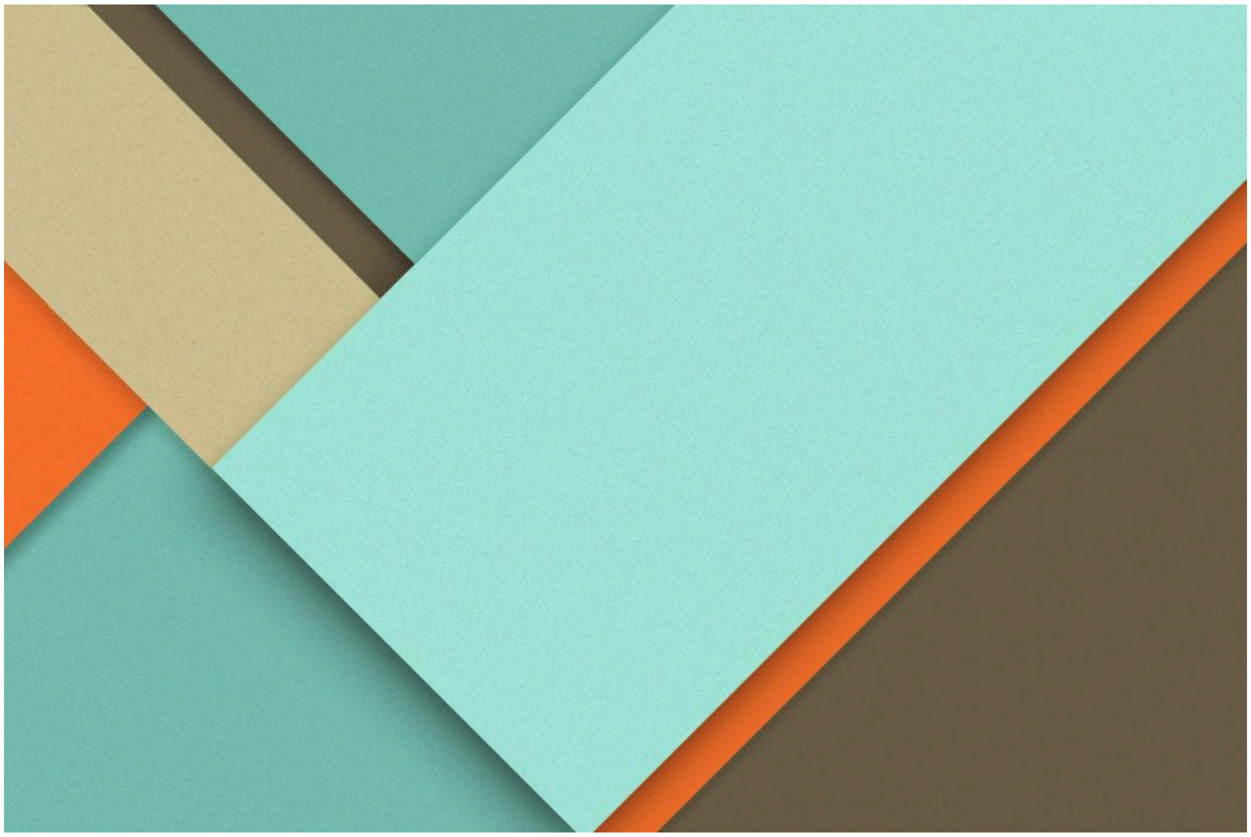


Lamda Data Analyser

12.08.2024

Christeena Giji
DashboardWorx

[Linkedin Profile]

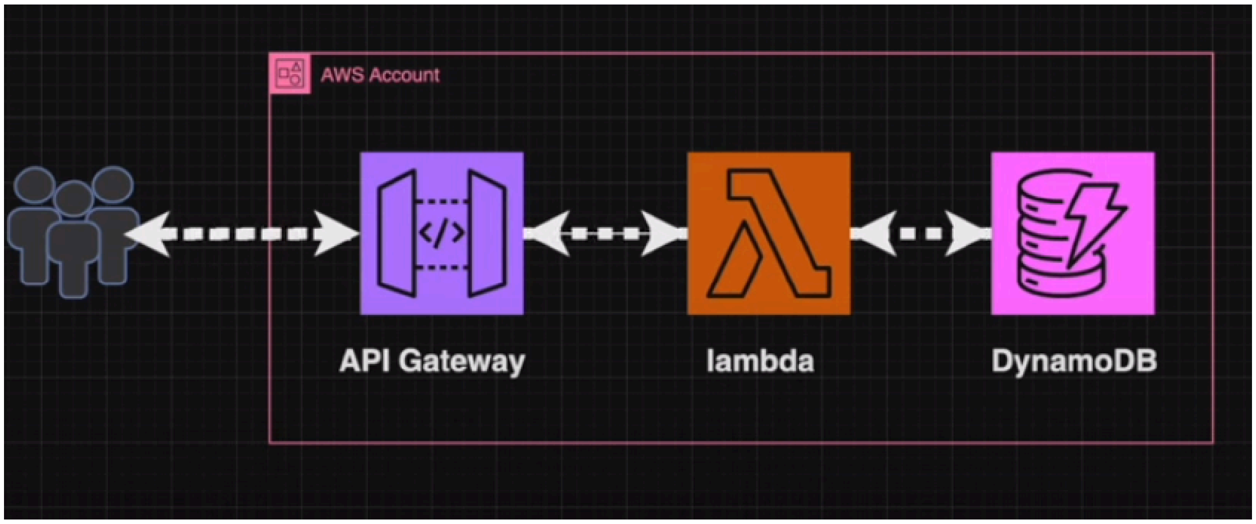
[Github]

