**Real Time Sports Dashboard Using   
AWS AppSync and DynamoDB**

*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

**Sai Vyseshika V**

**(2210030504)**

*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 **“Real Time Sports Dashboard Using   
AWS AppSync and DynamoDB”** which is a experimental &/ theoretical &/ Simulation&/ hardware work carried out by Sai Vyseshika V (2210030504), 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**

Page No.

1. Introduction 1

2. AWS Services Used as part of the project 2

3. Steps involved in solving project problem statement 3

4. Stepwise Screenshots with brief description 6

5. Learning Outcomes 15

6. Conclusion 16

7. Future Scope 16

8. References 17

**1.INTRODUCTION**

*Mini Project Title*

In today's fast-paced digital world, sports enthusiasts demand real-time updates on live matches, scores, and player performances. Traditional sports websites often rely on manual refreshes to display updated scores, causing delays in accessing real-time information. With the rapid advancement of cloud technologies, it has become possible to build a fully automated sports dashboard that delivers instant updates without requiring constant page refreshes.

This project aims to develop a real-time sports dashboard that fetches live cricket scores from an external API and displays them dynamically using AWS AppSync, DynamoDB, IAM roles, and JavaScript. The data is stored, processed, and updated automatically, allowing users to stay informed with the latest match progress through an interactive interface powered by Chart.js.[3]

The architecture follows a serverless model, meaning that the system efficiently handles real-time data flow with minimal maintenance. AWS AppSync provides GraphQL-based real-time data synchronization, DynamoDB acts as a scalable NoSQL database, and GraphQL subscriptions ensure that frontend clients receive automatic updates. The frontend is built with JavaScript and Chart.js, offering a user-friendly way to visualize live match scores in the form of dynamic pie charts.[1]

By leveraging AWS cloud services, this project demonstrates the power of serverless computing in modern real-time applications. The result is a highly scalable, cost-efficient, and real-time sports dashboard that can be extended to support multiple sports and additional features such as player statistics, match analysis, and push notifications.

**2.AWS Services Used as part of the project**

*Mini Project Title*

* **AWS AppSync**   
    
  Provides a GraphQL API that facilitates real-time data synchronization between  
  the database and frontend. It eliminates the need for backend servers and allows smooth integration with DynamoDB, making it an ideal choice for handling live sports data. Enables real-time updates through GraphQL subscriptions, reducing latency and improving user experience.[1]
* **AmazonDynamoDB**   
    
  A fast and scalable NoSQL database that stores live match data. DynamoDB’s   
  ability to handle rapid data updates ensures that match scores are always current, providing seamless real-time functionality. [2] Offers high-speed read/write operations with automatic scaling, ensuring efficient data storage and retrieval.
* **IAM Roles and Policies**   
    
  Used for granting necessary permissions to AppSync and DynamoDB without requiring an AWS Lambda function. IAM roles ensure secure access control, defining which AWS services can interact with each other.  
  Enhances security by restricting permissions and reducing potential vulnerabilities.
* **Amazon API Gateway**   
    
  Serves as a bridge between the external cricket API and AppSync. It helps manage incoming API requests, transforming them into a format suitable for GraphQL queries while ensuring secure access to live sports data. Acts as an intermediary, ensuring seamless API data retrieval and protecting backend services from excessive traffic.

.

**3.Steps Involved in solving project problem statement**

1.**Understanding Requirements**:   
  
 The first step was identifying the need for a real-time sports dashboard and defining   
 key functionalities such as data fetching, storage, and visualization.

### 2**. Setting Up AWS AppSync:**

AWS AppSync is crucial for handling real-time data in your dashboard. The setup   
 process includes:

* Creating an AppSync API: Using the AWS Management Console to create the GraphQL API for data handling. AppSync simplifies real-time data synchroniz  
  -ation, offering support for queries, mutations, and subscriptions.
* Connecting AppSync to DynamoDB: AppSync integrates seamlessly with DynamoDB, allowing automatic scaling. Setting this up involves choosing DynamoDB as a data source, configuring it as a back-end for match data storage.[2]
* Ensuring Real-Time Updates: AppSync helps maintain real-time updates to the frontend by utilizing GraphQL subscriptions. Every time data is modified in DynamoDB, AppSync pushes updates to the client, ensuring live data streams without constant polling.

3. **Creating the GraphQL Schema:**

The GraphQL schema serves as the backbone of your API. It defines the structure of   
 data and specifies how it can be queried or mutated. Key steps here include:

* Defining Data Types: Specify the data types for your objects, such as Match, Team,Score and PlayerStats.
* Defining Queries and Mutations: Queries retrieve data (e.g., fetching current match scores), while mutations modify data (e.g., updating a team’s score). The schema must support both.
* Defining Subscriptions: To allow real-time updates, you'll define a subscription that listens for changes (e.g., a new score update) and pushes it to the frontend.

**4. Configuring Data Sources and Resolvers:**

Resolvers are the critical component linking GraphQL operations to DynamoDB

* Setting up DynamoDB Table: Create a table in DynamoDB with appropriate partition and sort keys, such as MatchID for partition key and timestamp for sort key.
* Configuring Resolvers: Each operation in the schema (query, mutation, or subscription) is linked to a resolver. For example, a getMatchData query would resolve to fetching data from the DynamoDB table. Mutations would update the DynamoDB table based on incoming data.
* Optimizing Data Fetching: To ensure efficiency, consider implementing pagination for large data sets (e.g., for historical match data) and batch operations for bulk updates.

**5. Setting Up IAM Roles:**

* IAM Role Creation: Define IAM roles that grant AppSync permission to read/write data from/to DynamoDB securely. For example, you would create a policy like AppSyncDynamoDBAccessPolicy that dynamodb:Query, dynamodb:PutItem, and other relevant actions.
* Fine-Grained Access Control: You can set more granular permissions based on the operation (e.g., different roles for admins vs. regular users) or based on the data’s nature (e.g., limiting access to scores of specific matches).
* Security Best Practices: Ensure the least privilege principle is followed by restricting permissions to only necessary actions.

**6. Fetching Live Cricket Data:**

This step requires integrating external sources to bring in live data:

* API Integration: Integrating an external API, such as a cricket match data provider like ESPN or CricAPI, allows the system to fetch live match data.
* Data Parsing and Transformation: The fetched data needs to be parsed and transformed into a format that fits the GraphQL schema. For example, converting match score data into a format compatible with the Match and Teams types in the GraphQL schema.
* Error Handling and Fallbacks: Consider implementing error handling for failed data fetches, such as retries or fallback mechanisms in case the API is down or fails to return data.

**7. Developing the Frontend:**

Building a user-friendly frontend is crucial for displaying real-time data effectively:

* Frontend Framework: Since you're using JavaScript, a simple, lightweight frontend using vanilla JavaScript or frameworks like Vue.js (without React) can dynamically update based on GraphQL subscriptions.
* Visualizing Data: Use Chart.js to create dynamic and interactive visualizations, such as pie charts for team statistics, bar charts for score comparisons, and line charts for historical performance.
* Real-Time Data Binding: The frontend should automatically update whenever new data is received, without requiring a page refresh. This can be achieved using GraphQL subscriptions, where the frontend listens for changes and updates the UI in real-time.

**8. Implementing GraphQL Subscriptions:**

Subscriptions allow the frontend to automatically receive updates when the data changes:

* Setting Up Subscriptions: In the GraphQL schema, define a subscription for real-time updates on match data, such as a subscription that notifies the frontend of score changes.
* Handling Data in the Frontend: In the frontend, use Apollo Client (or a similar library) to manage the connection to the AppSync API. This will handle incoming data from subscriptions and update the UI accordingly.
* Testing Subscriptions: Ensure that subscriptions work as expected by testing with various match updates and verifying that the frontend receives real-time updates without delay.

