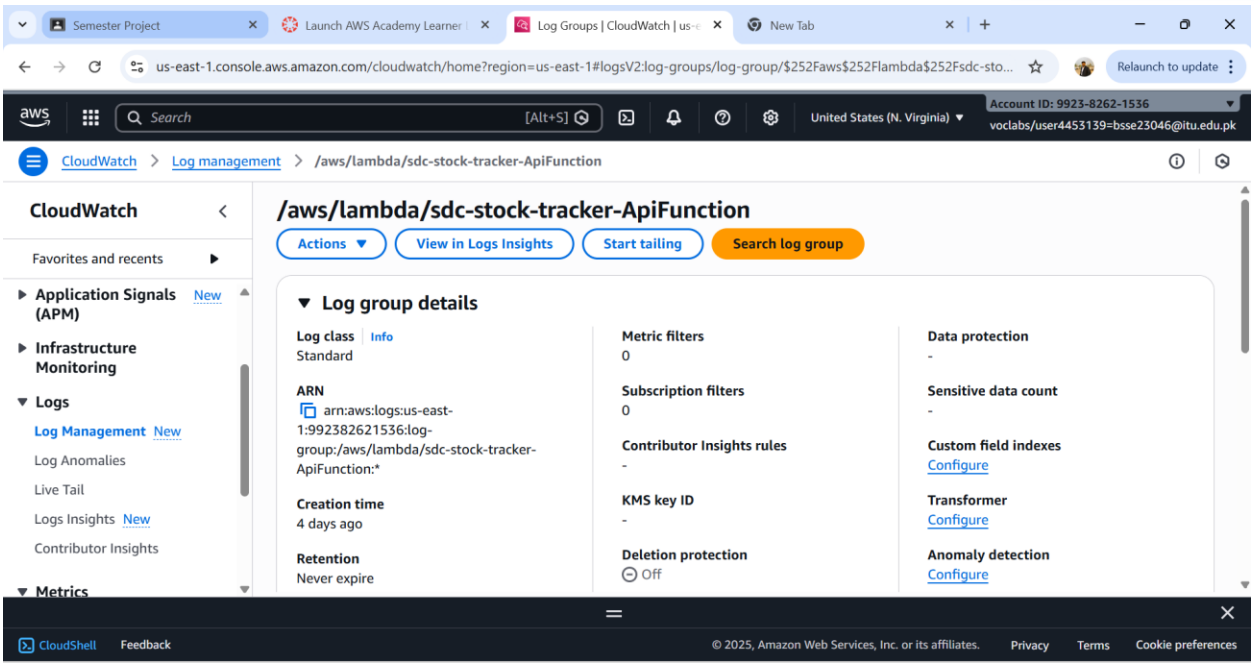


Figure 26 Cloud Watch Log Maintenance

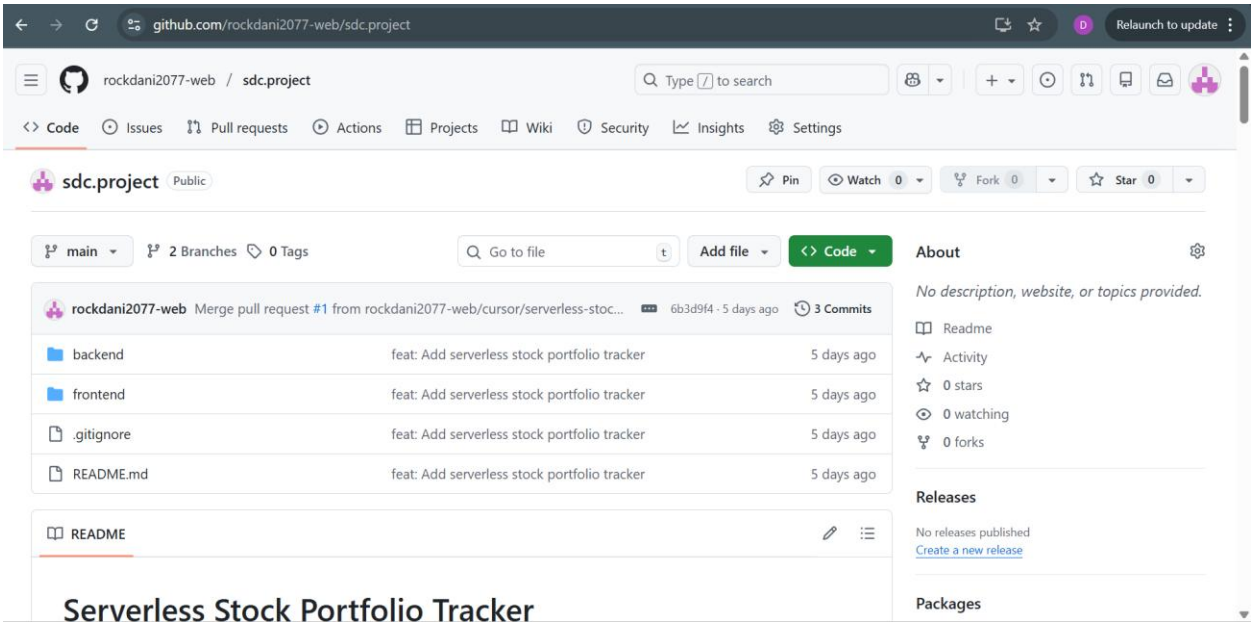


Figure 27 Docker +kubernets+Jenkins

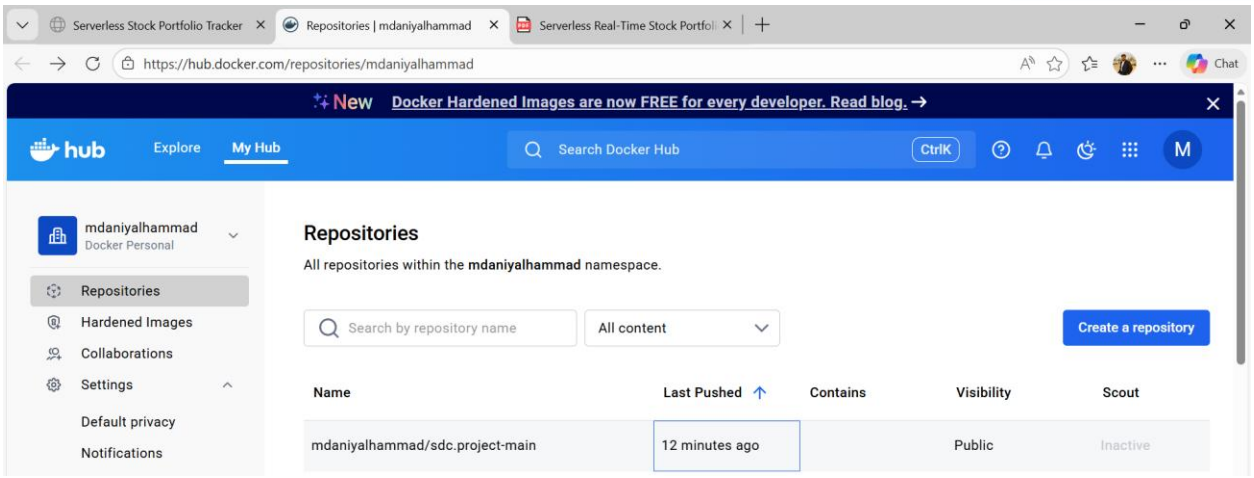


Figure 28 Docker Repository

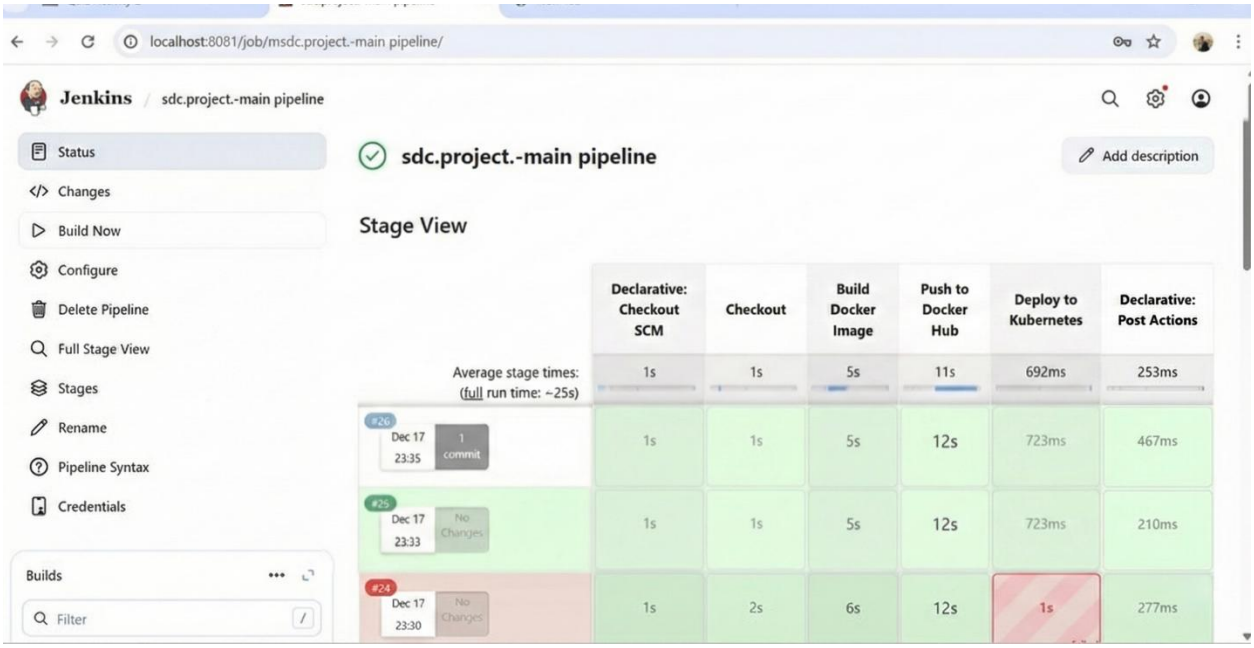


Figure 29 Jenkins Automation

API Endpoints:

Method	Endpoint	Description
POST	/portfolio	Add stock
GET	/portfolio	View portfolio
GET	/price	Fetch stock price
DELETE	/portfolio	Remove stock

Table 4 API Endpoints

3.3 Stock Tracker Project – Workflow Explanation

3.3.1 User Interaction (Frontend Layer)

- The user opens the **web application** in the browser.
- The **frontend is hosted on Amazon S3** as a static website.
- (Optional) **CloudFront** is used for fast global content delivery.

User actions include:

- Adding a stock (e.g., AAPL)
- Viewing portfolio
- Checking market value and profit/loss

The frontend **never talks directly to the database**.

3.3.2 API Request (API Gateway)

- When a user performs an action, the frontend sends an **HTTP request** to **Amazon API Gateway**.
- Example:
    - GET /portfolio
    - POST /portfolio

- GET /price?symbol=AAPL

