## Architecture Overview

This project implements a **Hexagonal Architecture (Ports and Adapters)** pattern using AWS services:

* **Primary Adapters**: API Gateway, Amazon SQS (input ports)
* **Domain Logic**: Central business logic (hexagon)
* **Secondary Adapters**: DynamoDB (output ports)
* **Lambda Functions**: Handle the port communications

## Prerequisites

* AWS Free Tier Account
* Basic understanding of AWS services
* AWS CLI installed (optional but recommended)

## Step-by-Step Implementation

### Step 1: Create DynamoDB Table (Secondary Adapter)

1. Navigate to DynamoDB Console
   * Go to AWS Console → Search "DynamoDB" → Click DynamoDB
2. Create Table
   * Click "Create table"
   * Table name: UserData
   * Partition key: userId (String)
   * Leave sort key empty
   * Use default settings (stays within free tier)
   * Click "Create table"
3. Note the Table ARN (needed for Lambda permissions)

### Step 2: Create Lambda Functions (Domain Logic)

#### Lambda Function 1: API Handler

1. Create Function
   * AWS Console → Lambda → Create function
   * Choose "Author from scratch"
   * Function name: api-handler
   * Runtime: Python 3.9
   * Architecture: x86\_64
   * Click "Create function"
2. Add Code

import json

import boto3

from decimal import Decimal

dynamodb = boto3.resource('dynamodb')

table = dynamodb.Table('UserData')

def lambda\_handler(event, context):

try:

http\_method = event['httpMethod']

if http\_method == 'POST':

body = json.loads(event['body'])

user\_id = body.get('userId')

data = body.get('data')

table.put\_item(Item={

'userId': user\_id,

'data': data

})

return {

'statusCode': 200,

'body': json.dumps({'message': 'Data saved successfully'})

}

elif http\_method == 'GET':

user\_id = event['queryStringParameters'].get('userId')

response = table.get\_item(Key={'userId': user\_id})

if 'Item' in response:

return {

'statusCode': 200,

'body': json.dumps(response['Item'], default=str)

}

else:

return {

'statusCode': 404,

'body': json.dumps({'message': 'User not found'})

}

except Exception as e:

return {

'statusCode': 500,

'body': json.dumps({'error': str(e)})

}

#### Lambda Function 2: SQS Handler

1. Create Function
   * Function name: sqs-handler
   * Runtime: Python 3.9
   * Click "Create function"
2. Add Code

import json

import boto3

dynamodb = boto3.resource('dynamodb')

table = dynamodb.Table('UserData')

def lambda\_handler(event, context):

try:

for record in event['Records']:

message\_body = json.loads(record['body'])

user\_id = message\_body.get('userId')

data = message\_body.get('data')

table.put\_item(Item={

'userId': user\_id,

'data': data,

'processed': True

})

return {

'statusCode': 200,

'body': json.dumps('Messages processed successfully')

}

except Exception as e:

print(f"Error: {str(e)}")

raise e

### Step 3: Configure Lambda Permissions

1. For both Lambda functions:
   * Go to Configuration → Permissions
   * Click on the execution role
   * Add inline policy:

{

"Version": "2012-10-17",

"Statement": [

{

"Effect": "Allow",

"Action": [

"dynamodb:PutItem",

"dynamodb:GetItem",

"dynamodb:UpdateItem",

"dynamodb:DeleteItem"

],

"Resource": "arn:aws:dynamodb:REGION:ACCOUNT:table/UserData"

}

]

}

### Step 4: Create SQS Queue (Primary Adapter)

1. Navigate to SQS Console
   * AWS Console → SQS → Create queue
2. Configure Queue
   * Type: Standard queue
   * Name: user-data-queue
   * Use default settings
   * Click "Create queue"
3. Connect SQS to Lambda
   * In SQS queue → Lambda triggers → Configure Lambda function trigger
   * Select sqs-handler function
   * Batch size: 10
   * Click "Save"

### Step 5: Create API Gateway (Primary Adapter)

1. Create API
   * AWS Console → API Gateway → Create API
   * Choose "REST API" → Build
   * API name: user-data-api
   * Click "Create API"
2. Create Resource
   * Actions → Create Resource
   * Resource name: users
   * Resource path: /users
   * Click "Create Resource"
3. Create Methods

**POST Method:**

* Select /users resource → Actions → Create Method
* Choose POST → Integration type: Lambda Function
* Lambda function: api-handler
* Click "Save"

**GET Method:**

* Select /users resource → Actions → Create Method
* Choose GET → Integration type: Lambda Function
* Lambda function: api-handler
* Click "Save"

1. Deploy API
   * Actions → Deploy API
   * Deployment stage: prod
   * Click "Deploy"
   * Note the Invoke URL

### Step 6: Test the Architecture

#### Test API Gateway

# POST request

curl -X POST https://YOUR-API-ID.execute-api.REGION.amazonaws.com/prod/users \

-H "Content-Type: application/json" \

-d '{"userId": "user123", "data": "test data"}'

# GET request

curl https://YOUR-API-ID.execute-api.REGION.amazonaws.com/prod/users?userId=user123

#### Test SQS

1. Go to SQS Console → Send message
2. Message body:

{

"userId": "user456",

"data": "SQS test data"

}

## Architecture Benefits

* Testability: Each adapter can be tested independently
* Flexibility: Easy to swap adapters (e.g., change from DynamoDB to RDS)
* Separation of Concerns: Business logic isolated from infrastructure
* Scalability: Each component can scale independently

## Cost Optimization (Free Tier)

* Lambda: 1M free requests/month
* API Gateway: 1M API calls/month
* DynamoDB: 25GB storage + 25 RCU/WCU
* SQS: 1M requests/month

## Monitoring & Logging

1. CloudWatch Logs: Monitor Lambda execution
2. CloudWatch Metrics: Track API Gateway and DynamoDB usage.3.AWS X-Ray: Trace requests across service  
     
   