[Tableau]

[Portfolio]

Objective:

The purpose of this project is to develop **LambdaDataAnalyzer**, a cloud-based data analytics service that processes and stores application usage metrics in real-time. The system leverages AWS Lambda for scheduled data processing tasks, DynamoDB for scalable NoSQL data storage, and Flask API for data access, allowing for real-time data visualization. This report outlines the components, setup, and testing of the LambdaDataAnalyzer system.



Components:

1. **AWS Lambda Functions:**
    - **Purpose:** The Lambda functions are triggered on a schedule (e.g., hourly) to process raw application usage logs. They calculate key metrics such as daily usage, session count, and time spent on tasks.
    - **Execution:** The Lambda functions are written in Python and utilize the AWS SDK (boto3) to interact with DynamoDB.
    - **Output:** The processed metrics are stored in a DynamoDB table for easy retrieval.
2. **DynamoDB:**
    - **Purpose:** DynamoDB serves as the data storage layer, holding the processed analytics data. It's optimized for fast and predictable performance.
    - **Structure:** The table is designed with a partition key (`Date`) to organize and query data efficiently by day.
3. **Flask API:**
    - **Purpose:** The Flask API serves as the interface for accessing the data stored in DynamoDB. It exposes endpoints for both retrieving and storing data.
    - **Routes:**

        - **GET `/data`:** Fetches the processed data from DynamoDB.
        - **POST `/data`:** Allows new data to be added to the DynamoDB table.
4. **API Gateway:**
    - **Purpose:** API Gateway is configured to handle incoming HTTP requests and route them to the appropriate Lambda functions. It acts as a bridge between the frontend application and the backend services.
    - **Endpoints:** Two API endpoints are set up—one for GET requests to fetch data and one for POST requests to store new data.
5. **CloudWatch Events:**
    - **Purpose:** CloudWatch Events are used to trigger the Lambda functions on a defined schedule. For example, a cron job is set to run the Lambda function every hour to ensure continuous data processing.
6. **Development Environment:**
    - **Tools:** The project is developed using Visual Studio Code (VSC) with the AWS Toolkit extension for easy deployment and management of AWS resources. The development environment is set up with Flask and boto3, installed in a virtual environment to manage dependencies.

Use Case: The system is designed for applications that require continuous monitoring and analysis of user engagement. It provides a scalable solution for collecting, processing, and analyzing user behavior data, which can be visualized through a frontend application. This enables the client to gain insights into metrics such as day-on-day retention, session counts, and time-on-task, aiding in data-driven decision-making.

How to set up:

Create an IAM Role

- Go to the AWS Management Console.
- Navigate to IAM > Role>create role
- Select use case as “Lamda”
- Permissions policies as below

AWSLambdaBasicExecutionRole

AmazonDynamoDBFullAccess

AWSLambdaInvocation-DynamoDB

- Next> Name the role (e.g., LambdaDynamoDBRole)>Create role

Set Up AWS Lambda Function

I. Create a Lambda Function

1. Log in to your AWS Management Console.
2. Navigate to AWS Lambda.
3. Click on Create Function.
4. Give any Function name
5. Rutime- Python 3.8(or Node.js if you prefer)
6. Execution Role: Use an existing role and chose the role we have created before
7. Create Function

II. Write the Lambda Function Code:

In the Lambda function editor, use the following Python code to fetch, process, and store data in DynamoDB:

```
# boto3 :aws sdk for python and allows to interact with aws services including dynamodb and datetime module to get current date and time

import boto3

from datetime import datetime

 dynamodb = boto3.resource('dynamodb')

table = dynamodb.Table('DynamoDBTableName')

 #Lamda handler function- this is entry point for lamda function and will call this function whenever the lambda is triggered.

def lambda_handler(event, context):

    daily_usage = calculate_daily_usage(event)  #This function would process the event data to calculate the daily usage.

    session_count = calculate_session_count(event) #This function would count the number of sessions based on the event data.

    time_on_task = calculate_time_on_task(event) #This function would calculate the time spent on a specific task.

    # Write the data to DynamoDB
```

```
  table.put_item( Item={ 'Date': str(datetime.now().date()),

 'DailyUsage': daily_usage,

'SessionCount': session_count,

'TimeOnTask': time_on_task } )
```

#Return response

```
return {

'statusCode': 200,

'body': 'Data processed and stored successfully'

}
```

def calculate_daily_usage(event): # Your logic to calculate daily usage return 0 def calculate_session_count(event): # Your logic to calculate session count return 0 def calculate_time_on_task(event): # Your logic to calculate time on task return 0

III. Click Deploy to save and deploy your function.

Schedule Lambda Function

Create a CloudWatch Event Rule and link with Lamda:

1. Go to Amazon CloudWatch.
2. Navigate to "Rules" under Events.
3. Click Create Rule.

- Event Source: Schedule (e.g., cron(0 * * * ? *) for hourly execution).
- Add Target: Select your Lambda function (e.g., "DataAnalyticsProcessor").
- Click "Configure details"
- Provide a name and description
- Click on "Create the Rule".