**9.Deploying the Application**:   
  
Launched the project on a cloud-based hosting platform, ensuring availability and reliability

**4.Stepwise Screenshots with brief description**

**Step 1:**

This step begins by creating a new GraphQL API using the AWS AppSync console. The API is named SportsDashboard to reflect its purpose in handling live sports data. Once the API is created, the GraphQL schema is defined to establish the structure of the data. This schema includes object types, queries, mutations, and subscriptions.

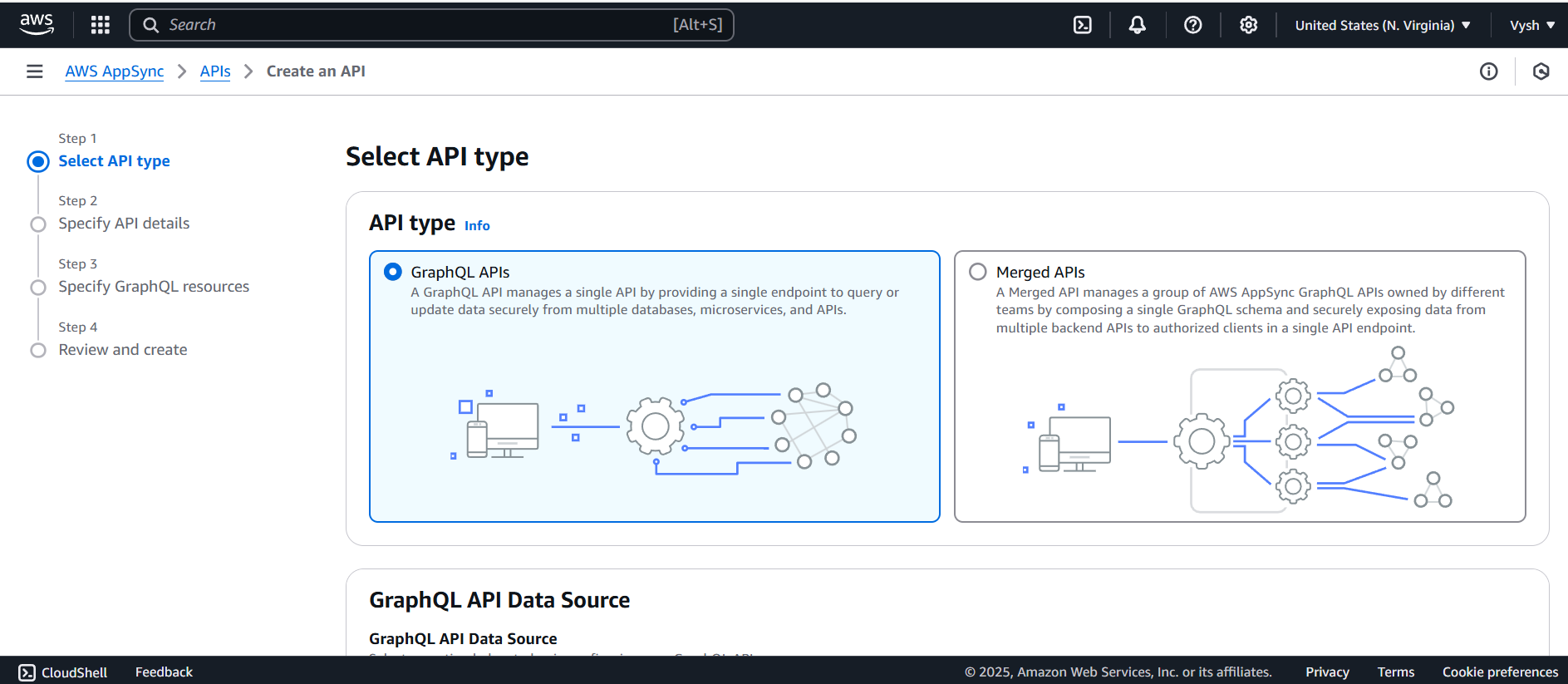
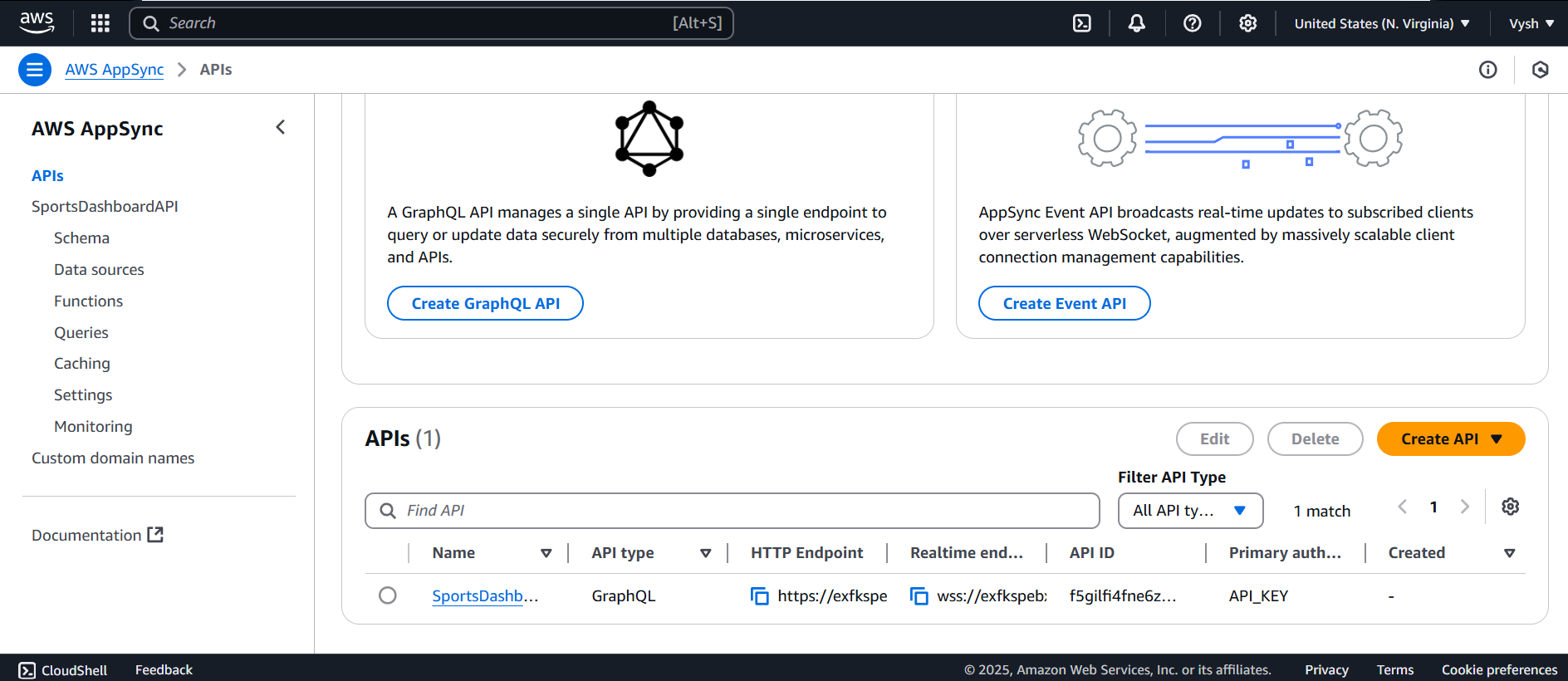
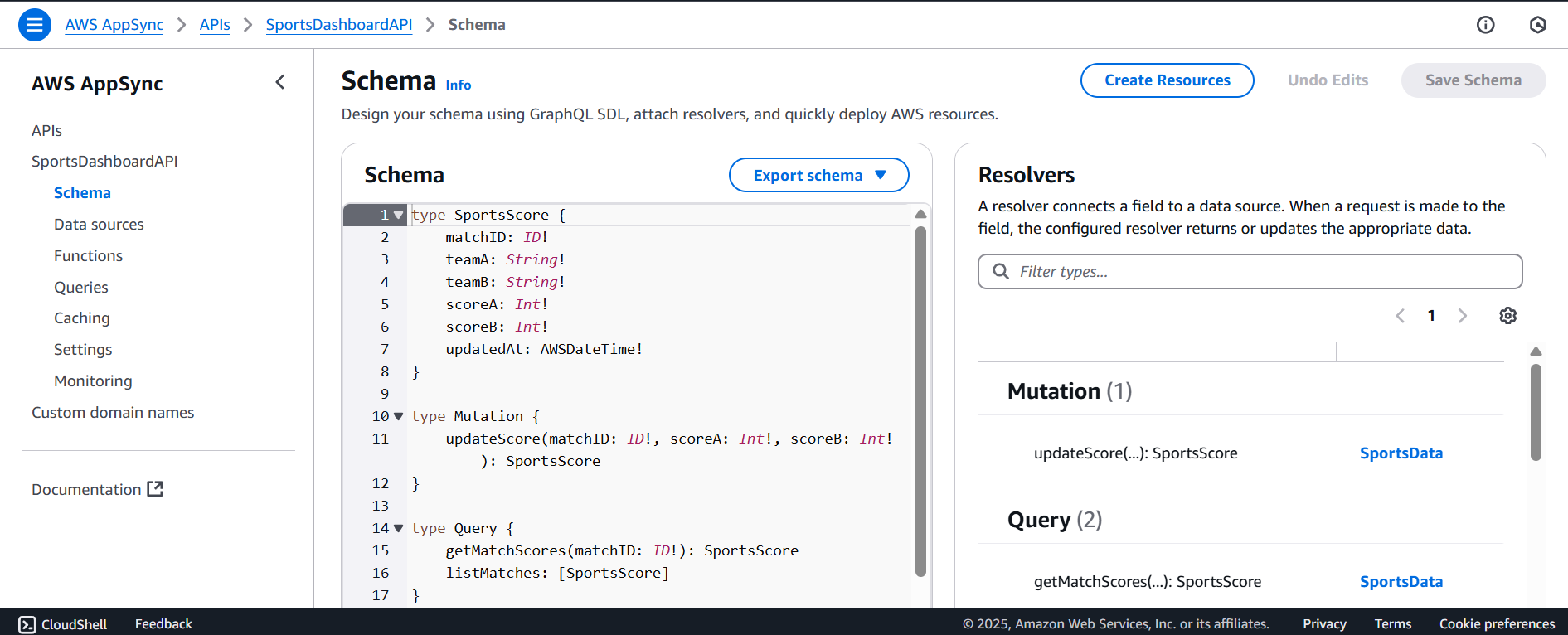
****

