**AWS ASSESSMENT**

*by Shakil Ahamed R (2116020)*

Contents

[PROBLEM STATEMENT 1](#_Toc164412576)

[SYSTEM ARCHITECTURE 1](#_Toc164412577)

[DESIGN DECISIONS 2](#_Toc164412578)

[IMPLEMENTATION DETAILS 2](#_Toc164412579)

[RESULT 3](#_Toc164412580)

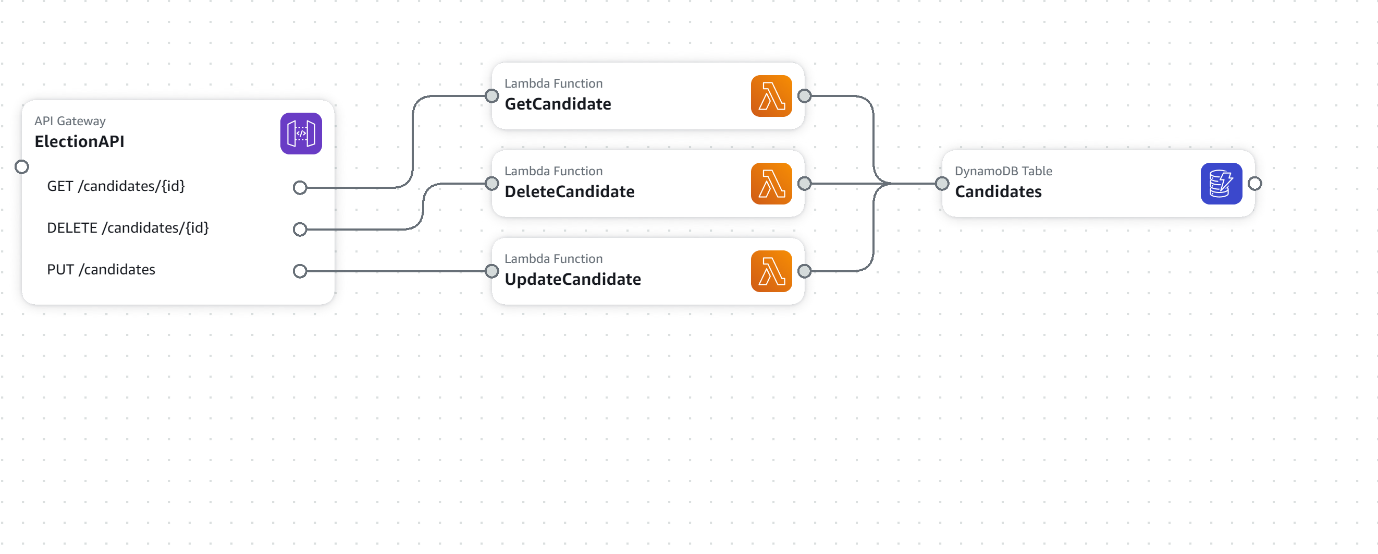
[AWS CLOUDFORMATION TEMPLATE 7](#_Toc164412581)

[CODE SAMPLES 10](#_Toc164412582)

# PROBLEM STATEMENT

The project aims to create a real-time data processing system by integrating Amazon DynamoDB with AWS Lambda using Java programming language. DynamoDB is a fully managed NoSQL database service provided by AWS, while AWS Lambda is a serverless compute service that allows running code without provisioning or managing servers. By combining these services, the project will demonstrate how to efficiently process, store, and retrieve data in real-time with minimal operational overhead.

# SYSTEM ARCHITECTURE



The project uses the below AWS services to create a secure, scalable, reliable, cost efficient application.

