



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

Table of Contents	2
1 Project Proposal, Executive Summary & Problem Definition:	6
1.1 Executive Summary:.....	6
1.2 Introduction:.....	6
1.3 Problem Statement:.....	6
1.4 Aim & Objectives:	7
1.4.1 Aim:	7
1.4.2 Objectives:	7
2 Equations & Formulae	8
2.1 Average Buy Price (Cost Basis).....	8
2.1.1 Purpose:	8
2.1.2 Formula:.....	8
2.1.3 Where:.....	8
2.2 Market Value of Holdings	8
2.2.1 Purpose:	8
2.2.2 Formula:.....	8
2.3 Total Investment (Cost).....	8
2.3.1 Purpose:	8
2.3.2 Formula:.....	8
2.4 Unrealized Profit / Loss (P&L).....	8
2.4.1 Purpose:	8
2.4.2 Formula:.....	8
2.5 Profit / Loss Percentage	9
2.5.1 Purpose:	9
2.5.2 Formula:.....	9
2.6 Realized Profit / Loss	9
2.6.1 Purpose:	9
2.6.2 Formula:.....	9
3 AWS Architecture Design, Implementation Steps & Screenshots;.....	10
3.1 AWS Architecture Diagram:	10

3.2	Implementation Steps:	10
3.3	Stock Tracker Project – Workflow Explanation	26
3.3.1	User Interaction (Frontend Layer)	26
3.3.2	API Request (API Gateway).....	26
3.3.3	Backend Logic (AWS Lambda).....	27
3.3.4	Market Price Fetching (Finn hub / Mock Provider).....	27
3.3.5	Data Storage (Amazon DynamoDB)	28
3.3.6	Calculations (Inside Lambda)	28
3.3.7	Response Back to User	28
3.3.8	Monitoring & Logs (CloudWatch).....	29
3.3.9	Complete Architecture Flow (One Line)	29
3.3.10	Key Benefits.....	29
4	End Point URL.....	29

List Of Table:

Table 1 Problem vs Solution	7
Table 2 Formula Summary.....	9
Table 3 Workflow Steps	10
Table 4 API Endpoints	26

List Of Equations:

Equation 1 Average Buy Price	8
Equation 2 Total Amount Invested.....	8
Equation 3 Market Value	8
Equation 4 Total Investment	8
Equation 5 Unrealized P&L.....	8
Equation 6 P&L Percentage	9
Equation 7 Realized P&L	9

List Of Figures:

Figure 1 AWS Architecture Diagram	10
Figure 2 Uploaded my Sdc.project main .zip in the Aws Console Via Cloud Shell	11
Figure 3 Verified that the files are uploaded.....	11
Figure 4 Installed Npm dependencies on the Aws	11
Figure 5 Lamda	12
Figure 6 Lambda Creation	12
Figure 7 Lamda creation successful.....	13
Figure 8 Uploading backed files	14
Figure 9 Handler.js uploaded	15
Figure 10 Dynamo.db Created.....	16
Figure 11 Aws Lambda Function Created.....	17
Figure 12 Api Gateways Attached	18
Figure 13 Lab Role Attached	18
Figure 14 Resource-based policy statements	19
Figure 15 Delete Policy Statement	19
Figure 16 Get Health Policy Statement	20
Figure 17 Get Portfolio Policy Statement.....	20
Figure 18 Api Created	21
Figure 19 Sdc-stock-tracker-API.....	21
Figure 20 Status	22
Figure 21 Resources In API Created.....	22
Figure 22 User Created	23
Figure 23 Get A specific Portfolio	23
Figure 24 Add holdings.....	23
Figure 25 Check stock price	23
Figure 26 Cloud Watch Log Maintenance	24
Figure 27 Docker +kubernets+Jenkins	24
Figure 28 Docker Repository	25
Figure 29 Jenkins Automation	25

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	$\Sigma(\text{Buy Price} \times \text{Qty})$	Total cost
4	Unrealized P/L	$(\text{CP} - \text{BP}) \times \text{Qty}$	Open profit
5	P/L %	$(\text{P/L} \div \text{Investment}) \times 100$	Performance

