Adaptive IAM System: Azure Guide

* Manik Singla(s222092193)

Overview

This guide walks you through building an Adaptive Identity and Access Management (IAM) system. The system uses Azure Functions, Cosmos DB, and a simple static website to manage user authentication and access based on real-time behaviour analysis. You'll learn how to handle login attempts, store user data securely, and provide personalized feedback to users.

What You’ll Need

* Azure Storage: For hosting the static HTML website.
* Azure Functions: To handle backend logic like user authentication and behaviour logging.
* Cosmos DB: To store user details, login attempts, and roles securely.

Steps to Build the IAM System

**1. Overview:**

The static website serves as the front-end interface, allowing users to interact with the IAM system. It is hosted on Azure Storage, a cost-effective and scalable solution for static content delivery.

**Steps:**

1. **Create an Azure Storage Account**:
   * Create a storage account in the Azure portal with suitable performance and redundancy options.
   * Enable static website hosting for the storage account.
2. **Upload Static Files**:
   * Upload the HTML, CSS, and JavaScript files to the $web container in the Azure Storage account.
3. **Configure Hosting**:
   * Specify the index and error pages in the Azure portal to ensure smooth user navigation.
4. **Ensure Security**:
   * Configure CORS settings to ensure secure communication between the static website and the backend services.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

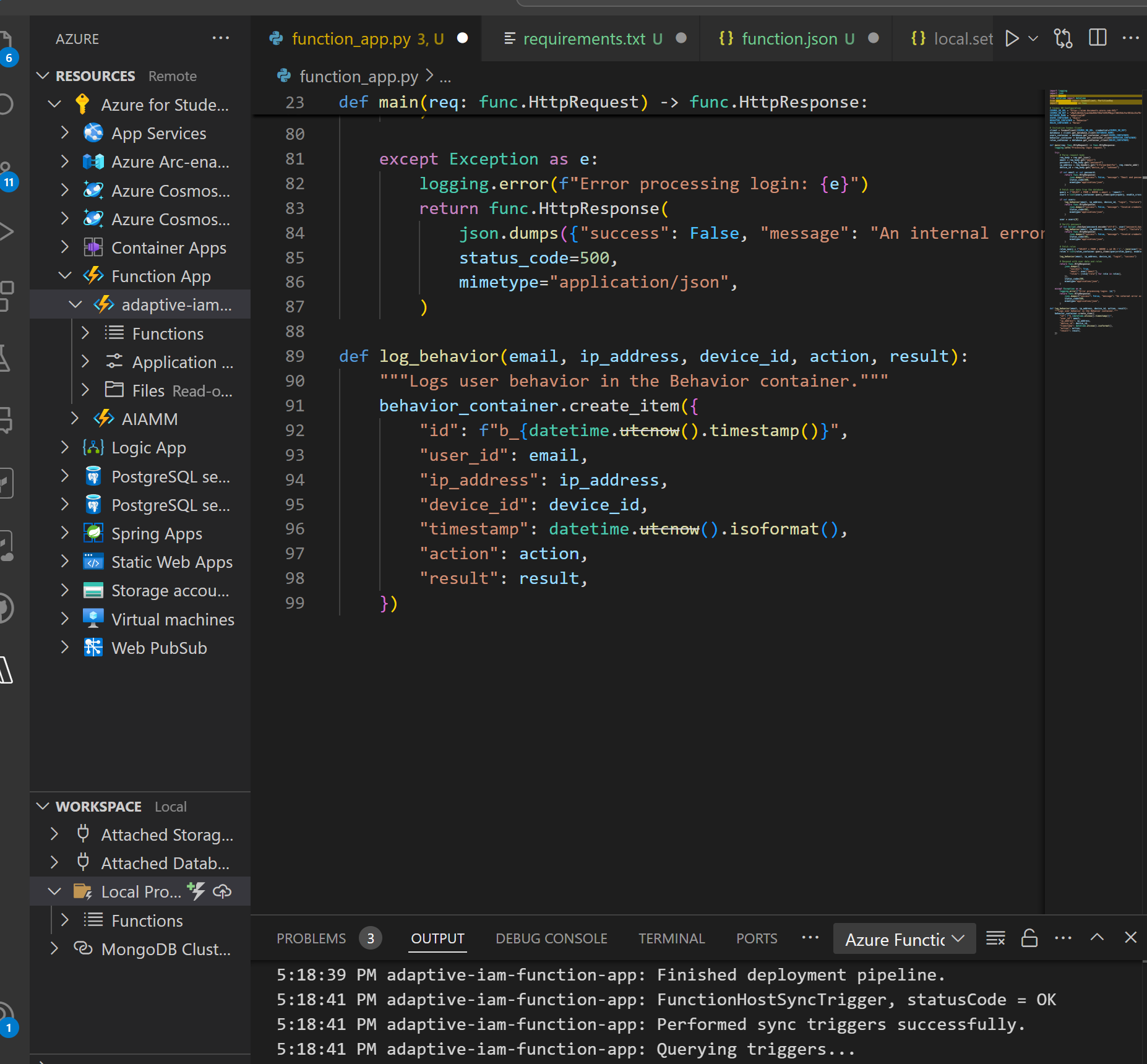
**2. Integrating Azure Functions for Authentication**