Fig 4.1: Creating a new GraphQL API in AWS AppSync

**** Fig 4.2: Created a GraphQL API named SportsDashboard

  
 Fig 4.3: Defining Schema

**Step 2:**

A new table is created in DynamoDB to store live match data, and it's named SportsScores. The table includes necessary attributes such as MatchID for partitioning and timestamp for sorting. Next, a data source is created in AppSync and connected to this table, allowing the API to interact with DynamoDB. This data source is named SportsData for clarity. Resolvers are then attached to map the GraphQL operations (queries and mutations) to the corresponding actions in DynamoDB. Finally, request mapping templates are configured to control how GraphQL requests are translated into DynamoDB commands and how responses are handled

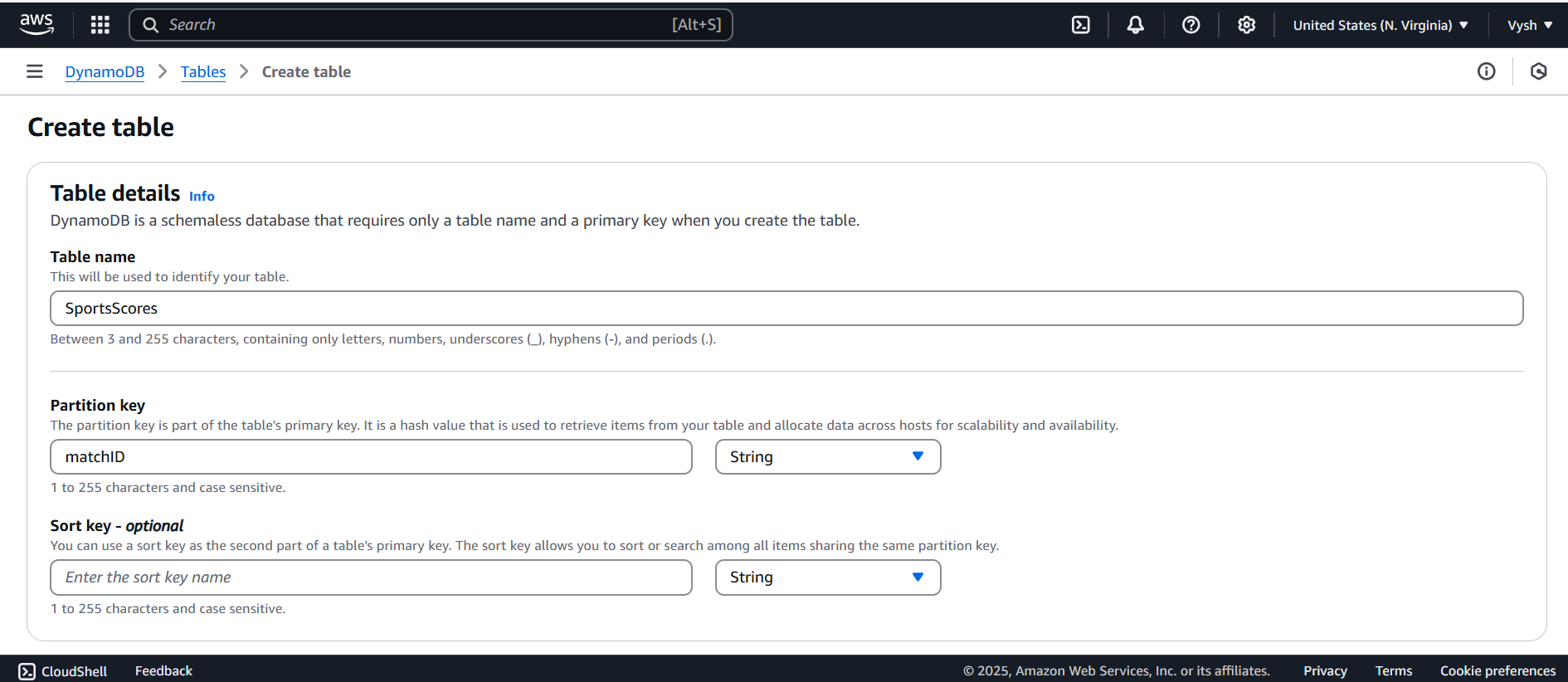


Fig 4.4: Creating a new Table in DynamoDB

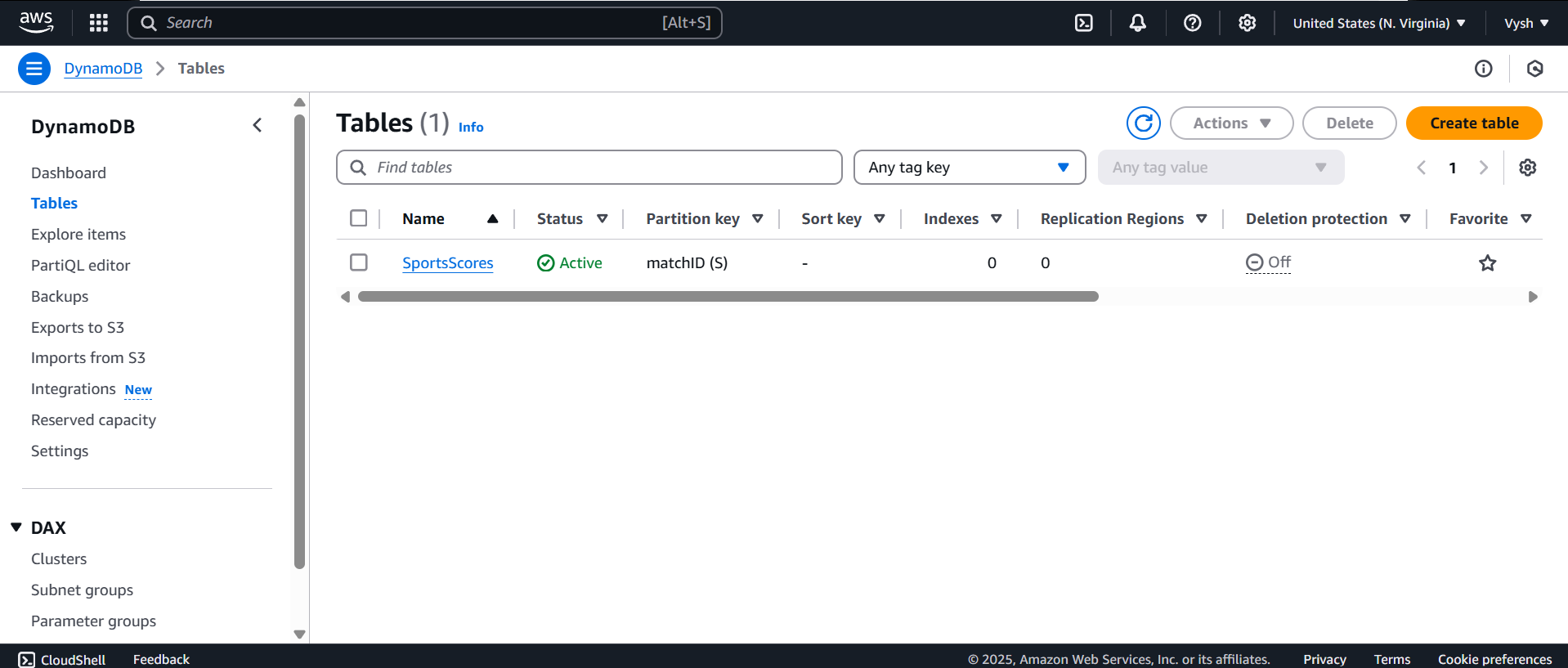


Fig 4.5: Created table named SportsScores

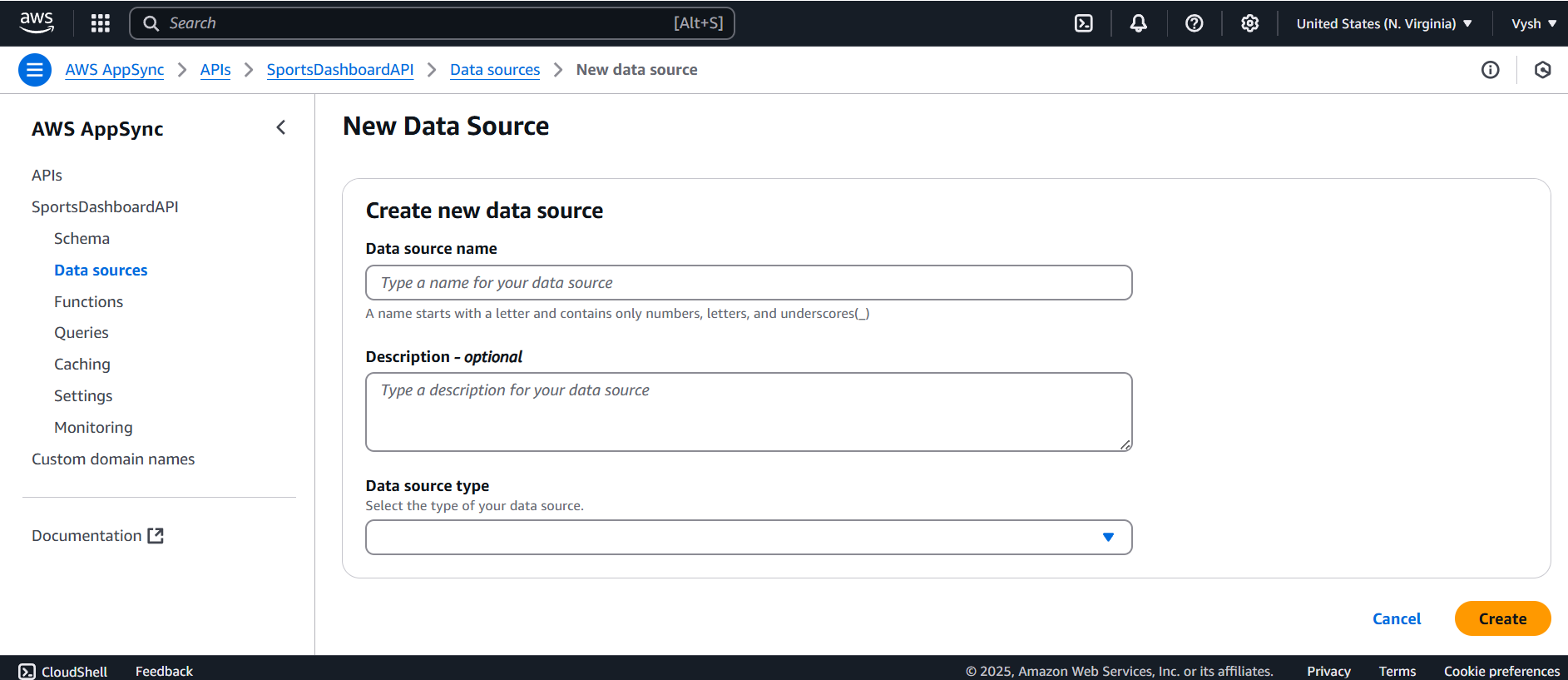


Fig 4.6: Creating A new data Source

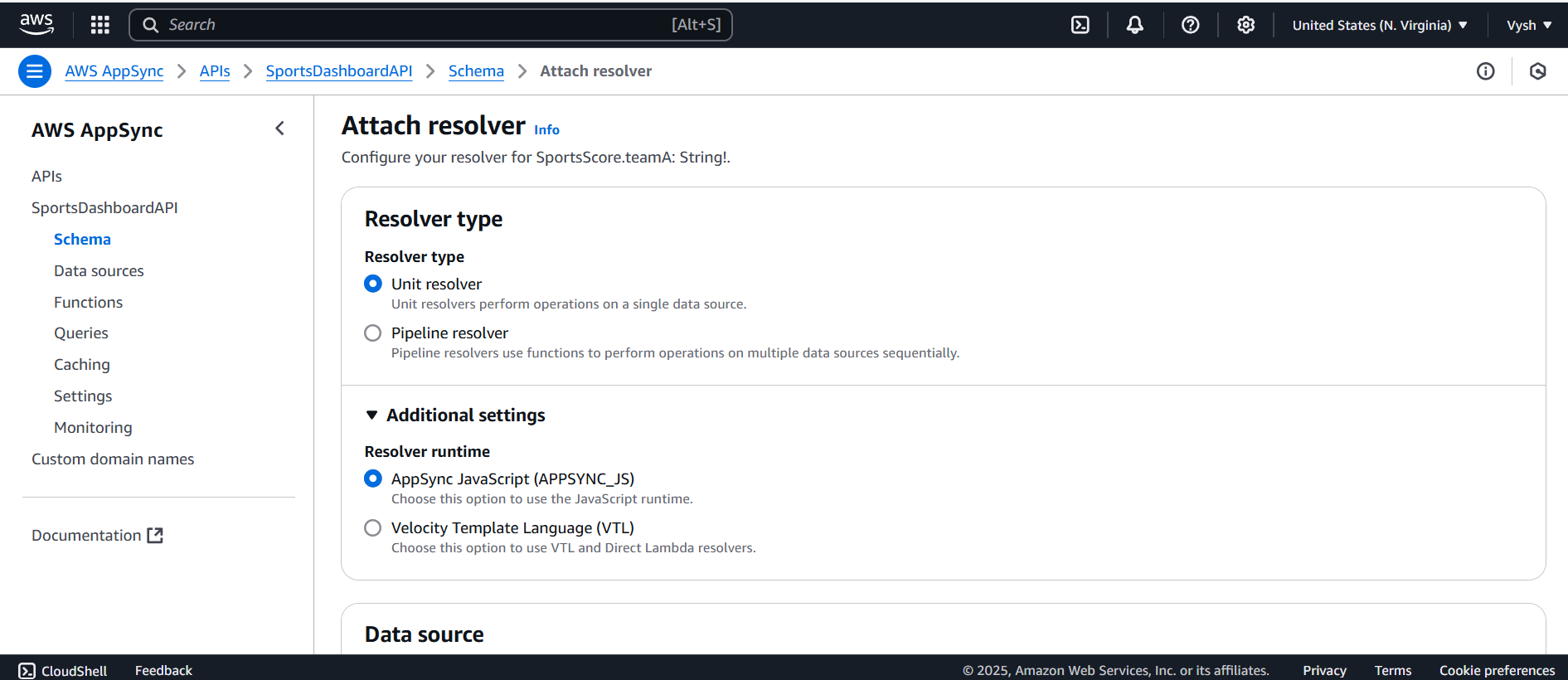


Fig 4.7: Attaching Resolvers

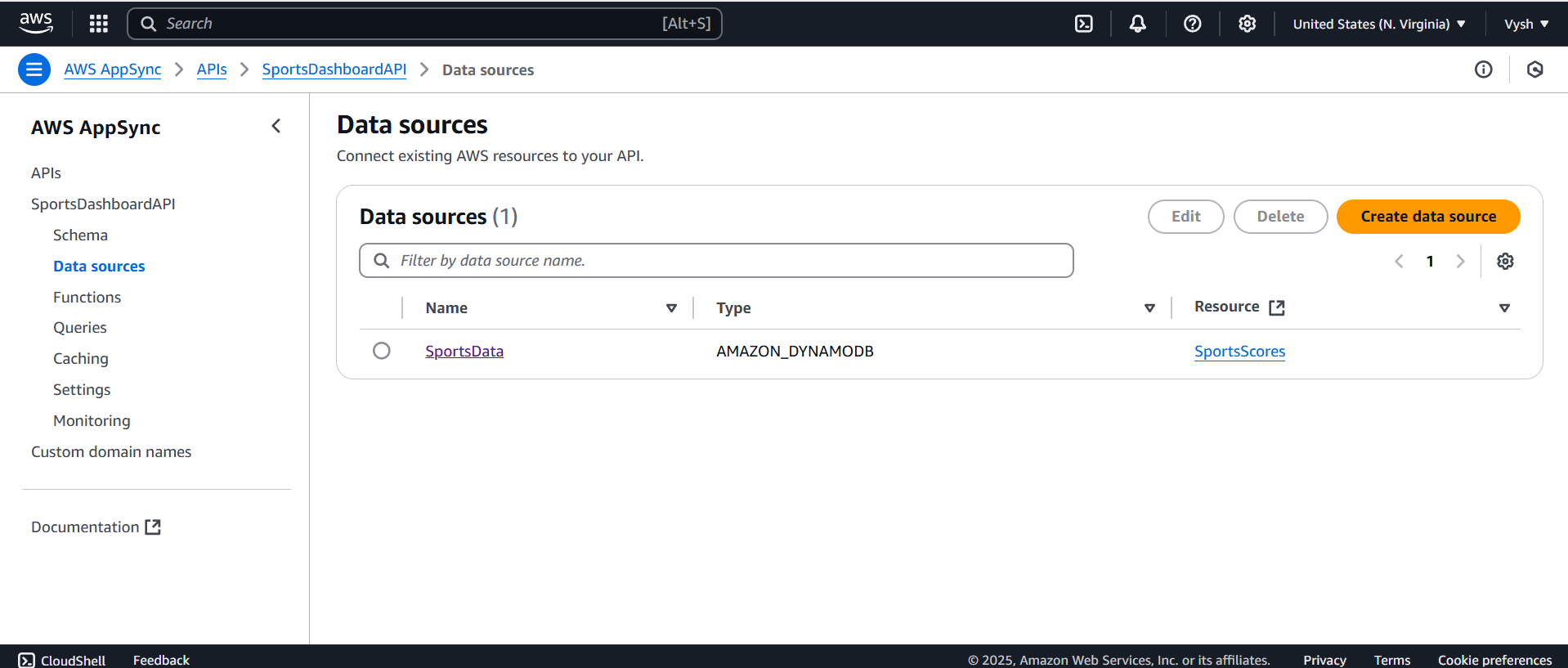