1. AWS DynamoDB
2. AWS Lambda
3. AWS API Gateway

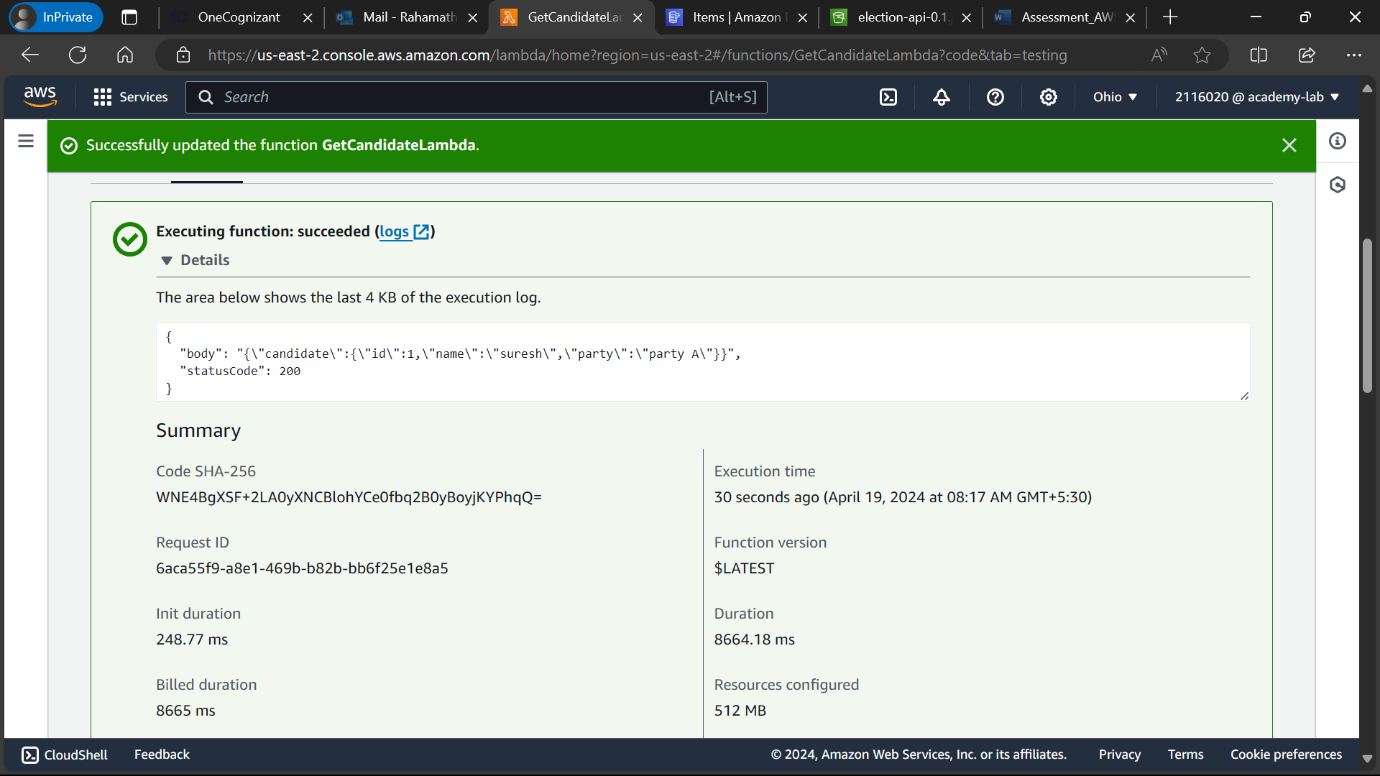