**Overview:**

Azure Functions are used to implement the backend logic of the IAM system. These serverless functions handle user authentication requests and validate credentials against stored data.

**Steps:**

1. **Create an Azure Function App**:
   * Set up a Function App in Azure and select the appropriate hosting plan.
   * Ensure that the function app can scale based on demand and that security measures (such as encryption) are applied.
2. **Implement Authentication Logic**:
   * Create HTTP-triggered functions that will receive login requests from the static website.
   * Implement logic to validate the user's credentials and return success or failure responses.
3. **Secure the Function**:
   * Use proper authentication and authorization mechanisms to ensure the security of the function



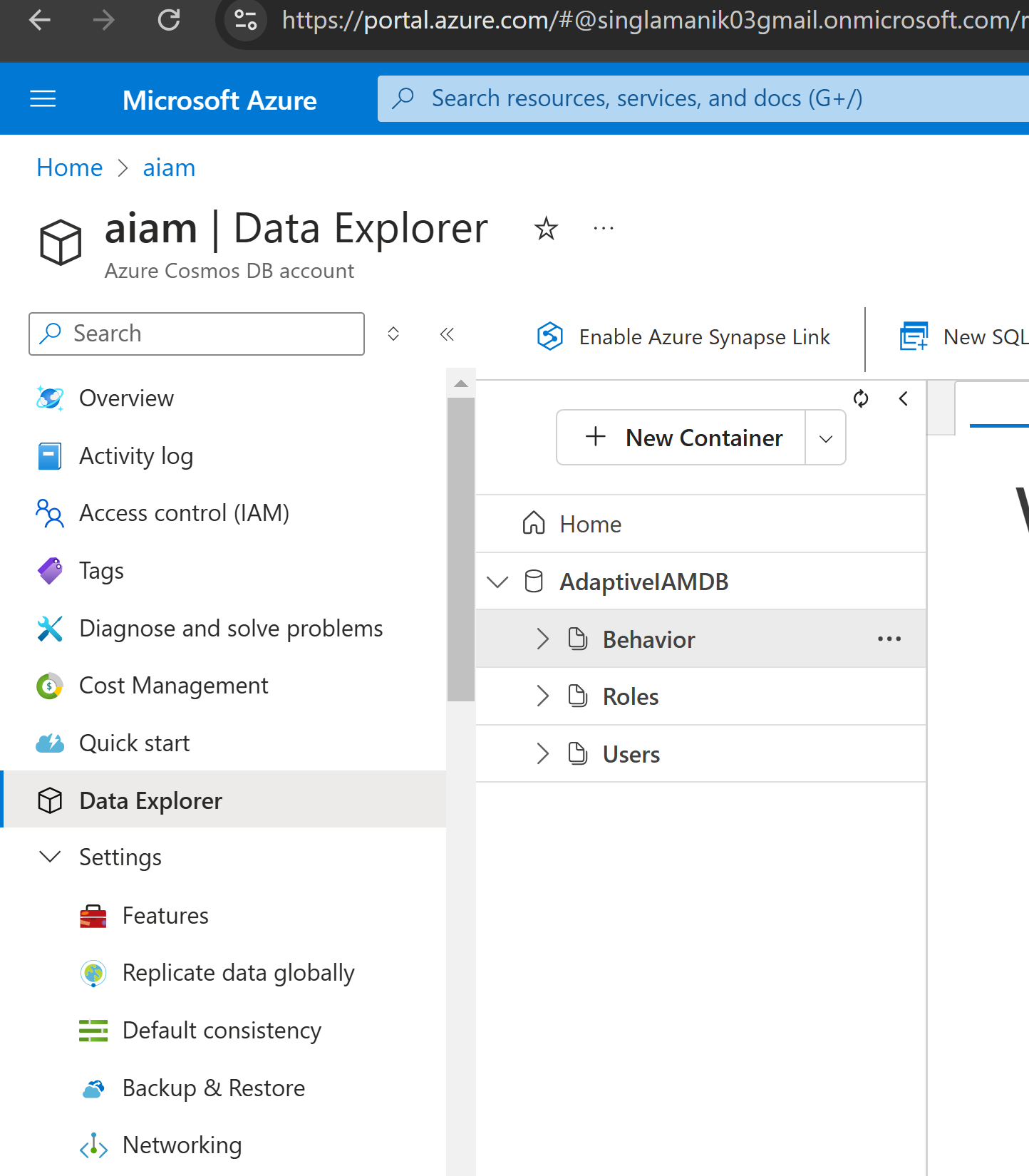
**3. Database Integration for Storing User Data and Behaviour**