Table 2 Formula Summary

3 AWS Architecture Design, Implementation Steps & Screenshots;

3.1 AWS Architecture Diagram:

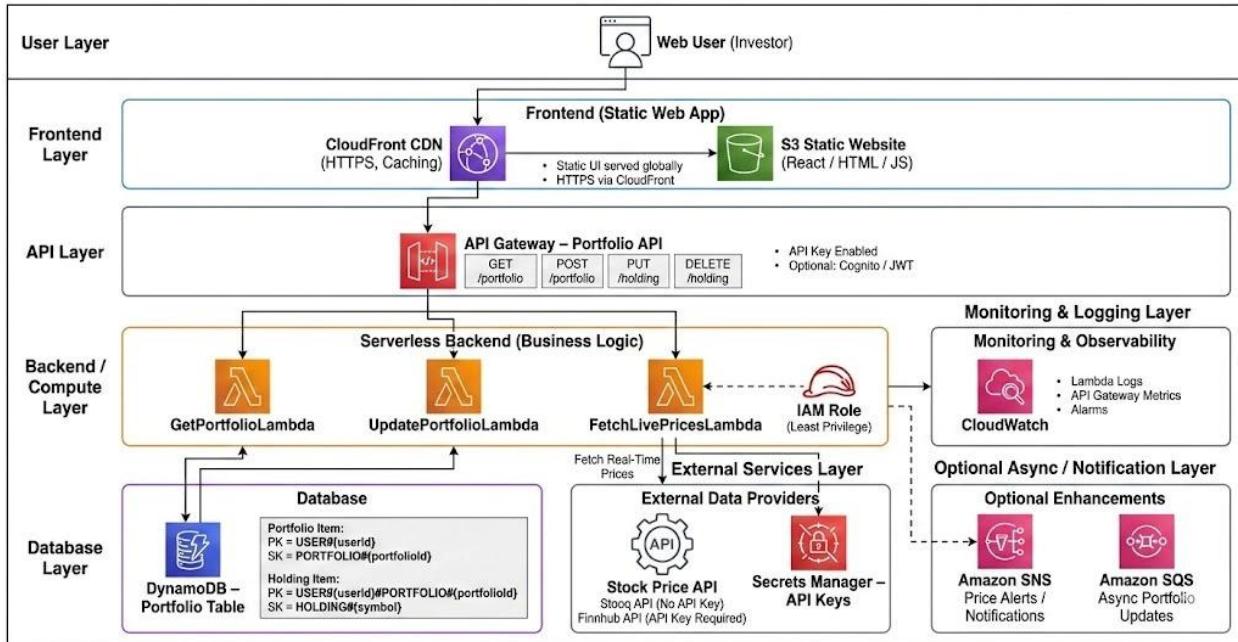


Figure 1 AWS Architecture Diagram

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

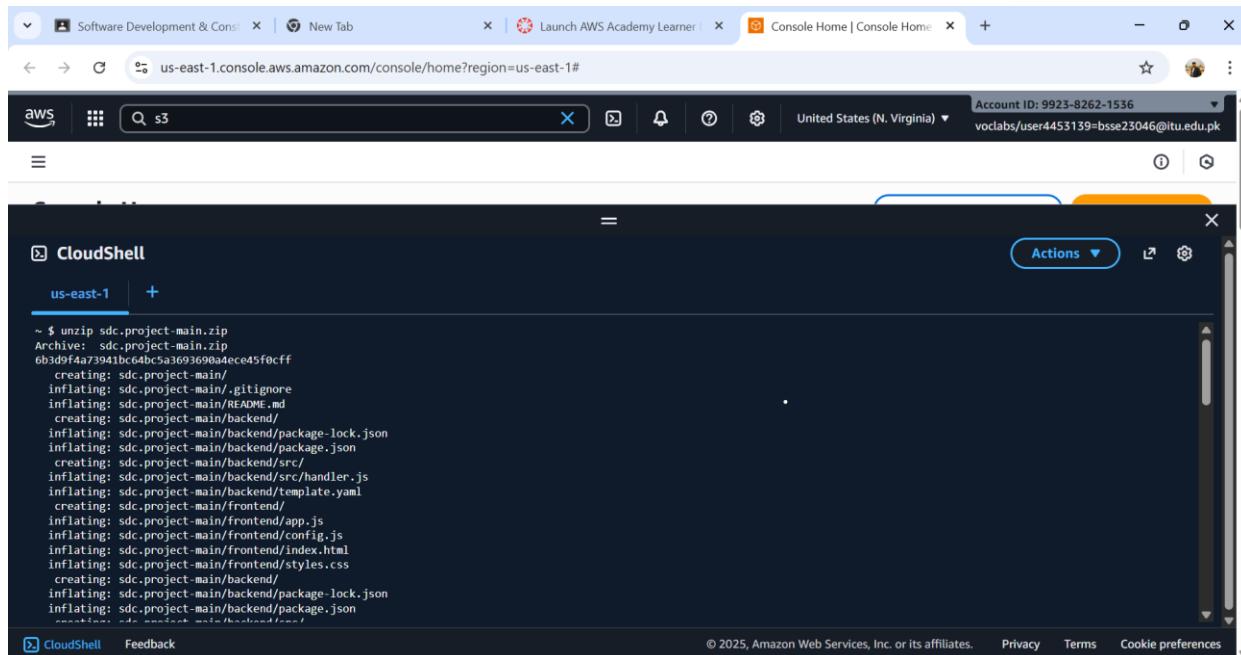


Figure 2 Uploaded my Sdc.project main .zip in the Aws Console Via Cloud Shell

```
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

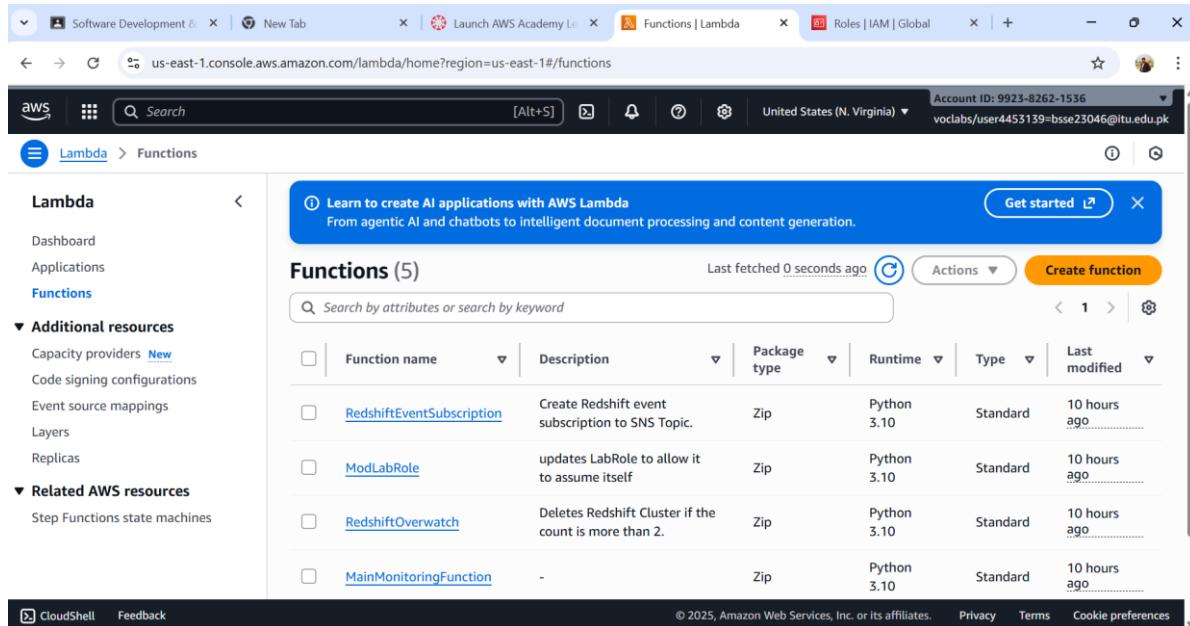
```
backend $ npm install
added 87 packages, and audited 88 packages in 4s
2 packages are looking for funding
  run `npm fund` for details

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.

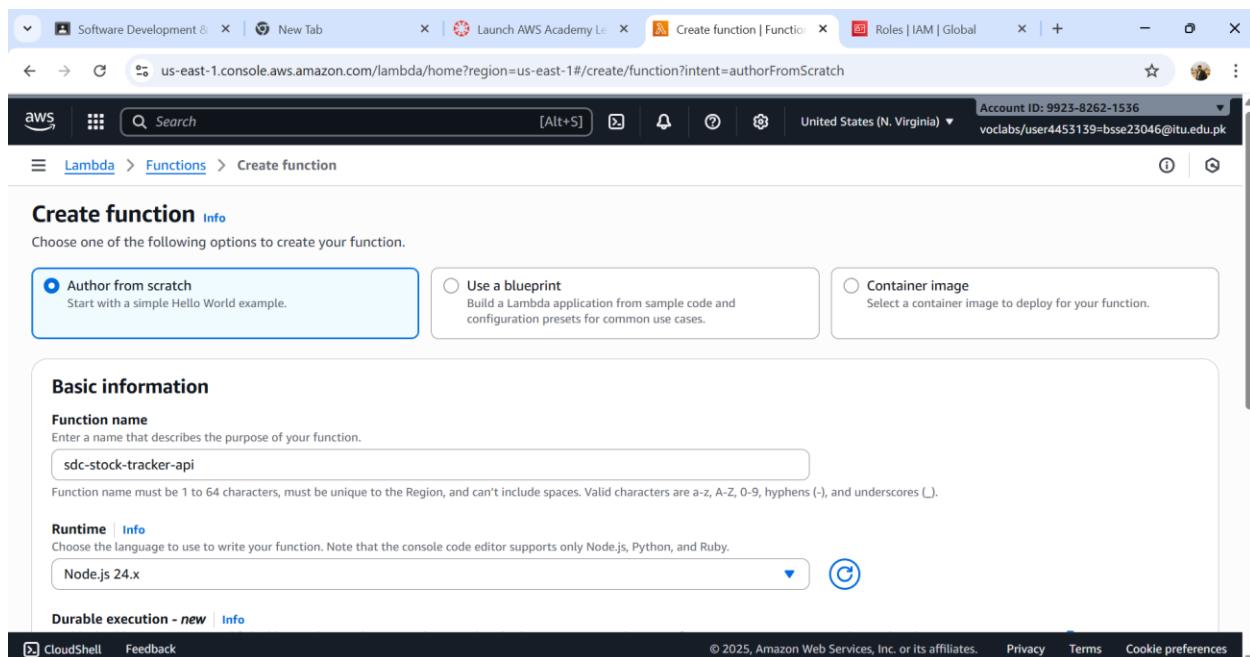
You can OPT OUT and disable telemetry collection by setting the
environment variable SAM_CLI_TELEMETRY=0 in your shell.
Thanks for your help!
```

Figure 4 Installed Npm dependencies on the Aws



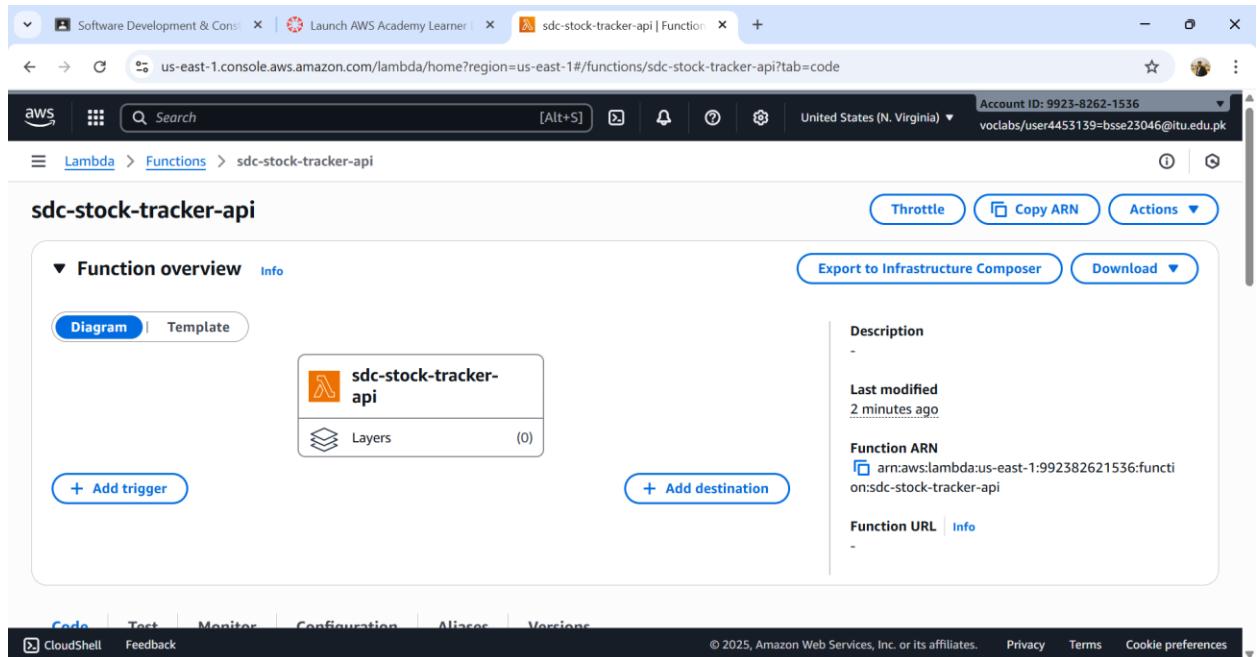
The screenshot shows the AWS Lambda console interface. On the left, there's a sidebar with navigation links like Dashboard, Applications, Functions, Additional resources, and Related AWS resources. The main area is titled "Functions (5)" and displays a table of existing Lambda functions. The columns include Function name, Description, Package type, Runtime, Type, and Last modified. The listed functions are RedshiftEventSubscription, ModLabRole, RedshiftOverwatch, and MainMonitoringFunction, all created 10 hours ago.

Figure 5 Lamda



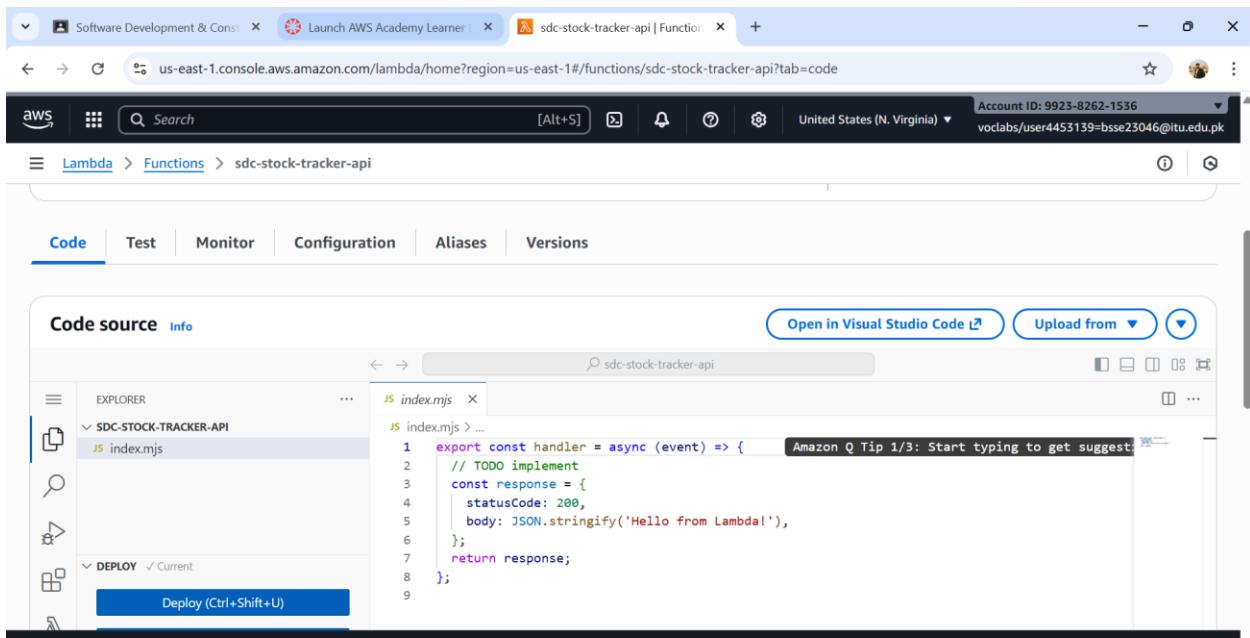
The screenshot shows the "Create function" wizard. It starts with a choice between "Author from scratch", "Use a blueprint", and "Container image". The "Author from scratch" option is selected. The next section, "Basic information", includes fields for "Function name" (sdc-stock-tracker-api), "Runtime" (Node.js 24.x), and "Durable execution - new". The runtime dropdown has a note: "Choose the language to use to write your function. Note that the console code editor supports only Node.js, Python, and Ruby."

Figure 6 Lambda Creation



The screenshot shows the AWS Lambda console interface. At the top, there are three tabs: 'Software Development & Const.' (selected), 'Launch AWS Academy Learner', and 'sdc-stock-tracker-api | Function'. The main title is 'sdc-stock-tracker-api'. Below the title, there's a 'Function overview' section with tabs for 'Diagram' (selected) and 'Template'. The 'Diagram' tab shows a single function icon labeled 'sdc-stock-tracker-api' with '(0)' layers. There are buttons for '+ Add trigger' and '+ Add destination'. To the right, there's a sidebar with sections for 'Description', 'Last modified' (2 minutes ago), 'Function ARN' (arn:aws:lambda:us-east-1:992382621536:function:sdc-stock-tracker-api), and 'Function URL'.

Figure 7 Lambda creation successful



The screenshot shows the AWS Lambda code editor for the function 'sdc-stock-tracker-api'. The 'Code' tab is selected. The code editor displays the 'index.mjs' file with the following content:

```

1 export const handler = async (event) => {
2     // TODO implement
3     const response = {
4         statusCode: 200,
5         body: JSON.stringify('Hello from Lambda!'),
6     };
7     return response;
8 }
9

```

The sidebar on the left shows the project structure under 'EXPLORER': 'SDC-STOCK-TRACKER-API' containing 'index.mjs'. Below it, under 'DEPLOY', is a 'Current' deployment. A blue button labeled 'Deploy (Ctrl+Shift+U)' is visible.

Upload a .zip file dialog box is overlaid on the interface, prompting the user to upload a new .zip file package. It includes a note about overwriting existing code, an 'Upload' button, and an optional checkbox for encryption with an AWS KMS customer managed key. The 'Save' button is highlighted in orange.

Figure 8 Uploading backed files

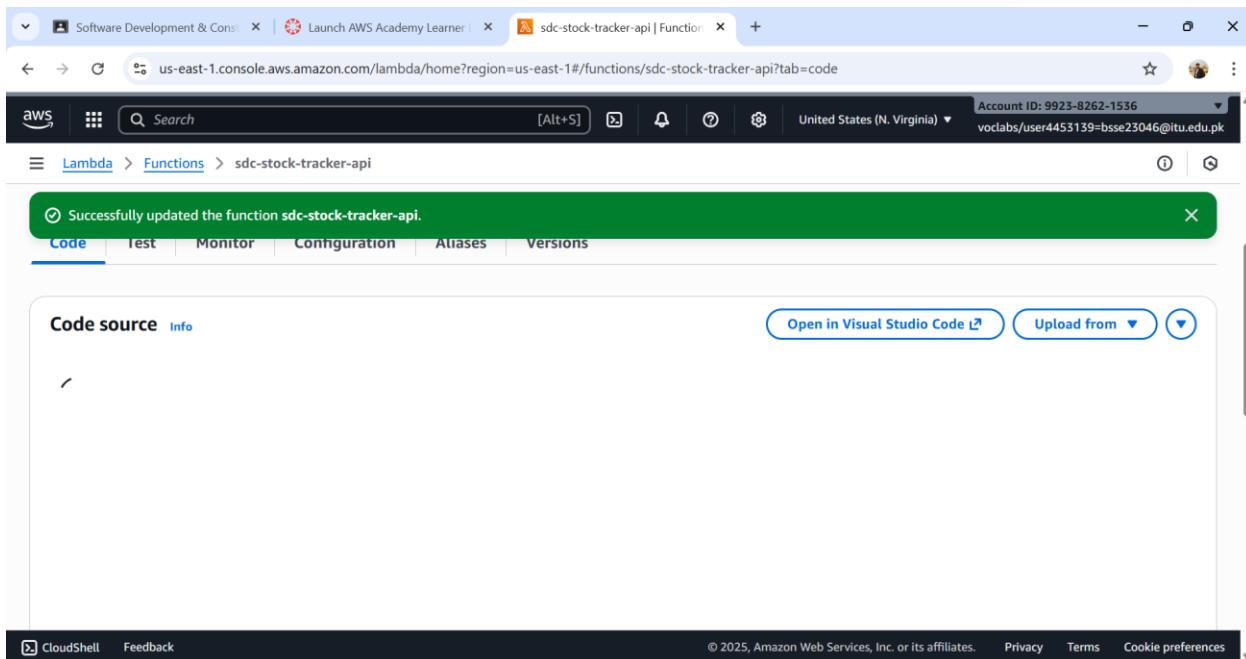
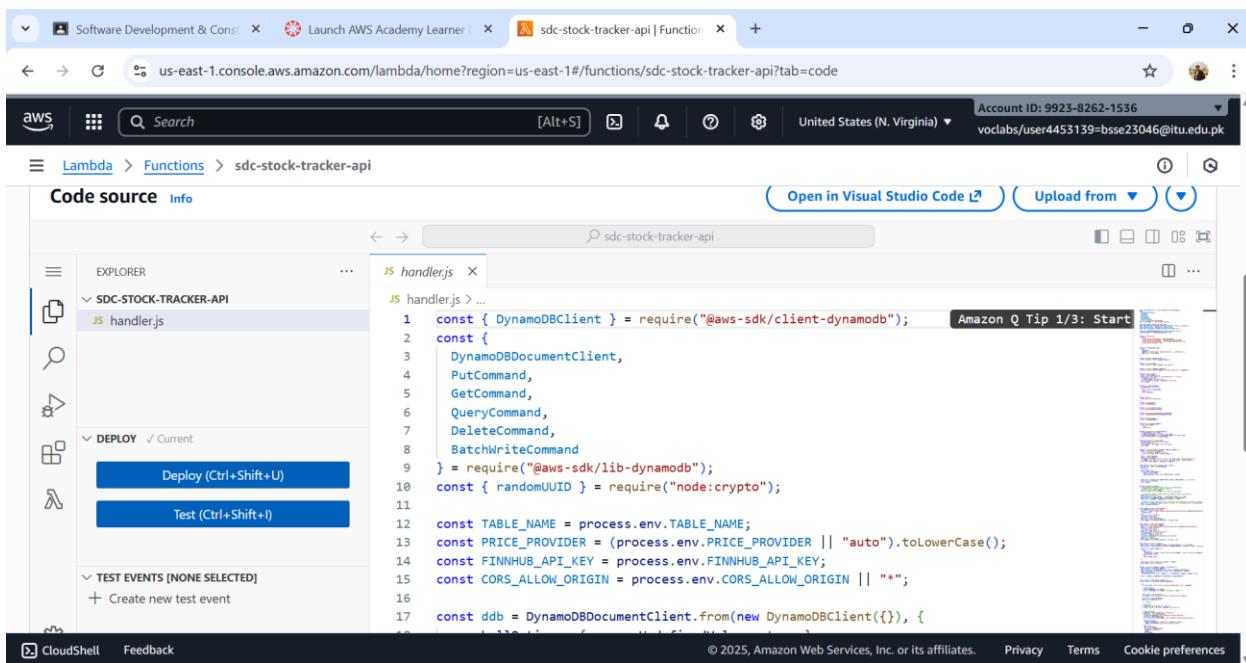
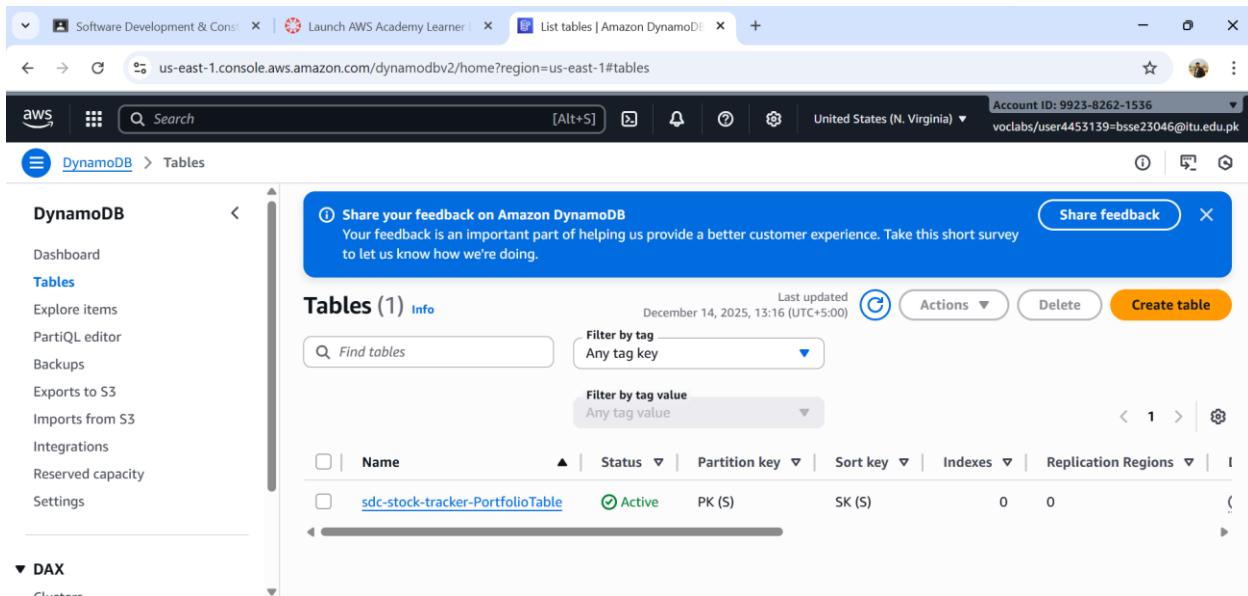


Figure 9 Handler.js uploaded

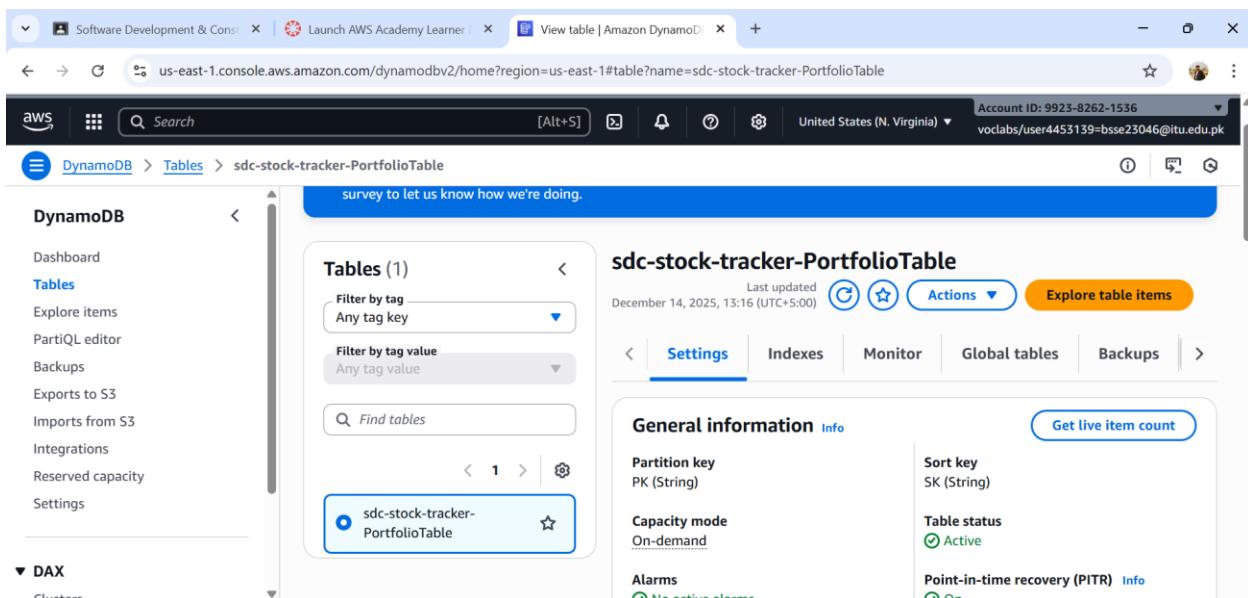




The screenshot shows the AWS DynamoDB console. On the left, a sidebar menu is open under the 'Tables' section, listing various options like Dashboard, Explore items, PartiQL editor, Backups, Exports to S3, Imports from S3, Integrations, Reserved capacity, and Settings. Below this, a 'DAX' section is partially visible. The main content area displays a table titled 'Tables (1)'. The table has one item: 'sdc-stock-tracker-PortfolioTable'. The table details are as follows:

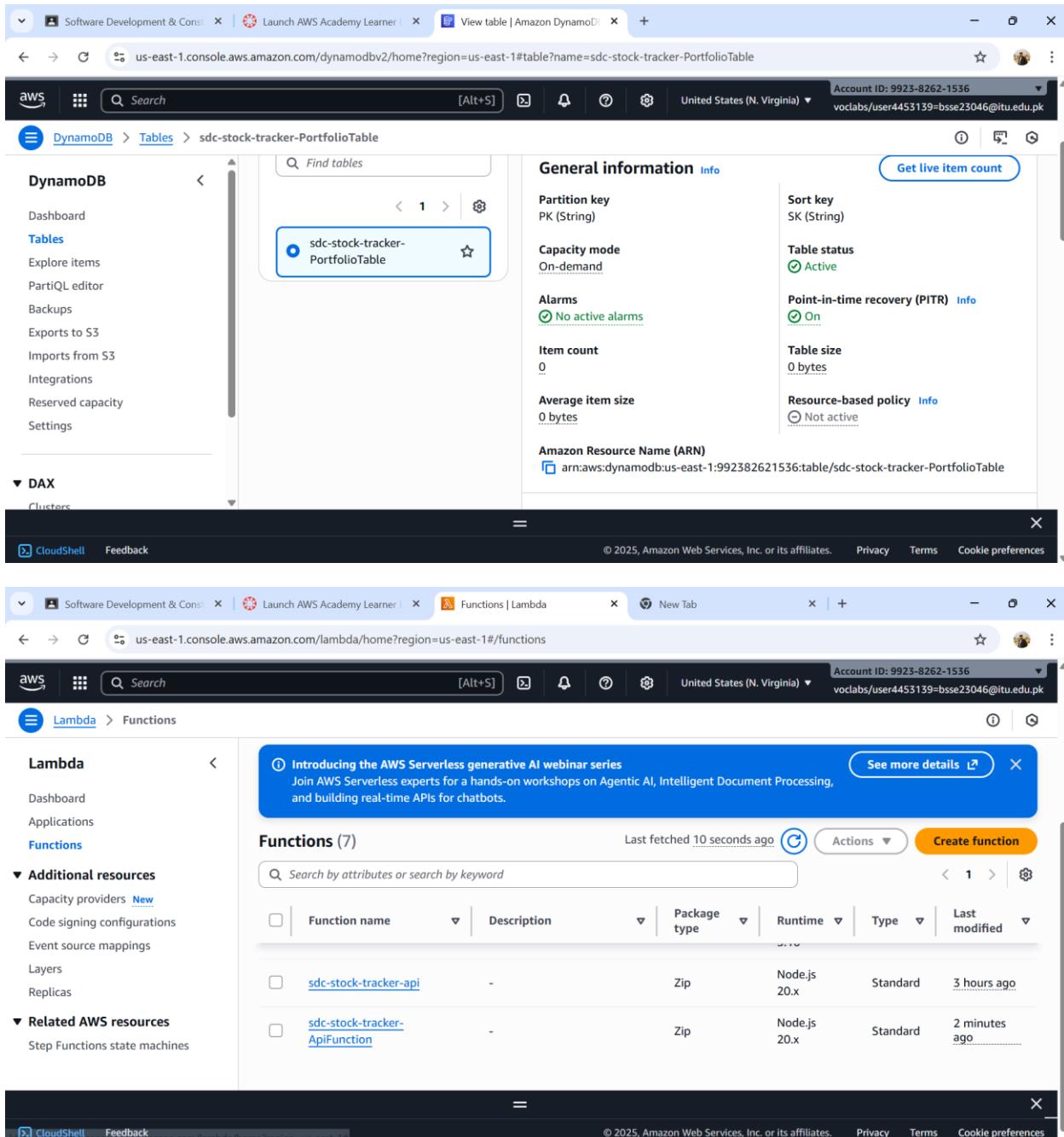
Name	Status	Partition key	Sort key	Indexes	Replication Regions
sdc-stock-tracker-PortfolioTable	Active	PK (S)	SK (S)	0	0

Figure 10 Dynamo.db Created



This screenshot shows the 'sdc-stock-tracker-PortfolioTable' settings page within the AWS DynamoDB console. The left sidebar remains the same as in Figure 10. The main area is titled 'sdc-stock-tracker-PortfolioTable' and includes tabs for 'Settings' (which is selected), 'Indexes', 'Monitor', 'Global tables', and 'Backups'. The 'General information' section contains the following details:

- Partition key:** PK (String)
- Sort key:** SK (String)