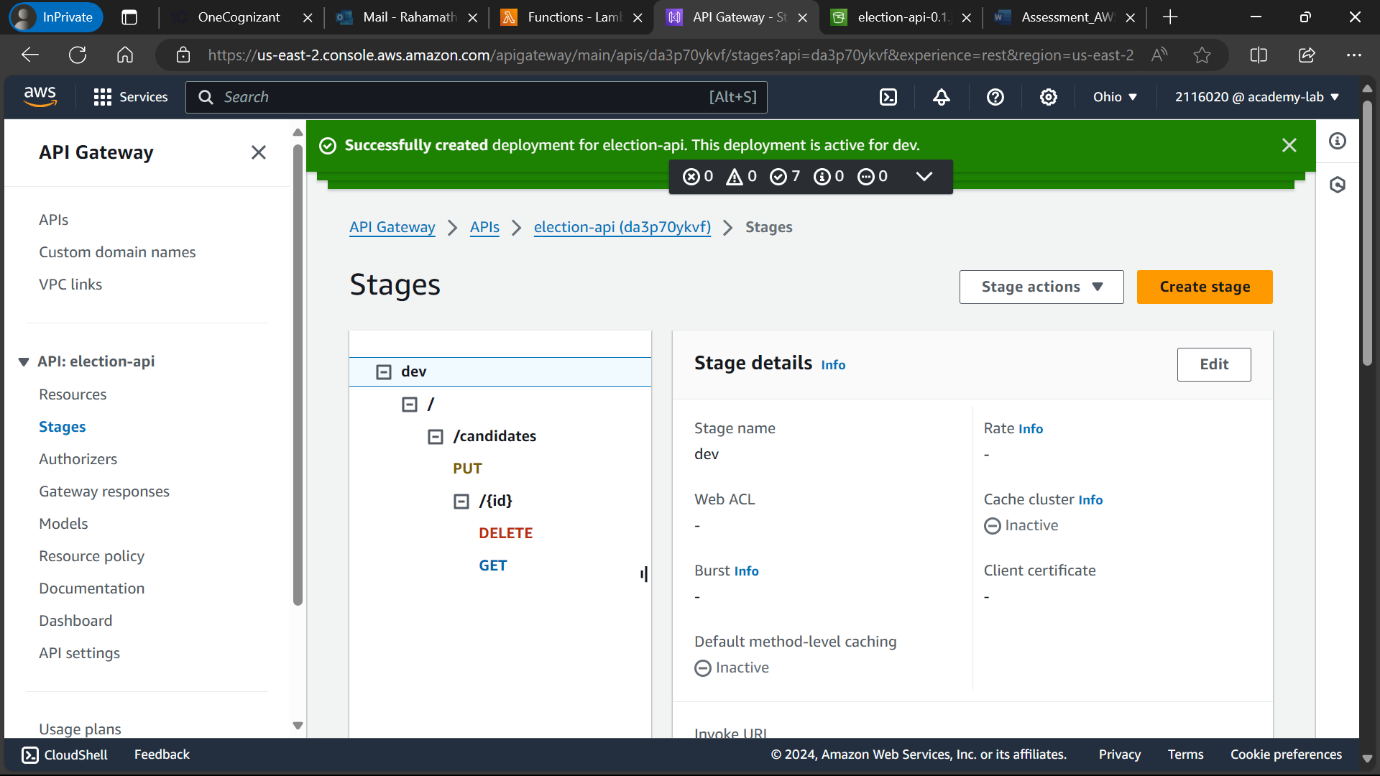
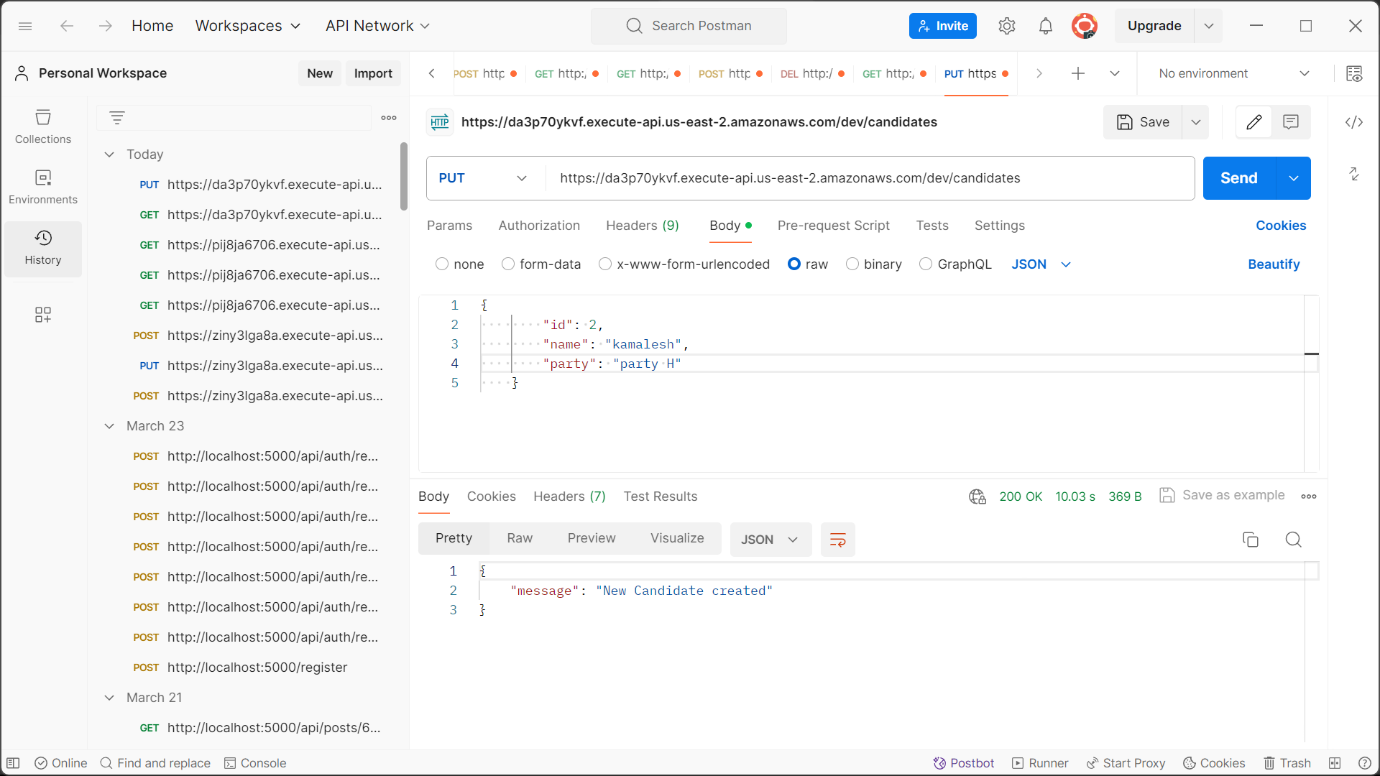