Why API Gateway?

- Provides a secure REST API
- Handles routing, throttling, and CORS

3.3.3 Backend Logic (AWS Lambda)

- API Gateway triggers an **AWS Lambda function**.
- Lambda contains the **business logic**:
  - Add stock to portfolio
  - Fetch current price
  - Calculate market value & P/L
  - Read/write data

Why Lambda?

- Serverless (no server management)
- Scales automatically
- Pay-per-use

3.3.4 Market Price Fetching (Finn hub / Mock Provider)

Inside Lambda:

- If **Finnhub API key exists**:
  - Lambda calls **Finnhub API**
  - Gets real-time stock price
- If **no API key** (lab/demo mode):
  - Lambda uses a **mock price provider**

This decision is controlled by:

PRICE_PROVIDER = auto

FINNHUB_API_KEY = (optional)

3.3.5 Data Storage (Amazon DynamoDB)

- Portfolio data is stored in **DynamoDB**:
    - Stock symbol
    - Quantity
    - Buy price

Why DynamoDB?

- Fully managed NoSQL database
- Fast read/write performance
- Scales automatically

3.3.6 Calculations (Inside Lambda)

Lambda calculates:

Market Value

Market Value = Current Price × Quantity

Unrealized Profit / Loss

Unrealized P/L = (Current Price − Buy Price) × Quantity

Results are returned to the frontend.

3.3.7 Response Back to User

- Lambda sends response → API Gateway
- API Gateway returns JSON → Frontend
- Frontend updates the UI dynamically

User sees:

- Updated portfolio
- Market value
- Profit / Loss

3.3.8 Monitoring & Logs (CloudWatch)

- All Lambda executions are logged in **Amazon CloudWatch**
- Used for:
    - Debugging
    - Performance monitoring
    - Error tracking

3.3.9 Complete Architecture Flow (One Line)

User → S3 Frontend → API Gateway → Lambda → Finnhub/DynamoDB → Lambda → API Gateway → Frontend

3.3.10 Key Benefits

- Fully **serverless architecture**
- Scalable & cost-efficient
- Secure API layer
- Real-time or mock market data support
- Easy to deploy using **AWS SAM**

30-Second Summary:

"This project is a serverless stock portfolio tracker where the frontend is hosted on S3, API Gateway handles HTTP requests, Lambda processes business logic and fetches market prices using Finnhub, DynamoDB stores portfolio data, and CloudWatch is used for monitoring."

4 End Point URL

http://sdc-stock-frontend-12345.s3-website-us-east-1.amazonaws.com/