- Capacity mode:** On-demand
- Table status:** Active
- Alarms:** No active alarms
- Point-in-time recovery (PITR):** On



The screenshot shows the AWS Lambda Functions page. The left sidebar includes sections for Lambda (Dashboard, Applications, Functions), Additional resources (Capacity providers, Code signing configurations, Event source mappings, Layers, Replicas), and Related AWS resources (Step Functions state machines). The main content area displays a table of functions:

Function name	Description	Package type	Runtime	Type	Last modified
sdc-stock-tracker-api	-	Zip	Node.js 20.x	Standard	3 hours ago
sdc-stock-tracker-ApiFunction	-	Zip	Node.js 20.x	Standard	2 minutes ago

Figure 11 Aws Lambda Function Created

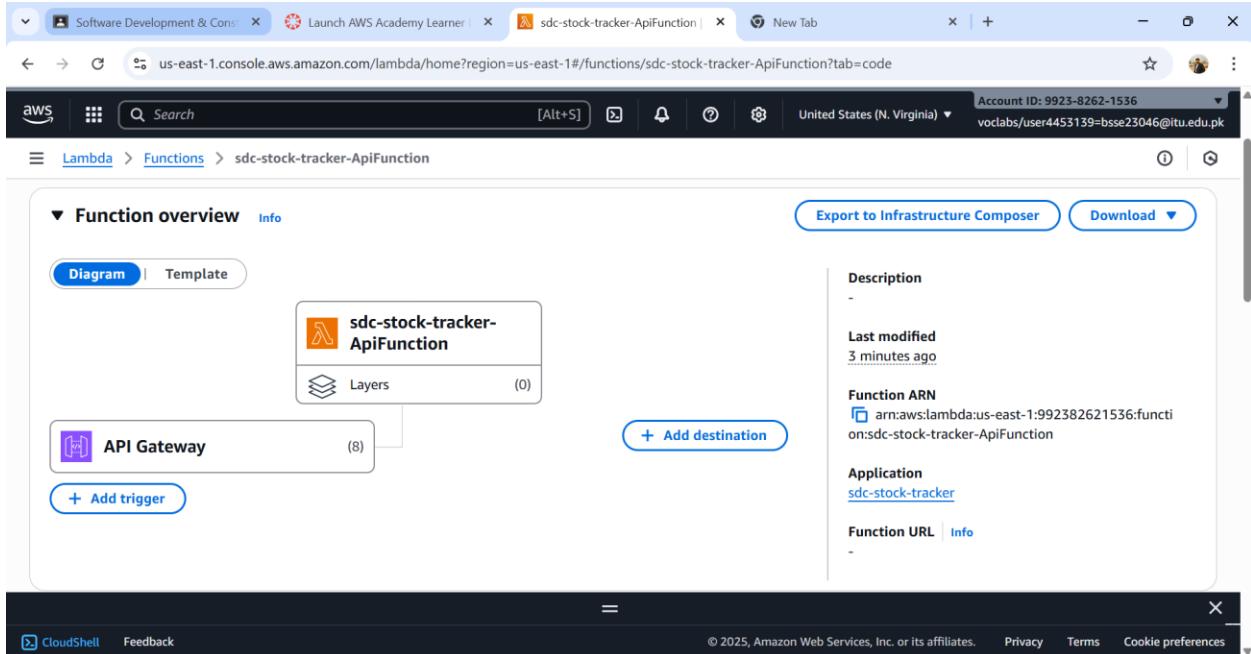


Figure 12 Api Gateways Attached

IAM Role Attached with lambda:

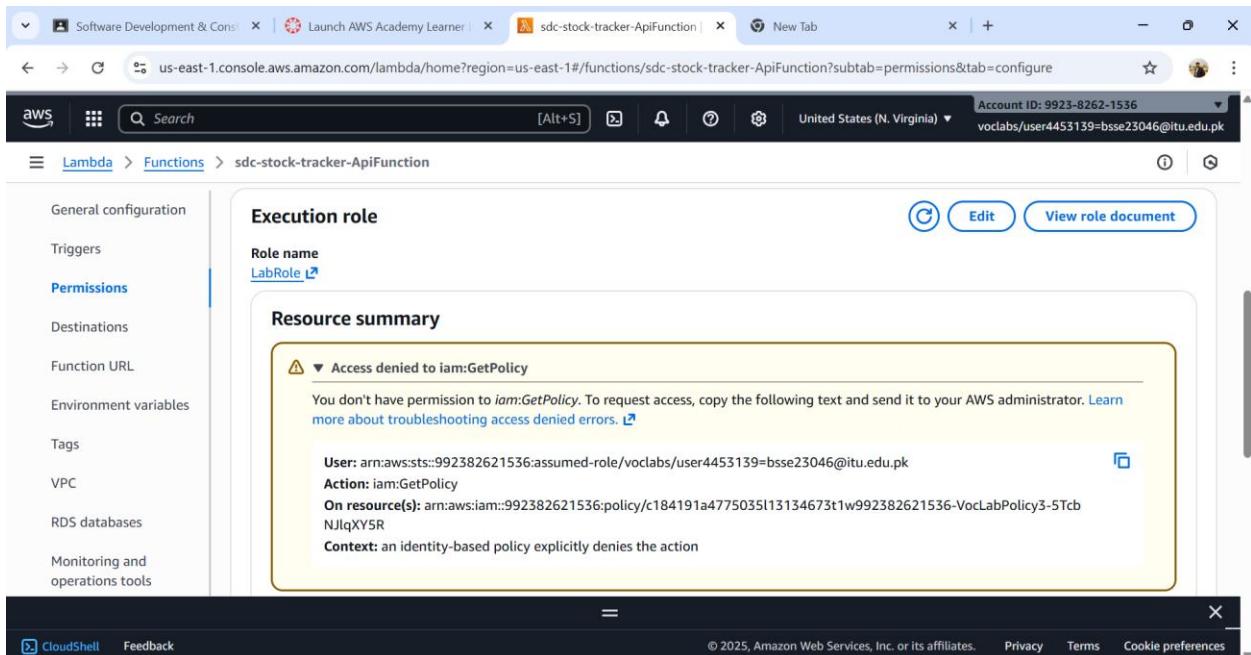
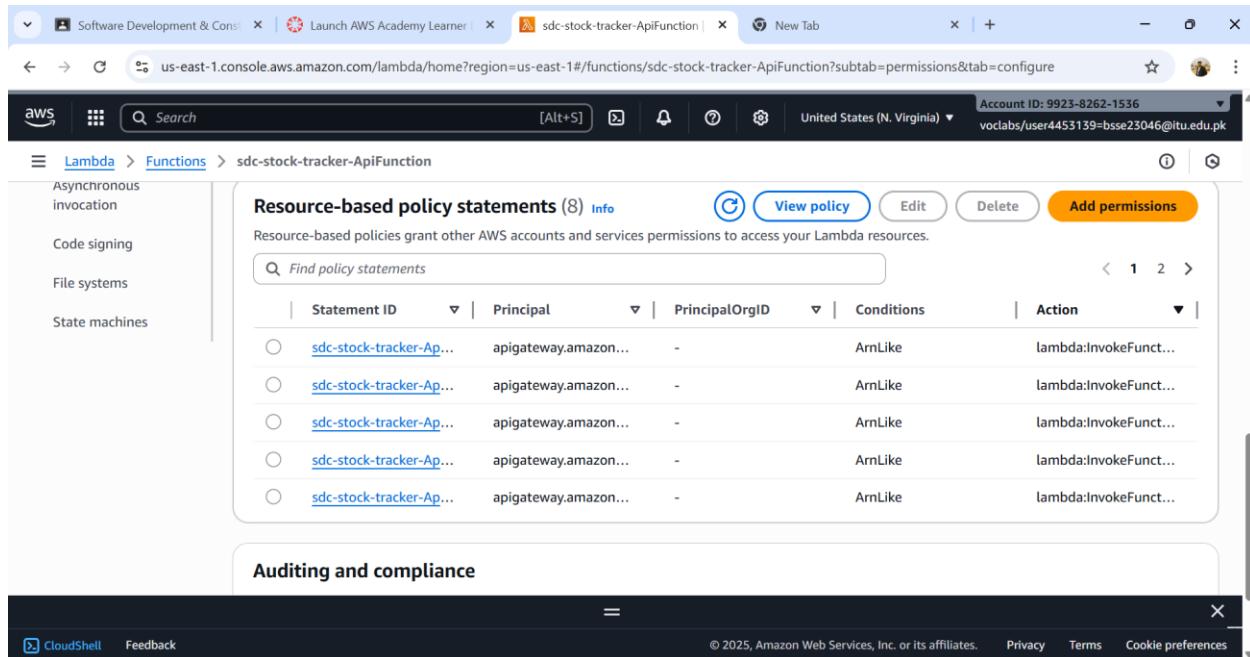
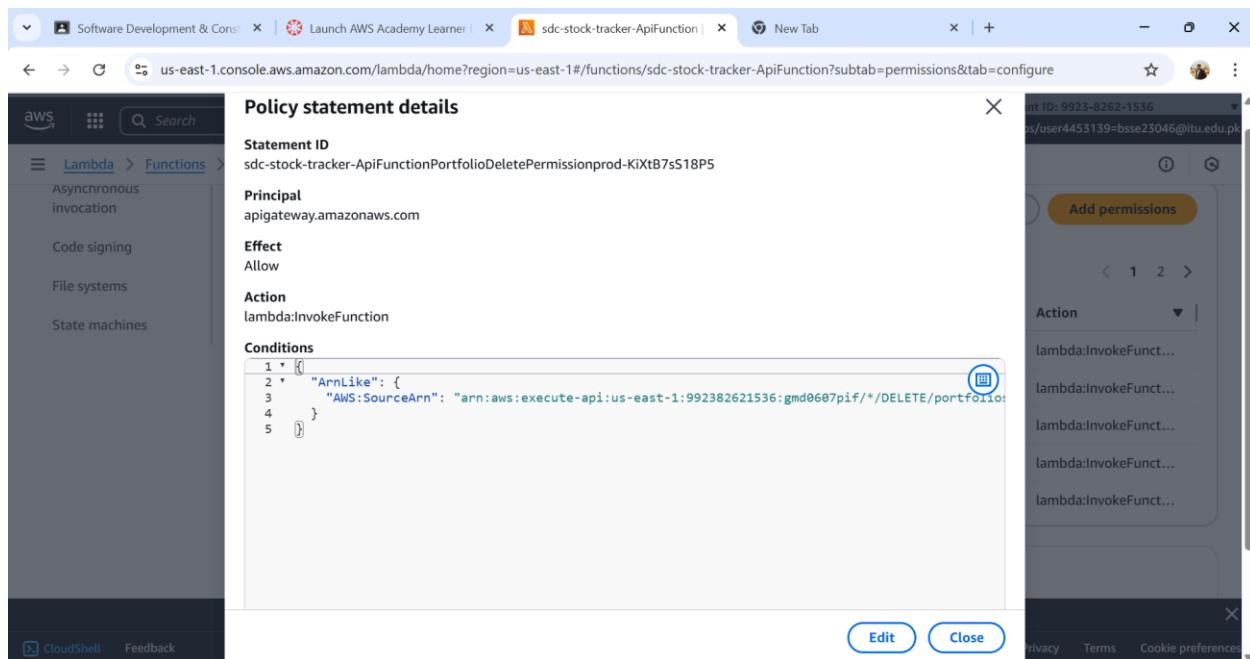


Figure 13 Lab Role Attached



The screenshot shows the AWS Lambda Functions page for the 'sdc-stock-tracker-ApiFunction'. On the left sidebar, under the 'Permissions' section, there are links for 'Asynchronous invocation', 'Code signing', 'File systems', and 'State machines'. The main content area is titled 'Resource-based policy statements (8)'. It includes a search bar 'Find policy statements' and a table with columns: Statement ID, Principal, PrincipalOrgID, Conditions, and Action. The table lists eight policy statements, each with 'arn:aws:execute-api:us-east-1:992382621536:gmd0607pif/*/DELETE/portfolioDeletePermissionprod-K1xtB7s18P5' as the Principal and 'lambda:InvokeFunction' as the Action. The Conditions column shows 'ArnLike' conditions for the AWS:SourceArn field.

Figure 14 Resource-based policy statements



This screenshot shows the 'Policy statement details' for a specific policy statement. The statement ID is 'sdc-stock-tracker-ApiFunctionPortfolioDeletePermissionprod-K1xtB7s18P5'. The principal is 'apigateway.amazonaws.com', the effect is 'Allow', and the action is 'lambda:InvokeFunction'. The conditions are defined as follows:

```

1 "ArnLike": {
2   "AWS:SourceArn": "arn:aws:execute-api:us-east-1:992382621536:gmd0607pif/*/DELETE/portfolioDeletePermissionprod-K1xtB7s18P5"
3 }
4
5
    
```

At the bottom right of the modal window, there are 'Edit' and 'Close' buttons.

Figure 15 Delete Policy Statement

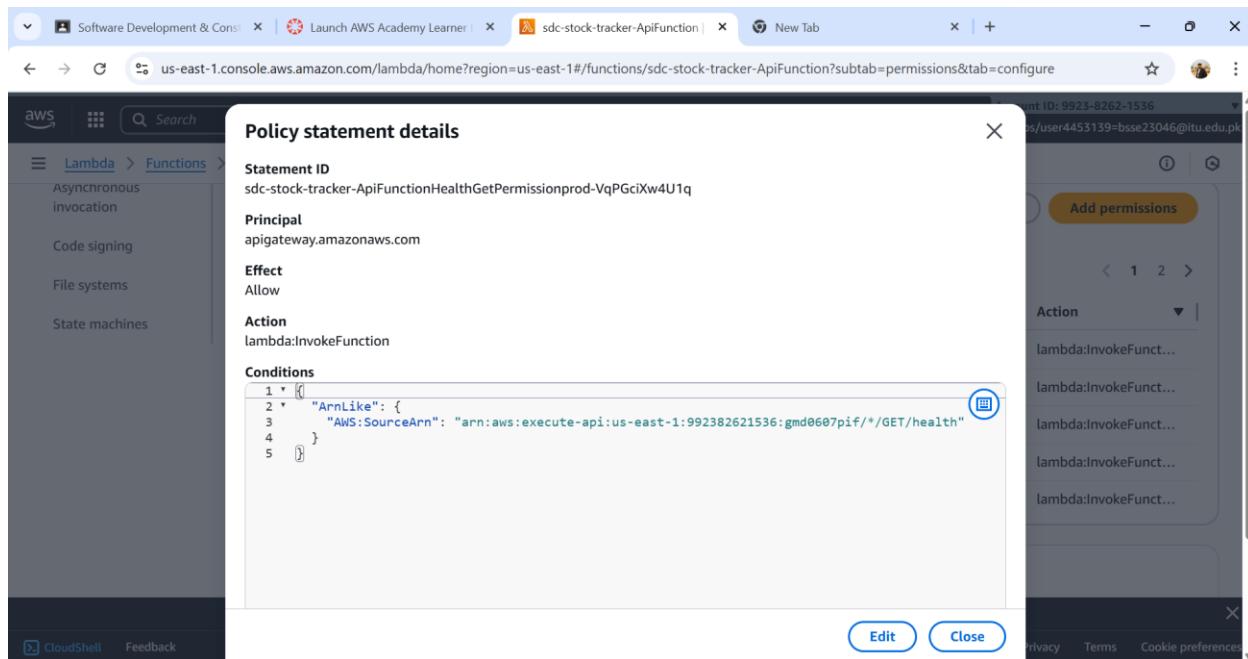


Figure 16 Get Health Policy Statement

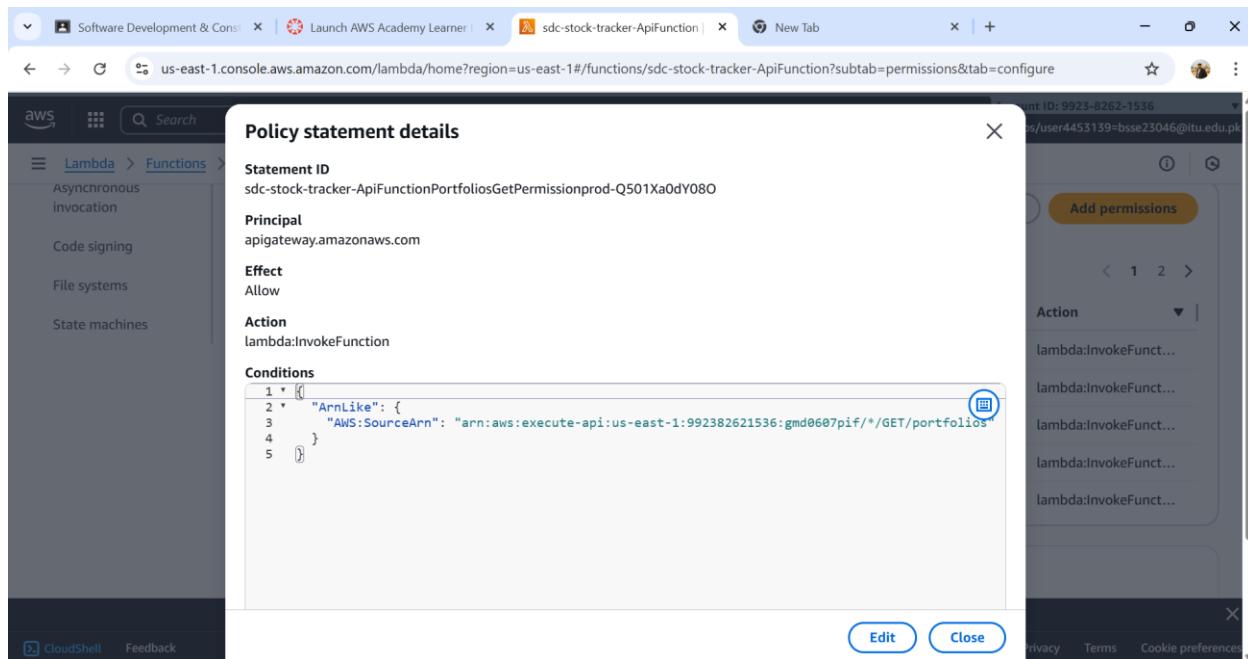


Figure 17 Get Portfolio Policy Statement

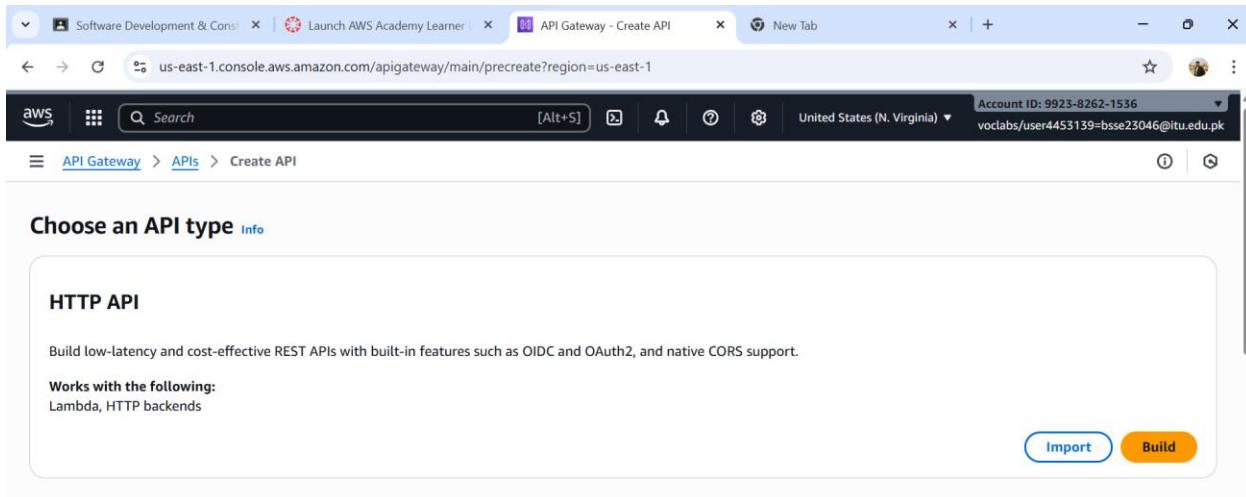


Figure 18 Api Created

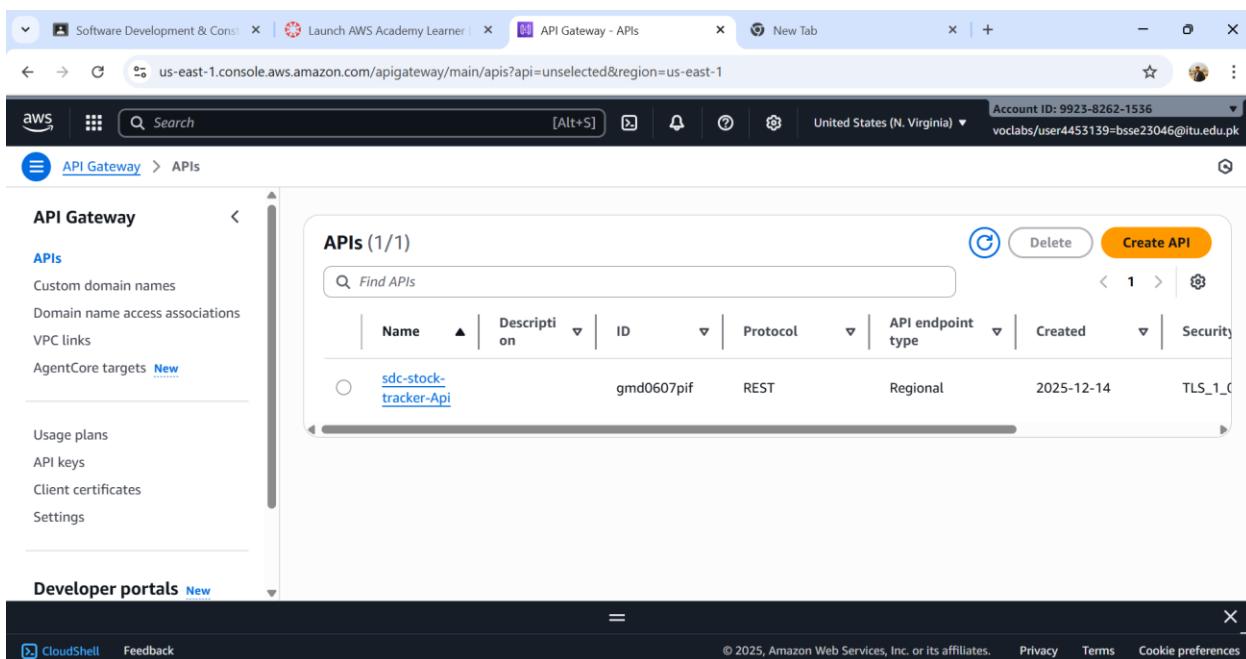
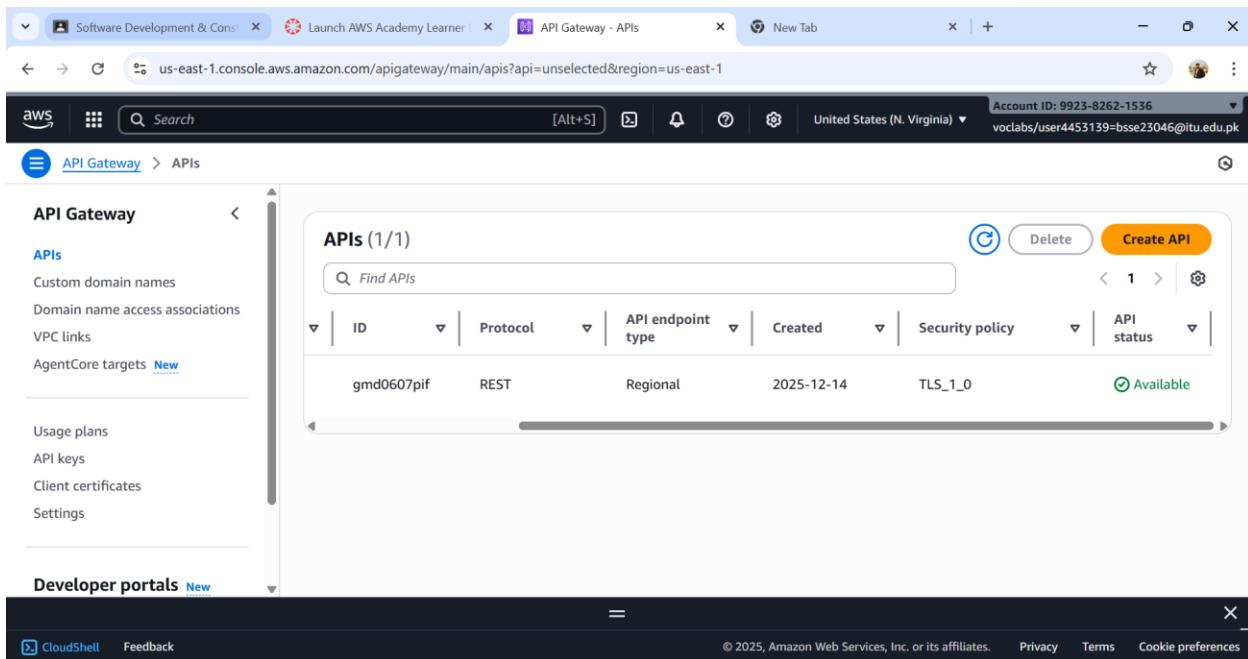


Figure 19 Sdc-stock-tracker-Api

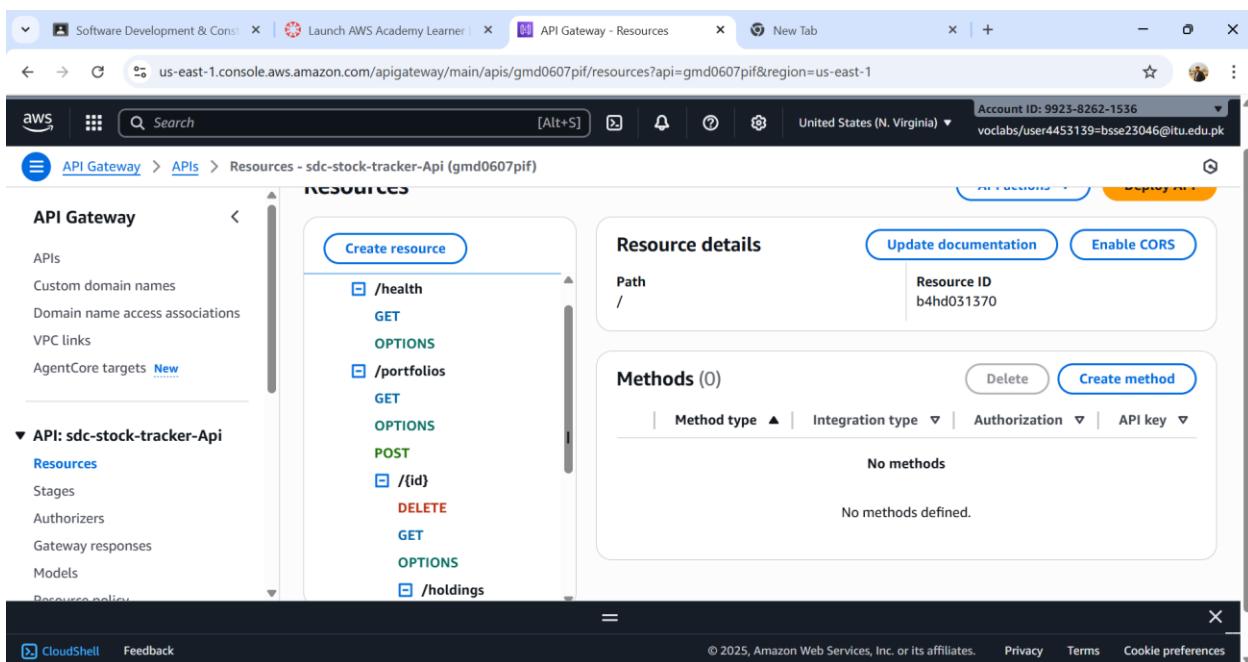


The screenshot shows the AWS API Gateway APIs page. On the left, a sidebar menu includes 'APIs', 'Custom domain names', 'Domain name access associations', 'VPC links', 'AgentCore targets', 'Usage plans', 'API keys', 'Client certificates', and 'Settings'. The main content area displays a table titled 'APIs (1/1)' with one entry:

ID	Protocol	API endpoint type	Created	Security policy	API status
gmd0607pif	REST	Regional	2025-12-14	TLS_1_0	Available

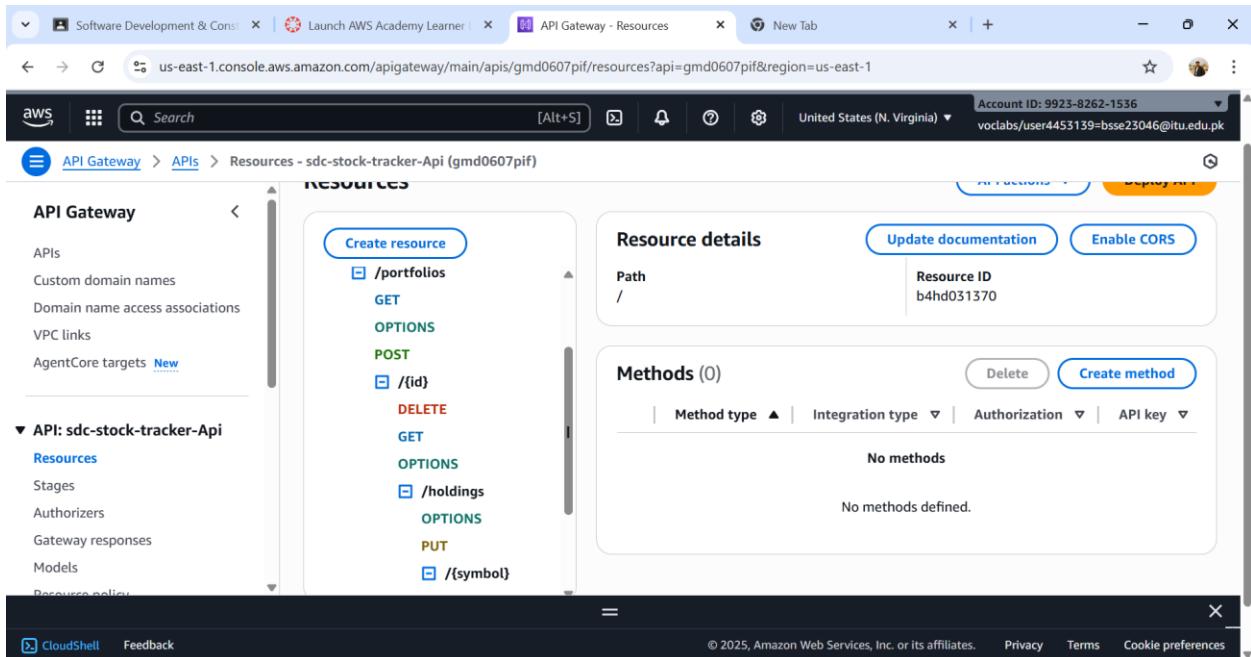
At the bottom of the page, there are links for 'CloudShell', 'Feedback', and copyright information: '© 2025, Amazon Web Services, Inc. or its affiliates.' and 'Privacy Terms Cookie preferences'.

Figure 20 Status