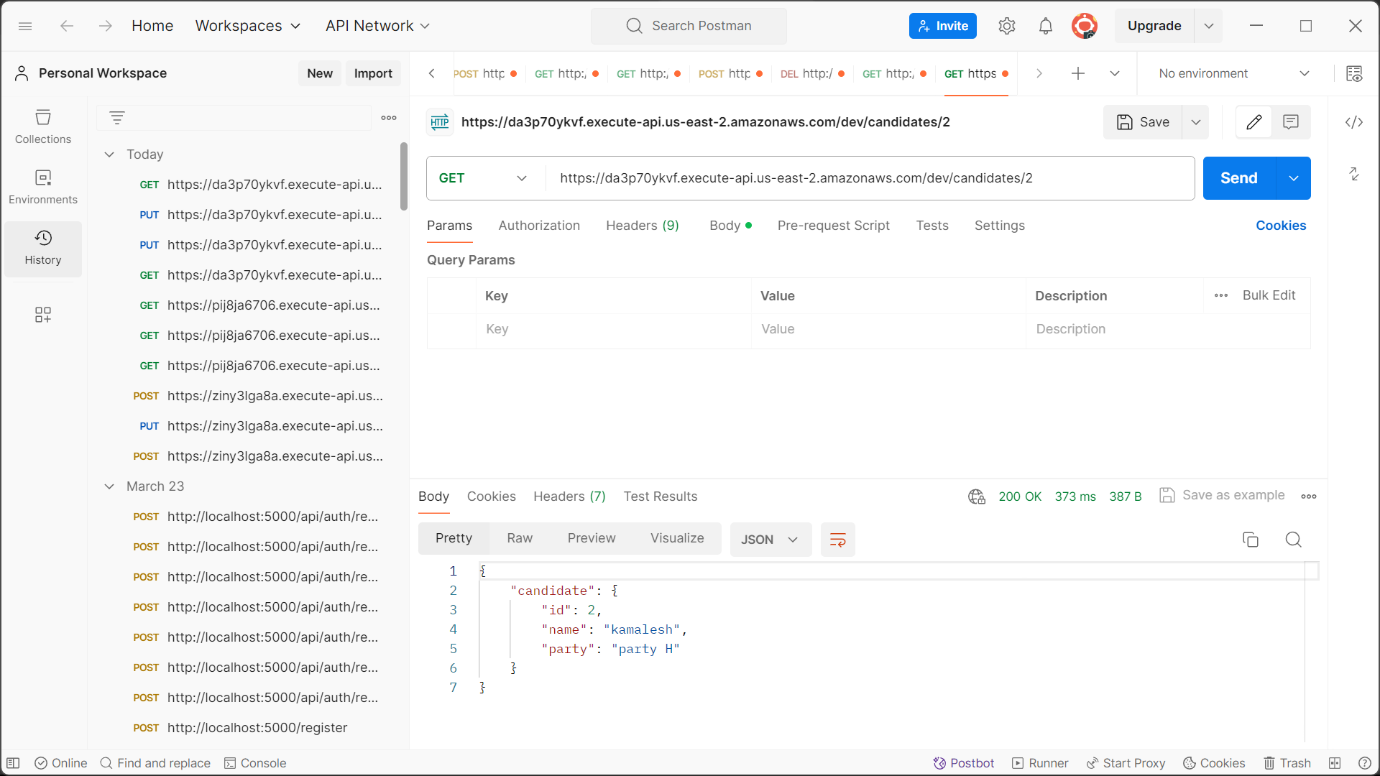