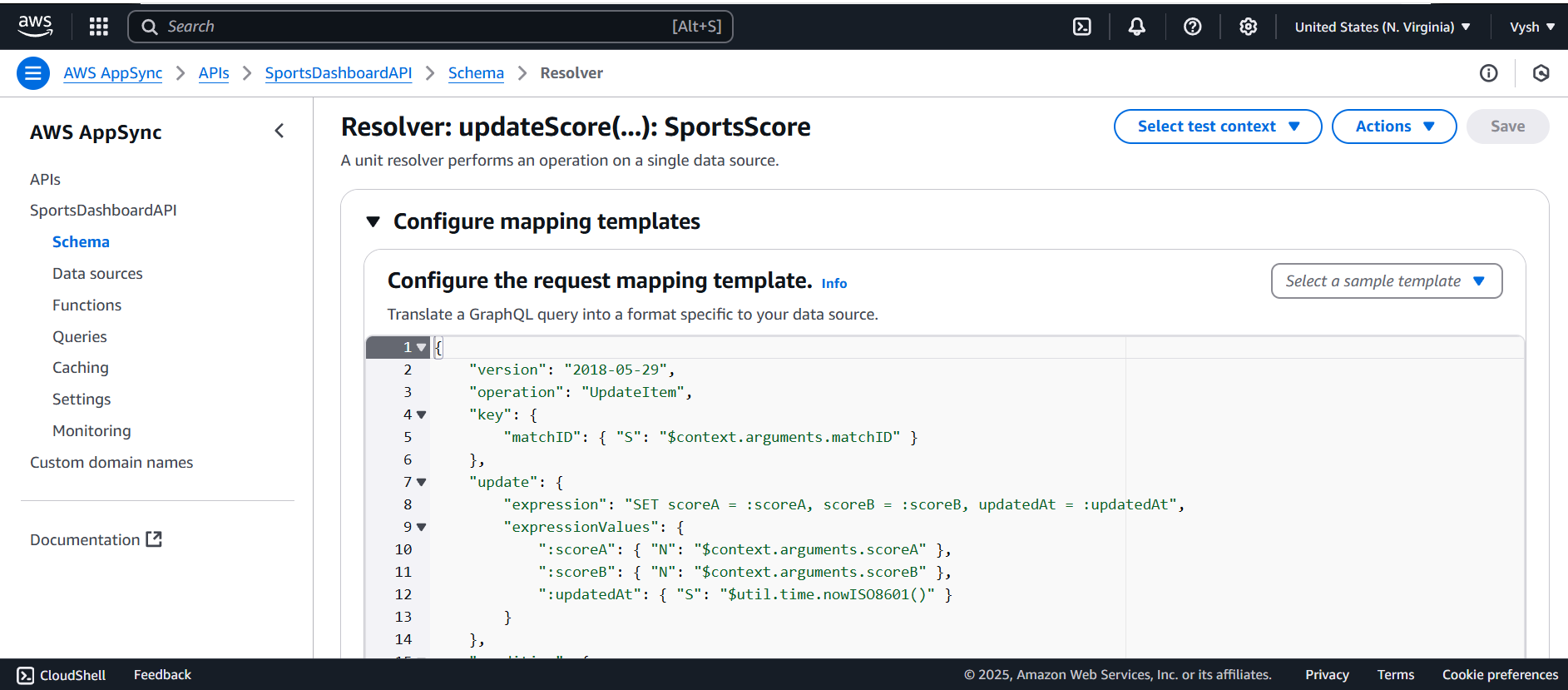


Fig 4.8: Created a new Data Source Named SportsData

  
 Fig 4.9: Configuring the request Mapping Templates  
**Step 3:**To manage permissions securely, a new user group is created in AWS IAM. Within this group, a specific IAM user is added to represent application-level access. Permissions such as dynamodb:Query, dynamodb:PutItem, and other relevant actions are assigned to the user, allowing secure interaction between AppSync and DynamoDB. The access details for this user are carefully noted down for future use in the application integration.