**Overview:**

Azure Cosmos DB is used to store user data, roles, and authentication logs. The database is also used to track user behaviors, which can be leveraged for adaptive IAM, improving access control over time.

**Steps:**

1. **Set Up Cosmos DB**:
   * Create a Cosmos DB instance in the Azure portal and configure the necessary database containers for users, roles, and behavior data.
2. **Connect Azure Functions to Cosmos DB**:
   * Use Azure SDKs or REST APIs to integrate the Azure Functions with Cosmos DB to fetch user data and update behavior logs in real-time.
3. **Track and Analyse User Behaviour**:
   * Collect user activity data such as login attempts, IP addresses, and device identifiers to build behavioural analytics. This data can be used to assess access risk and apply adaptive authentication policies.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

**4. Testing and Deployment**

**Overview:**

Before deploying the system to production, thorough testing is required to ensure it works as expected and meets security and performance requirements.

**Steps:**

1. **Test the Static Website**:
   * Ensure that the website is responsive, accessible, and functional across different browsers and devices.
2. **Test the Backend Functions**:
   * Verify that the Azure Functions handle user requests correctly and securely, ensuring proper error handling and logging.
3. **Test Database Connectivity**:
   * Ensure that data retrieval and storage operations in Cosmos DB are efficient and secure.
4. **Deploy to Production**:
   * Once testing is complete, deploy the solution to production, ensuring that all components are configured for optimal performance.

**Challenges Encountered**

**1. Azure Functions: HTTP Trigger Not Working as Expected**

During development, the HTTP-triggered Azure Functions encountered issues such as:

* **Timeouts**: Certain requests took too long to process, leading to timeouts.
* **CORS Configuration**: Improper configuration of CORS settings caused communication failures between the static website and the backend.
* **Deployment Conflicts**: After deploying the function, changes to the underlying logic weren’t immediately reflected, requiring multiple redeployments and troubleshooting.
* **Debugging Limitations**: Debugging the serverless environment in Azure Functions was more challenging compared to a locally hosted solution.



**2. Database Costs Too High**

While using Azure Cosmos DB for user and behavior data storage, the following issues contributed to high operational costs:

* **Provisioned Throughput**: Setting a high Request Unit (RU/s) limit for the database resulted in unnecessary costs during low usage periods.
* **Inefficient Queries**: Complex or non-optimized database queries increased RU consumption, inflating costs.
* **Underutilized Features**: Certain advanced features of Cosmos DB, such as multi-region write support, were enabled but not utilized in the system.

**Future Enhancements**

1. **Advanced Behavioral Analytics**:
   * Incorporate advanced analytics and machine learning techniques to improve the accuracy of access risk assessments.
2. **Scalability Improvements**:
   * Implement auto-scaling for the Azure Functions to handle higher loads and ensure consistent performance during peak times.
3. **Multi-Factor Authentication (MFA)**:
   * Add MFA support to further strengthen the security of the authentication process, especially for high-risk access scenarios.

**Conclusion**

This guide has outlined the key steps involved in building an adaptive IAM system using Azure’s suite of services, including static website hosting, Azure Functions, and Cosmos DB. By leveraging serverless computing and scalable storage solutions, the system can be cost-effective while ensuring high security and flexibility. The implementation of adaptive IAM ensures that the system remains secure while providing a seamless user experience.

This solution is highly scalable, and future enhancements can further strengthen the system’s ability to manage and secure user access efficiently.