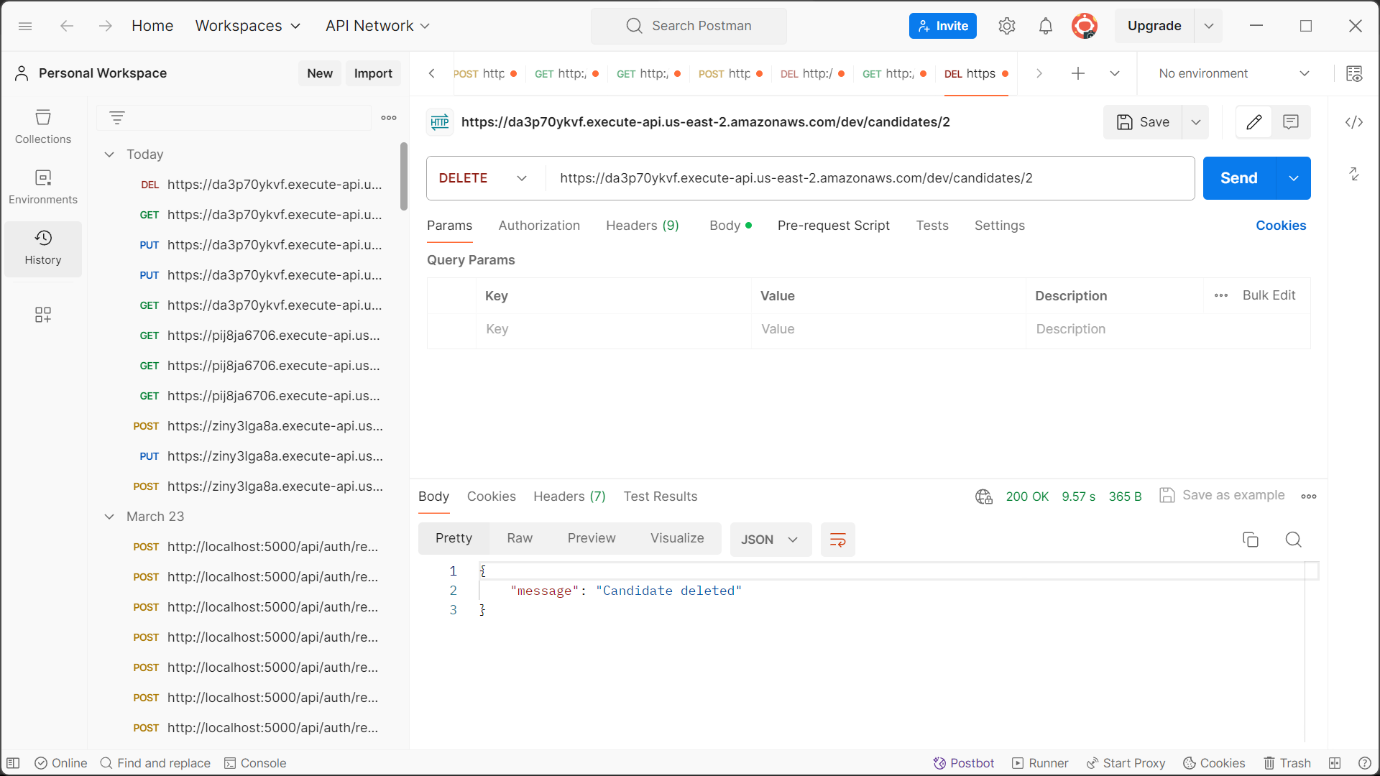