The screenshot shows the AWS API Gateway Resources page for the 'sdc-stock-tracker-Api' (gmd0607pif). The left sidebar shows 'APIs', 'Custom domain names', 'Domain name access associations', 'VPC links', 'AgentCore targets', and 'API: sdc-stock-tracker-Api' with 'Resources' selected. Under 'Resources', options include 'Stages', 'Authorizers', 'Gateway responses', 'Models', and 'Resource policy'. The main content area shows a 'Create resource' button and a 'Resource details' section for the root resource '/'. The 'Resource details' section includes a 'Path' field with a value of '/', a 'Resource ID' field with 'b4hd031370', and buttons for 'Update documentation' and 'Enable CORS'. Below this is a 'Methods (0)' section with a table header for 'Method type', 'Integration type', 'Authorization', and 'API key'. A note states 'No methods defined.' at the bottom.

Figure 21 Resources In API Created



The screenshot shows the AWS API Gateway Resources page for the 'sdc-stock-tracker-Api' (gmd0607pif). On the left sidebar, under 'APIs', the 'sdc-stock-tracker-Api' is selected. In the main content area, a single resource named '/portfolios' is listed. This resource has several methods defined: GET, OPTIONS, POST, DELETE, GET, OPTIONS, PUT, and OPTIONS. The 'Methods (0)' section indicates that no methods have been defined for this resource.

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 Portfolio", "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 $
```