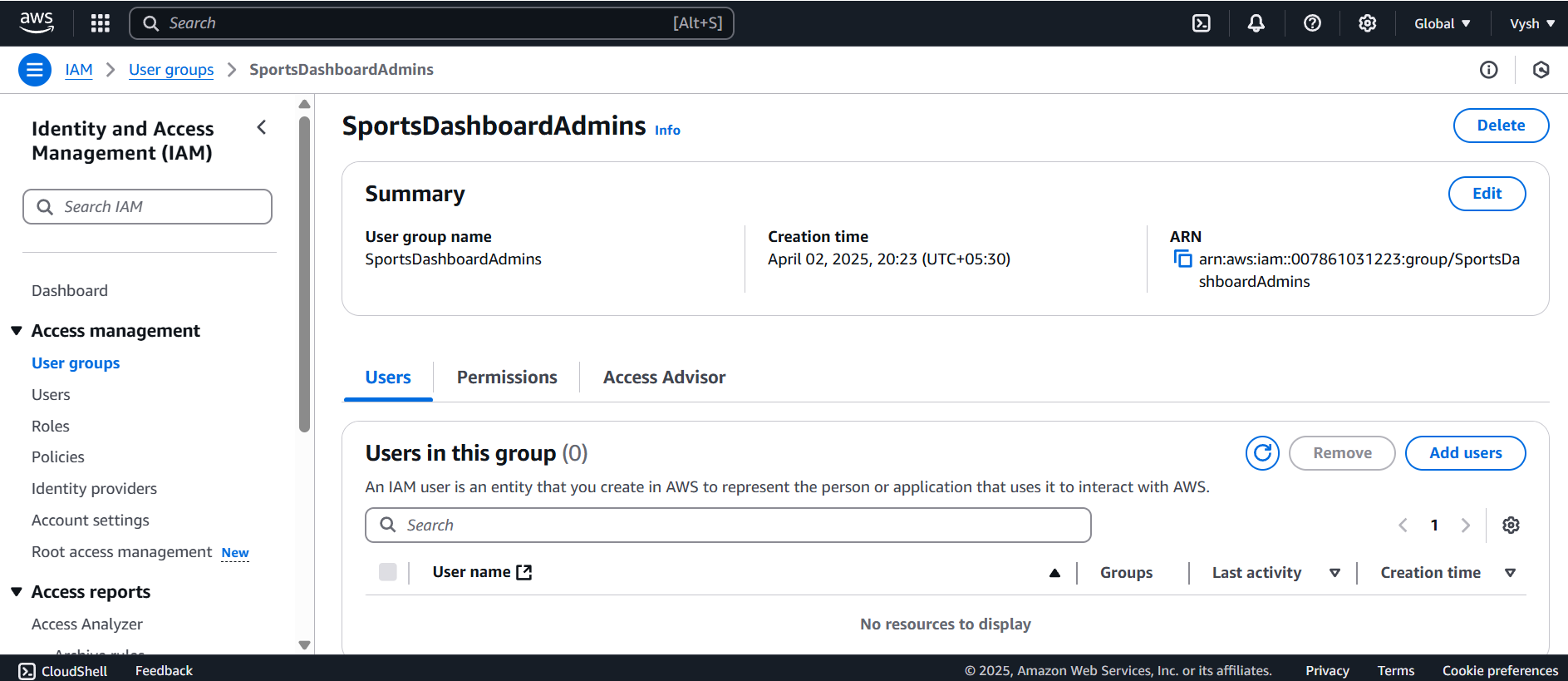
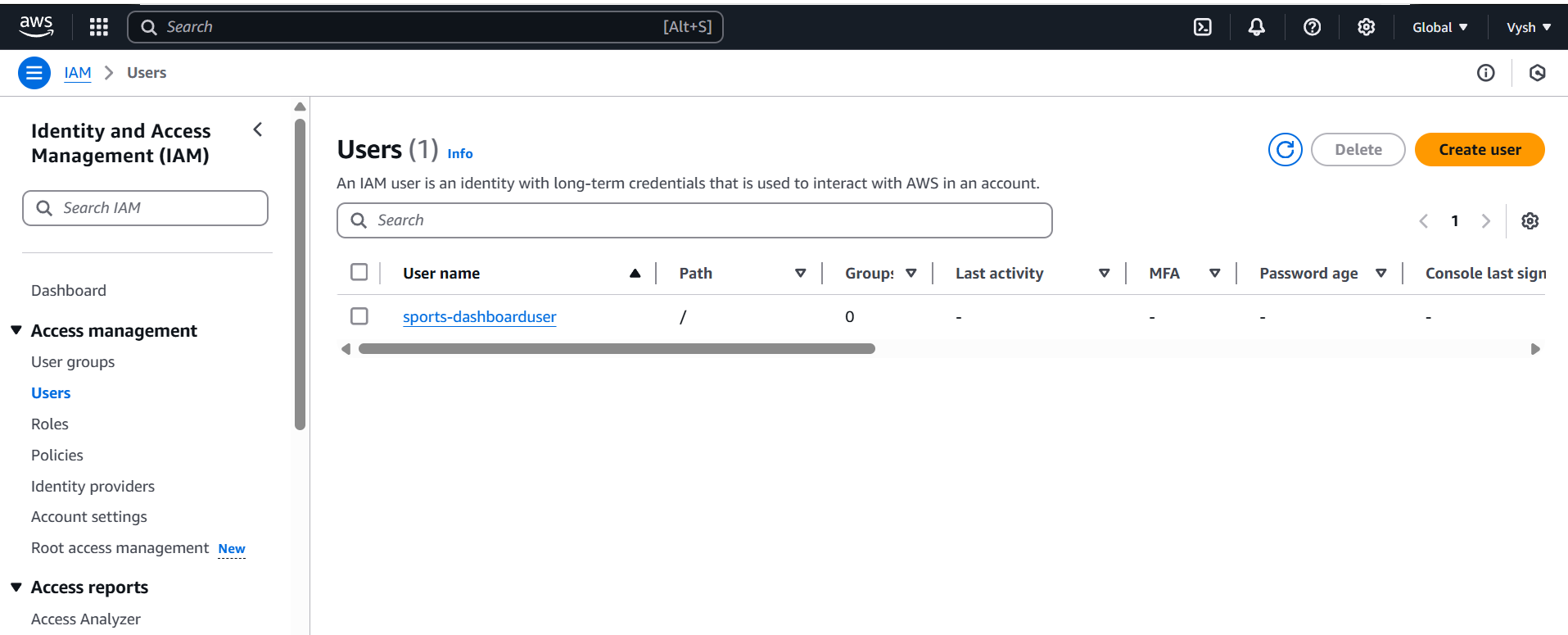


