Fraud Detection with AWS and Hazelcast Cloud

Yahya Elmokhtari
Adam Essaba

January 1, 2025

Contents

Introduction

Fraud detection is a critical component in financial systems, ensuring the security and trust of transactions. This lab demonstrates the implementation of a fraud detection system using AWS services and Hazelcast Cloud. By following these steps, we validate transactions against stored data to detect anomalies efficiently.

This report provides a detailed explanation of the entire process, including the challenges faced and the solutions implemented to overcome them. The goal is to provide a comprehensive guide for replicating the lab and understanding each step's purpose.

1 Overview of the Lab

The objective of this lab is to build a system that validates transactions against predefined rules. The key goals include:

- Setting up a Hazelcast Viridian Cluster to store and cache data.
- Using AWS Lambda to process and validate transaction data.
- Exposing Lambda functions through API Gateway for external applications.
- Handling errors and optimizing the system for real-time fraud detection.

2 Step-by-Step Implementation

2.1 Step 1: Setting Up the Project Files

Objective: Prepare the environment for the lab by obtaining the necessary files and dependencies.

Process:

1. Clone the GitHub repository for the project:

```
git clone https://github.com/YAHYA280/Fraud_Detection_with_AWS_and_Hazelcast_
```

2. Navigate to the `code/` directory and install the project dependencies:

```
npm install
```

3. Download the TLS certificate files from the Hazelcast Cloud console and save them in the `code/` directory.

2.2 Step 2: Setting Up the AWS CLI

Objective: Configure the AWS CLI to interact with AWS services programmatically.
Process:

1. Install the AWS CLI tool by following the instructions on the official AWS CLI page.
2. Configure the CLI with your AWS credentials:

   ```
   aws configure
   ```

   Provide the Access Key ID, Secret Access Key, and set the default region to `us-west-2`.
3. Test the configuration by listing the S3 buckets:

   ```
   aws s3 ls
   ```

2.3 Step 3: Creating a Hazelcast Viridian Cluster

Objective: Set up a Hazelcast Cloud Standard cluster to store and cache data efficiently.
Process:

1. Log in to the Hazelcast Cloud console.
2. Create a new cluster in the `us-west-2` region to minimize latency with AWS services.
3. Note the cluster name and discovery token for later use.

2.4 Step 4: Creating an S3 Bucket and IAM Roles

Objective: Store airport data in an S3 bucket and provide permissions to Lambda functions.
Process:

1. Create an S3 bucket named `hazelcast-fraud-bucket-12345`:

   ```
   aws s3api create-bucket --bucket hazelcast-fraud-bucket-12345 \
       --region us-west-2 --create-bucket-configuration LocationConstraint=us-we
   ```

2. Create an IAM role `HazelcastServerlessFraudDetection` with permissions for Lambda and S3.
3. Attach the role to the Lambda functions.

2.5 Step 5: Ingesting Airport Data

Objective: Load airport data from S3 into the Hazelcast cluster.
Process:

1. Create a Lambda function `ImportAirportsFn` to read data from S3 and write to the Hazelcast cluster.
2. Zip the relevant files and deploy the function using the AWS CLI:

```
zip -r import.zip import.js hazelcast.js node_modules ca.pem cert.pem key.pem
aws lambda create-function --function-name ImportAirportsFn \
    --role <ROLE_ARN> --zip-file fileb://import.zip --handler import.handle \
    --runtime nodejs16.x --region us-west-2
```

3. Configure S3 to trigger the Lambda function on file uploads.
4. Upload the `airports.json` file to the S3 bucket and verify data ingestion.

2.6 Step 6: Validating Transactions

Objective: Implement a Lambda function to validate transactions using Hazelcast.
Process:

1. Develop the `validate.js` file to include fraud detection logic.
2. Package the code and deploy it as a Lambda function `ValidateFn`:

```
zip -r validate.zip validate.js hazelcast.js node_modules ca.pem cert.pem key
aws lambda create-function --function-name ValidateFn \
    --role <ROLE_ARN> --zip-file fileb://validate.zip --handler validate.hand
    --runtime nodejs16.x --region us-west-2
```

3. Test the function in the AWS Lambda console using sample data.

2.7 Step 7: Creating the API Gateway Endpoint

Objective: Expose the `ValidateFn` Lambda function as a REST API.
Process:

1. Create a REST API in API Gateway named `ServerlessFraudDetectionGateway`.
2. Add a resource `/ValidateTransactions` with a POST method linked to `ValidateFn`.
3. Deploy the API to a stage named `Test` and copy the invoke URL.
4. Test the API endpoint using tools like `httpie` or Postman.

2.8 Step 8: Testing the System

Objective: Ensure the entire pipeline functions correctly.
Process:

1. Send valid and invalid transaction data to the API endpoint.
2. Confirm correct validation responses.
3. Review logs in CloudWatch for debugging and monitoring.

3 Challenges and Solutions

Summary of Issues Encountered:

- **Permission Errors:** Resolved by attaching correct IAM policies.
- **Syntax Errors:** Fixed `await` misplacement in the `validate.js` file.
- **API Gateway Errors:** Corrected endpoint URL and HTTP method.
- **Data Ingestion:** Verified data upload and caching using Hazelcast logs.

4 Potential Errors and Their Solutions

Throughout the lab, several errors were encountered. Below is a summary of the potential errors and the steps taken to solve them:

4.1 Error: Permission Denied for AWS CLI Commands

Issue: The AWS CLI returned `AccessDenied` errors during bucket creation or Lambda deployment. **Solution:** Ensure that the IAM user or role has the necessary permissions by attaching policies such as `AmazonS3FullAccess`, `AWSLambdaFullAccess`, and `IAMFullAccess`.

4.2 Error: `MissingAuthenticationToken` for API Gateway

Issue: API Gateway returned a `MissingAuthenticationToken` error when testing the endpoint. **Solution:** Verify that the correct endpoint URL and HTTP method (POST) are used. Additionally, ensure the API is deployed to a stage.

4.3 Error: Syntax Error in `validate.js`

Issue: An `await` statement was used outside of an `async` function, causing a runtime error. **Solution:** Move the `await` statements into properly defined `async` functions. Debugging and testing fixed this issue.

4.4 Error: Data Not Ingested into Hazelcast Cluster

Issue: The `ImportAirportsFn` function did not load data into the Hazelcast cluster. **Solution:** Confirm the S3 bucket triggers are correctly set up. Check the Lambda logs in CloudWatch for detailed error messages and fix any file path or permission issues.

4.5 Error: API Gateway Deployment Issues

Issue: The API Gateway resource could not be deployed due to missing configurations. **Solution:** Ensure that all resources and methods are configured correctly and linked to the appropriate Lambda function. Redeploy the API after making the changes.

5 Conclusion

This lab demonstrates how to build a scalable, efficient fraud detection system using AWS and Hazelcast Cloud. By leveraging Lambda functions, API Gateway, and Hazelcast caching, the system achieves real-time validation of transactions. The detailed steps and solutions provided here should serve as a comprehensive guide for similar projects.

Future Work

- Implementing additional validation rules to enhance fraud detection.
- Scaling the system to handle a higher volume of transactions.
- Integrating user authentication for securing API Gateway endpoints.