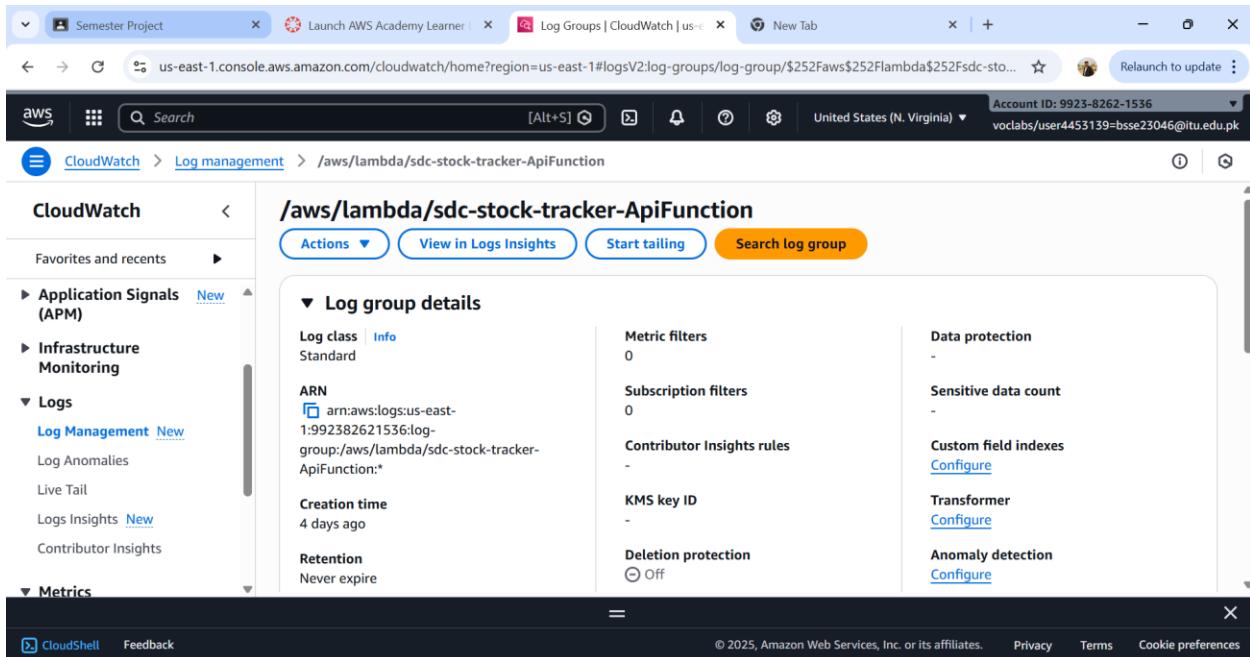
Figure 23 Get A specific Portfolio

```
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", "symbol": "AAPL", "quantity": 10, "avgCost": 150, "createdAt": "2025-12-14T10:53:35.995Z", "updatedAt": "2025-12-14T10:53:35.995Z"}backend $
```