Fig 4.10: Creating new User Group in IAM

  
 Fig 4.11: Creating an IAM User

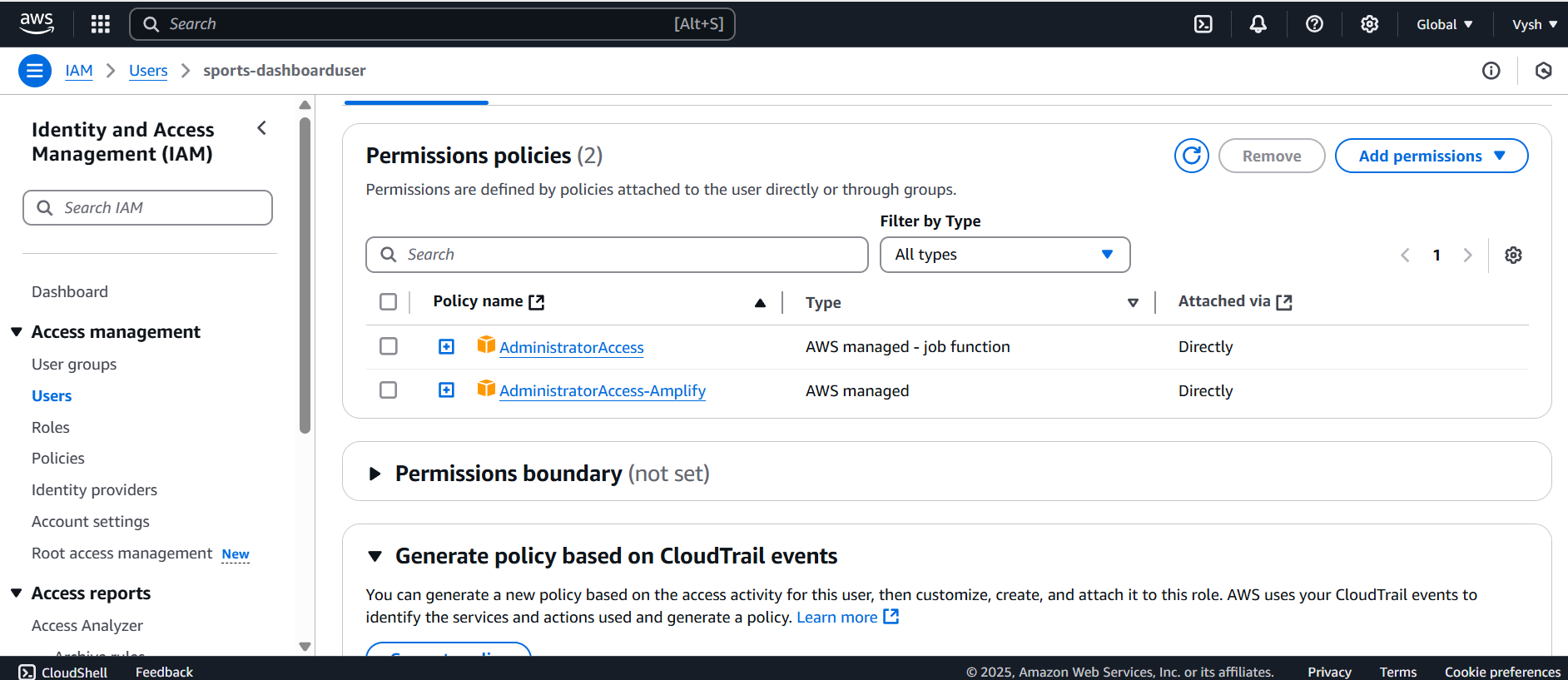
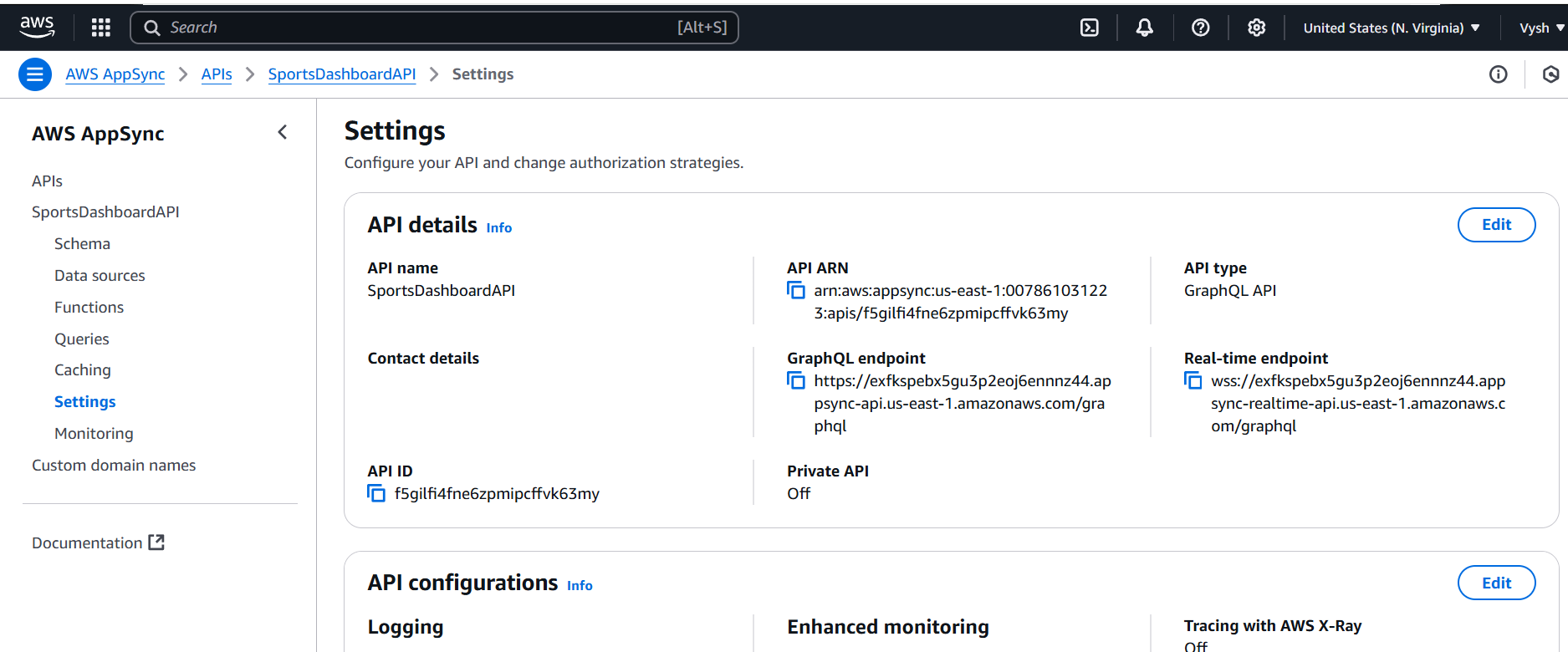


Fig 4.12: Adding the permissions and giving access to the User  
  
  
 Fig 4.13: Note the details

**Step 4:**A new folder is created to organize the frontend files of the project. Inside this folder, the main application script (app.js) and the HTML structure (index.html) are developed. The frontend uses JavaScript to subscribe to GraphQL updates and dynamically displays match scores using Chart.js. This setup ensures that any updates in match data are immediately reflected in the user interface without needing manual page refreshes.