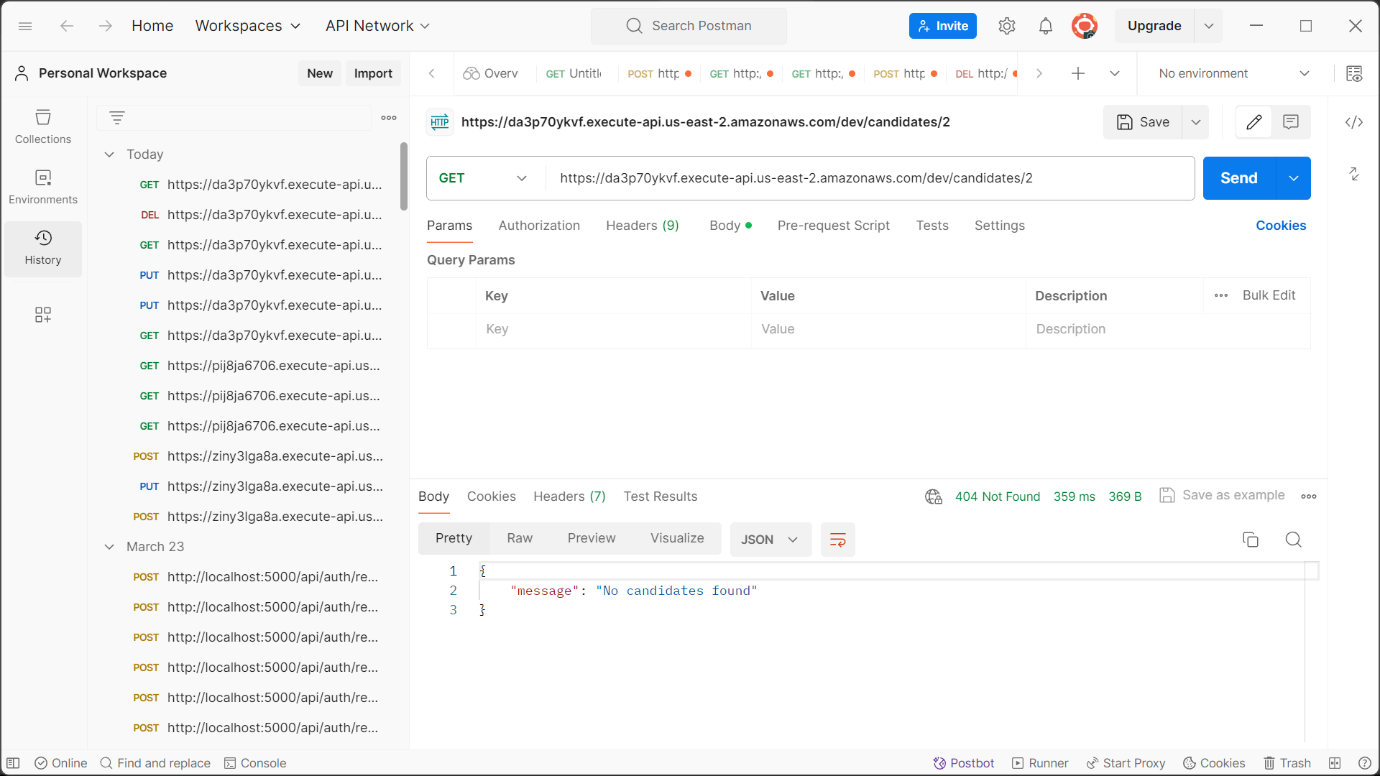
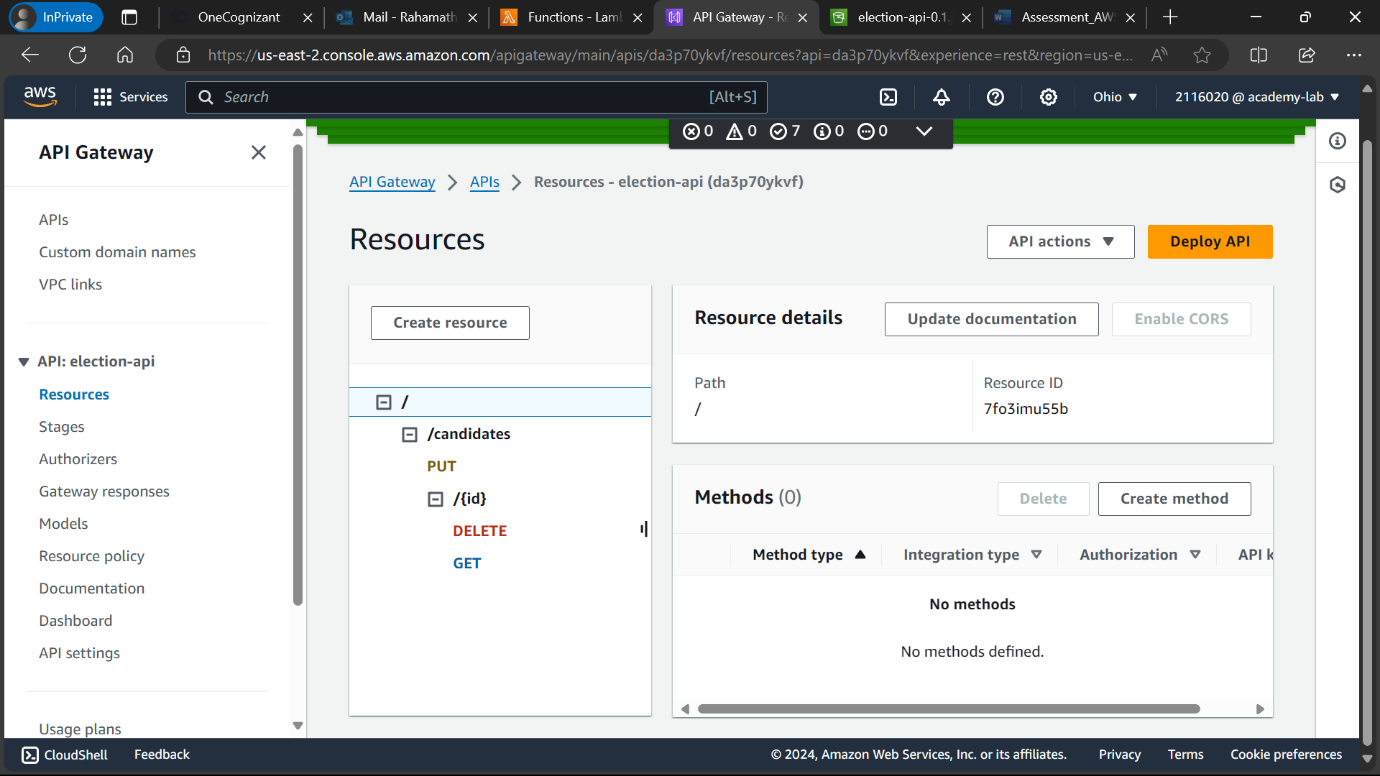