Figure 24 Add holdings

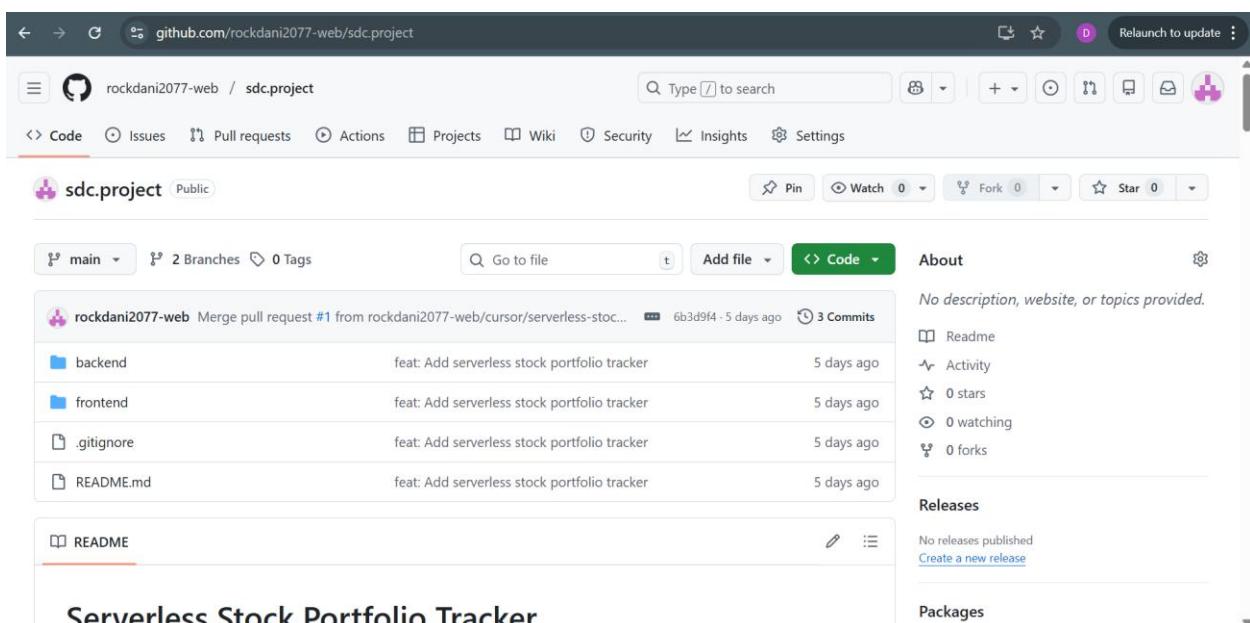
```
{"error": "Query param 'symbols' is required (comma-separated)"}backend $ curl -X GET "https://gmd0607pif.execute-api.us-east-1.amazonaws.com/prod/prices/symbols=AAPL" \
> -H "x-user-id: 123"
{"prices": {"AAPL": null}, "ts": "2025-12-14T10:54:43.411Z"}backend $
```

Figure 25 Check stock price



The screenshot shows the AWS CloudWatch Log Management interface. The left sidebar has sections for CloudWatch, Application Signals (APM), Infrastructure Monitoring, Logs (Log Management, Log Anomalies, Live Tail, Logs Insights, Contributor Insights), Metrics, and CloudShell. The main area displays log group details for the path /aws/lambda/sdc-stock-tracker-ApiFunction. The ARN is listed as arn:aws:logs:us-east-1:992382621536:log-group:/aws/lambda/sdc-stock-tracker-ApiFunction:. The log group was created 4 days ago and has a retention policy of 'Never expire'. Metric filters, Subscription filters, Contributor Insights rules, KMS key ID, and Deletion protection (set to Off) are also listed. Data protection features like Sensitive data count, Custom field indexes, Transformer, and Anomaly detection are shown with their respective configuration links.

Figure 26 Cloud Watch Log Maintenance



The screenshot shows a GitHub repository page for 'sdc.project' under the user 'rockdani2077-web'. The repository is public. The main branch is 'main' with 2 branches and 0 tags. There are 3 commits from 'rockdani2077-web' merged into the main branch. The commits are: 'feat: Add serverless stock portfolio tracker' for both 'backend' and 'frontend' branches, and 'feat: Add serverless stock portfolio tracker' for '.gitignore' and 'README.md'. The repository has 0 stars, 0 forks, and 0 releases. The README file contains the title 'Serverless Stock Portfolio Tracker'.

Figure 27 Docker +kubernetes+Jenkins

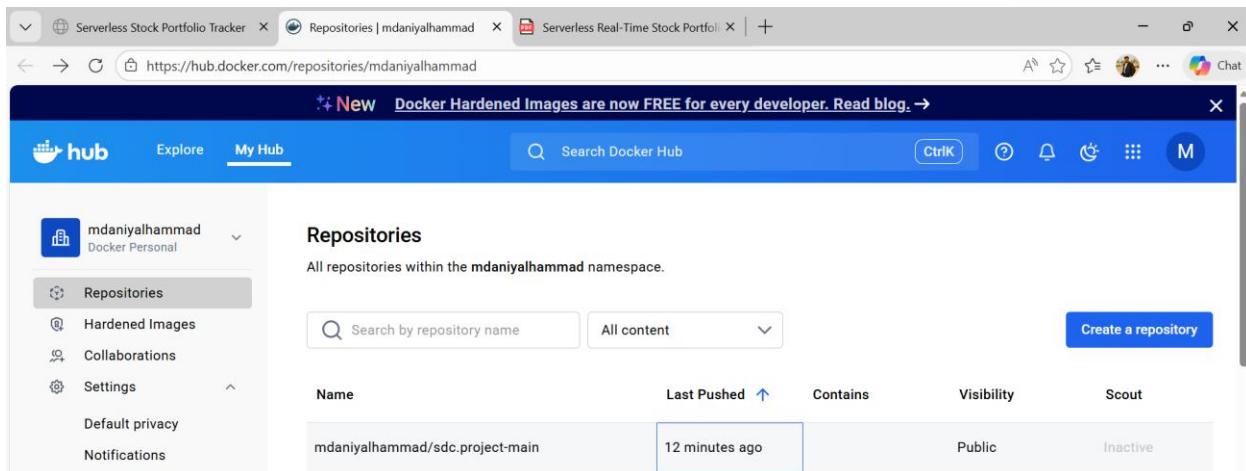


Figure 28 Docker Repository

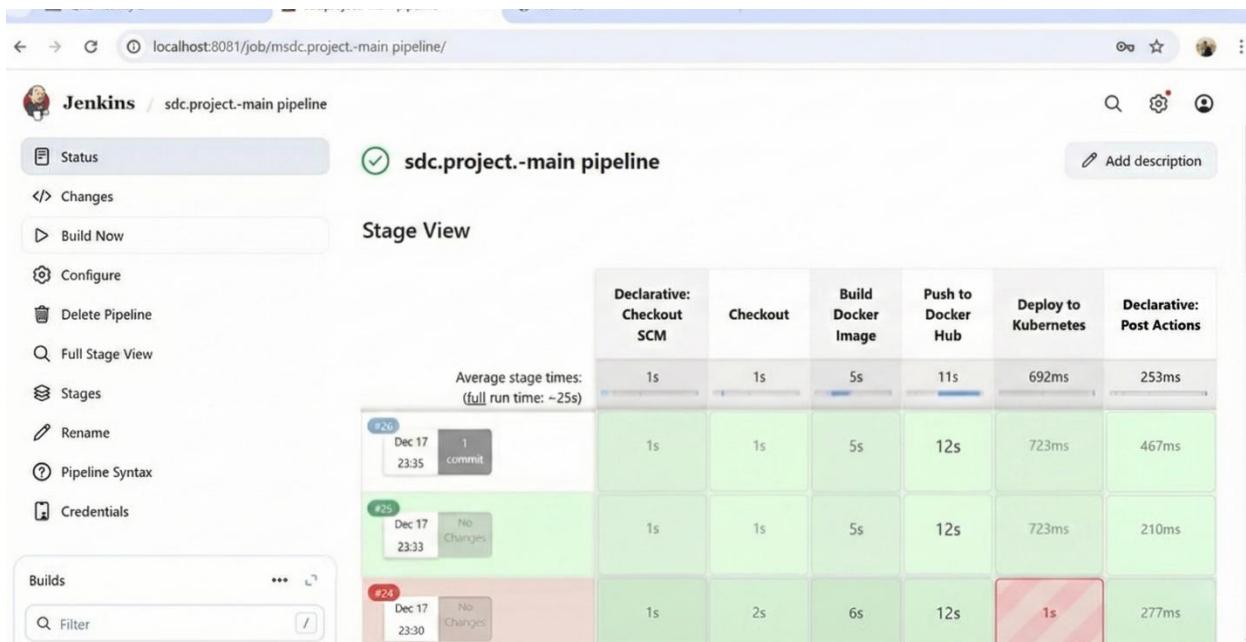


Figure 29 Jenkins Automation

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

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

Unrealized Profit / Loss

$\text{Unrealized P/L} = (\text{Current Price} - \text{Buy Price}) \times \text{Quantity}$

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/>