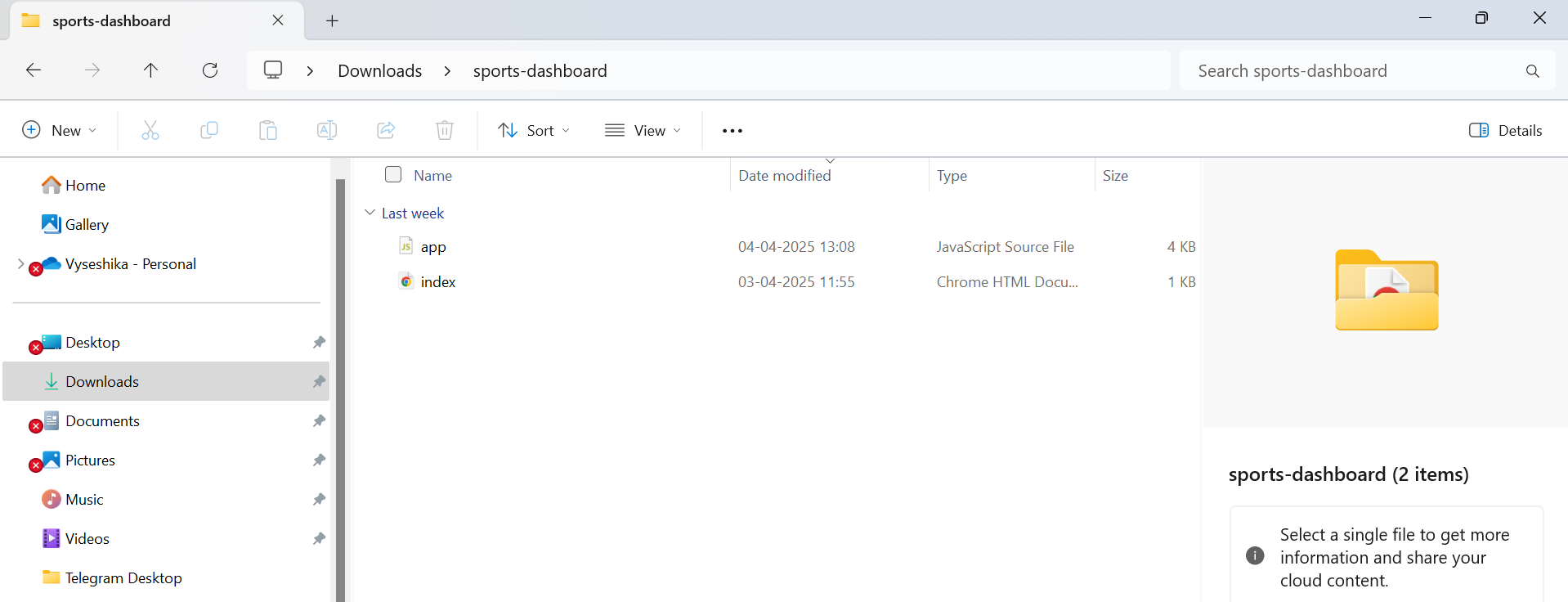


Fig 4.14: Create a new Folder



Fig 4.15: Develop the app.js

****

Fig 4.16: Develop the index.html

**Step 5:**

After all configurations and coding are complete, the project is launched. The final output is a working real-time sports dashboard that displays live cricket scores. The frontend connects seamlessly with AppSync and DynamoDB, and updates are shown instantly through visual charts and dynamic content.

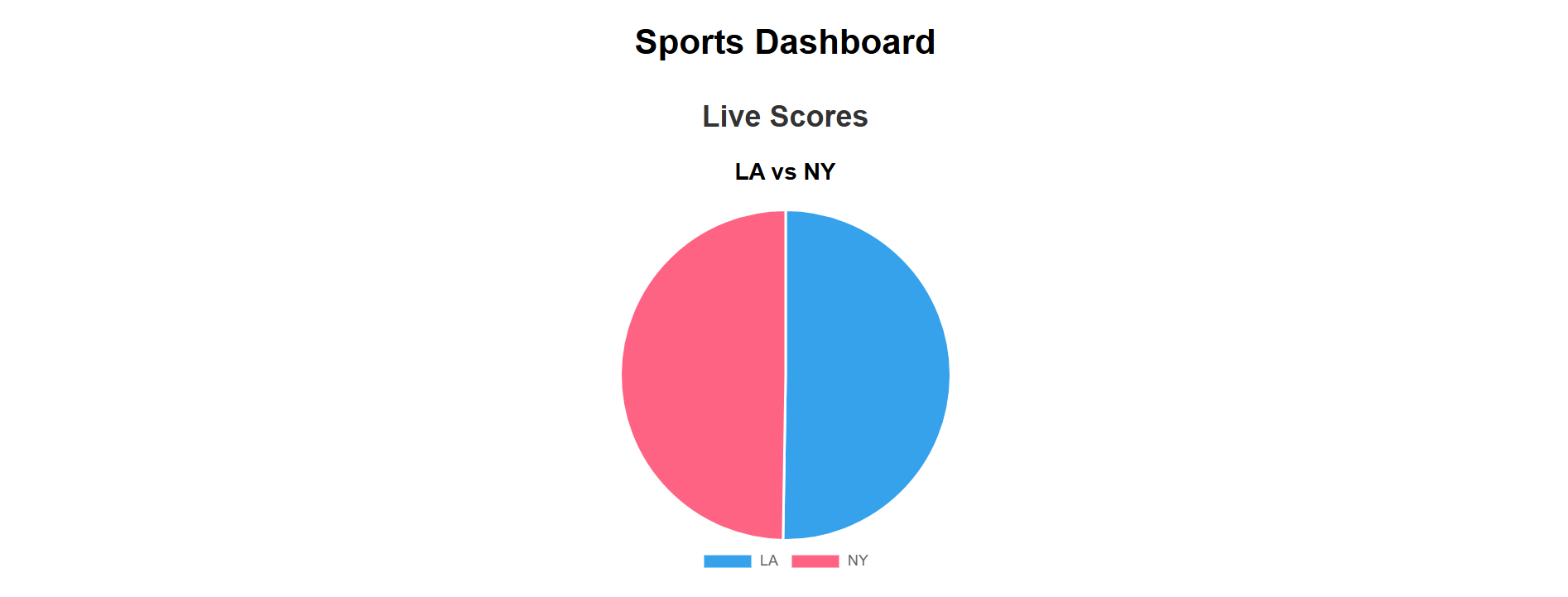


Fig 4.17: Final Outcome after launching

**5.Learning Outcomes**

During the course of this project, hands-on experience was gained in using AWS AppSync and DynamoDB to build serverless applications. The process involved learning how to configure IAM roles effectively to ensure secure interactions between AWS services. A solid understanding of real-time data synchronization was developed through the use of GraphQL subscriptions. Additionally, the project involved exploring methods to efficiently fetch and process external API data. Skills in JavaScript and Chart.js were strengthened for developing interactive and dynamic data visualizations. The role of API Gateway in managing secure access to external data sources was also better understood. Overall, the project enhanced problem-solving abilities by addressing and troubleshooting various challenges encountered in deploying cloud-based applications.

**6.Conclusion**

This project successfully demonstrates how AWS cloud services can be used to create a real-time sports dashboard with automated data updates. By leveraging AWS AppSync, DynamoDB, and IAM roles, we built a system that efficiently fetches, processes, and displays live cricket scores in an interactive format. The serverless architecture ensures scalability, cost efficiency, and minimal maintenance, making it a practical solution for real-time data-driven applications.

**7.Future Scope**

In the future, this system can be enhanced with additional features such as player performance analytics, access to detailed player statistics, historical match data visualization, user-customizable dashboards, and push notifications for live score updates. These improvements can offer a more immersive and informative experience for users, expanding the dashboard’s capabilities and overall engagement

**7.References**

[1]. hams, R., Wang, X., & Li, Y. (2021). "Efficiency of GraphQL Subscriptions in Real-Time Applications." *Journal of Web Engineering*, 20(3), 145-162.

[2]. Hussain, M., Patel, S., & Kumar, R. (2020). "Performance Analysis of NoSQL Databases in Real-Time Applications." *International Journal of Computer Science and Information Security*, 18(5), 231-245.

[3]. Amazon Web Services (AWS). (2023). "AWS AppSync: Real-Time Data Synchronization with GraphQL." Retrieved from <https://aws.amazon.com/appsync/>

[4]. Amazon Web Services (AWS). (2023). "Amazon DynamoDB: High-Performance NoSQL Database." Retrieved from <https://aws.amazon.com/dynamodb/>

[5]. ESPN. (2022). "Leveraging Cloud Technologies for Real-Time Sports Data."

Retrieved from <https://athelogroup.com/blog/ai-in-sports-how-espn-is-leveraging-tech-to-highlight-niche-sports/>