4. Verify scheduling :Ensure that your CloudWatch rule is listed and associated with the correct Lambda function.

Set Up Your Development Environment

I. Set Up Your Development Environment

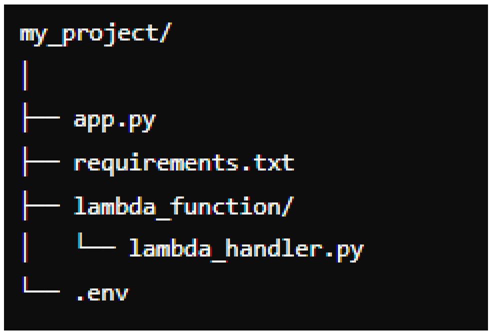
- Open VSC.

- Go to the Extensions view by clicking on the Extensions icon in the sidebar or pressing Ctrl+Shift+X.
- Search for "AWS Toolkit" and click "Install"
- Install AWS CLI
    1. Configure AWS CLI: Open terminal or command prompt
    2. Run 'aws configure'
    3. Enter your AWS access key ID, secret access key, region, and output format when prompted.
- Create a virtual environment and install flask and boto3 in Virtual studio code (VSC)

  pip install Flask boto3

II. Set Up Your Project Structure

Create a project directory and set up a basic structure:



```
my_project/
|
├── app.py
├── requirements.txt
├── lambda_function/
|   └── lambda_handler.py
└── .env
```

Create and Configure Flask App

I. In 'app.py', create a Basic Flask App:

```
from flask import Flask, jsonify, request

import boto3

import os


app = Flask(__name__)


# Initialize DynamoDB

dynamodb = boto3.resource('dynamodb')

table = dynamodb.Table(os.getenv('DYNAMODB_TABLE_NAME'))


@app.route('/')

def home():
    return "Welcome to the Data Analytics Service!"


@app.route('/data', methods=['GET'])
def get_data():
    # Fetch data from DynamoDB
    response = table.scan()
    data = response['Items']
    return jsonify(data)


@app.route('/data', methods=['POST'])
def store_data():
    item = request.json
    table.put_item(Item=item)
    return jsonify({'status': 'success'}), 200


if __name__ == '__main__':
    app.run(debug=True)
```

GET Request: @app.route('/data', methods=['GET']) - This route handles requests to fetch data from DynamoDB and display it to the user. For example, fetching metrics like daily usage, session count, etc., from the database.

POST Request: @app.route('/data', methods=['POST']) - This route handles requests to store new data in DynamoDB. For example, if you have new data or metrics to add to the database, you'd use a POST request.

II. List the requirements in ‘requirements.txt’ :

```
Flask
boto3
```

Create DynamoDB Table

Create the DynamoDB table where the processed data will be stored.

- Navigate to **DynamoDB** in the AWS Management Console.
- Click **Create Table**.
- Set the table name (e.g., `AnalyticsData`) and set **Partition Key** to `Date` (String).
- Click **Create**.

Setup API Gateway

Set up API Gateway to interact with your Lambda functions.

I. Get API gateway

- Click on **Rest API**(works only inside our vpc)>"Build"
- New API
- Assign any name to API
- Endpoint> Regional
- Create API
- In API> Create method > Method type-"**Get**"
- Enable **Lambda proxy Integration**(To get structured event)
- Select correct region and lambda function
- Create Method

II. Post API gateway

- Click on Rest API(works only inside our vpc)>"Build"
- New API
- Assign any name to API
- Endpoint> Regional
- Create API
- In API> Create method > Method type-"**Post"**
- Enable **Lambda proxy Integration**(To get structured event)
- Select correct region and lambda function
- Create Method
- Deploy API with any stage name and note the Invoke URL.

Verification and Testing

I. Test API Gateway:

- Use tools like Postman or Curl to send GET and POST requests to your API Gateway endpoints and verify that data is correctly processed and stored in DynamoDB.

II. Monitor CloudWatch Logs:

- Check CloudWatch logs to ensure that your Lambda function is executing correctly.

Conclusion

The LambdaDataAnalyzer project successfully integrates AWS Lambda, DynamoDB, Flask API, and API Gateway to provide a scalable and real-time analytics solution. The setup ensures continuous data processing and easy access to metrics, enabling data-driven decision-making through real-time visualization. Future enhancements can include more complex data processing and additional API endpoints to further expand functionality.