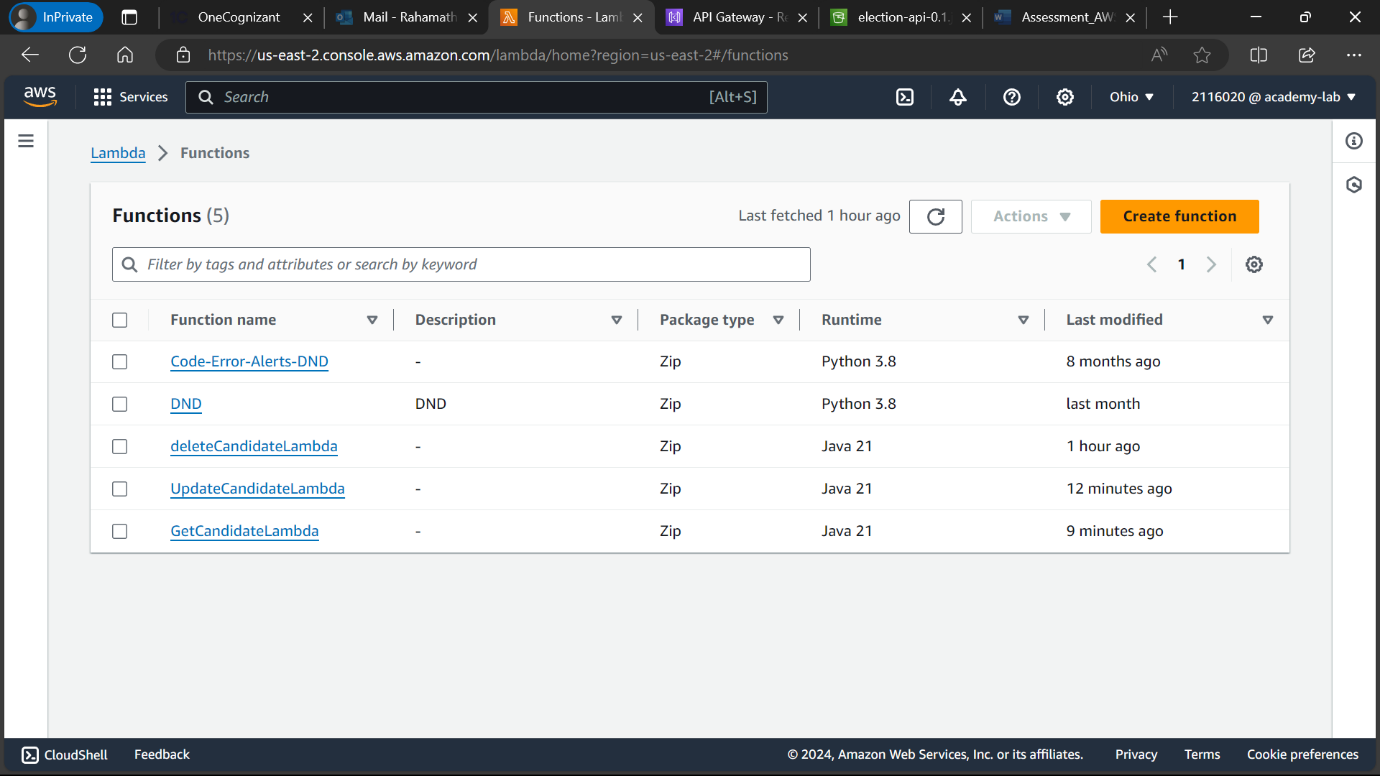
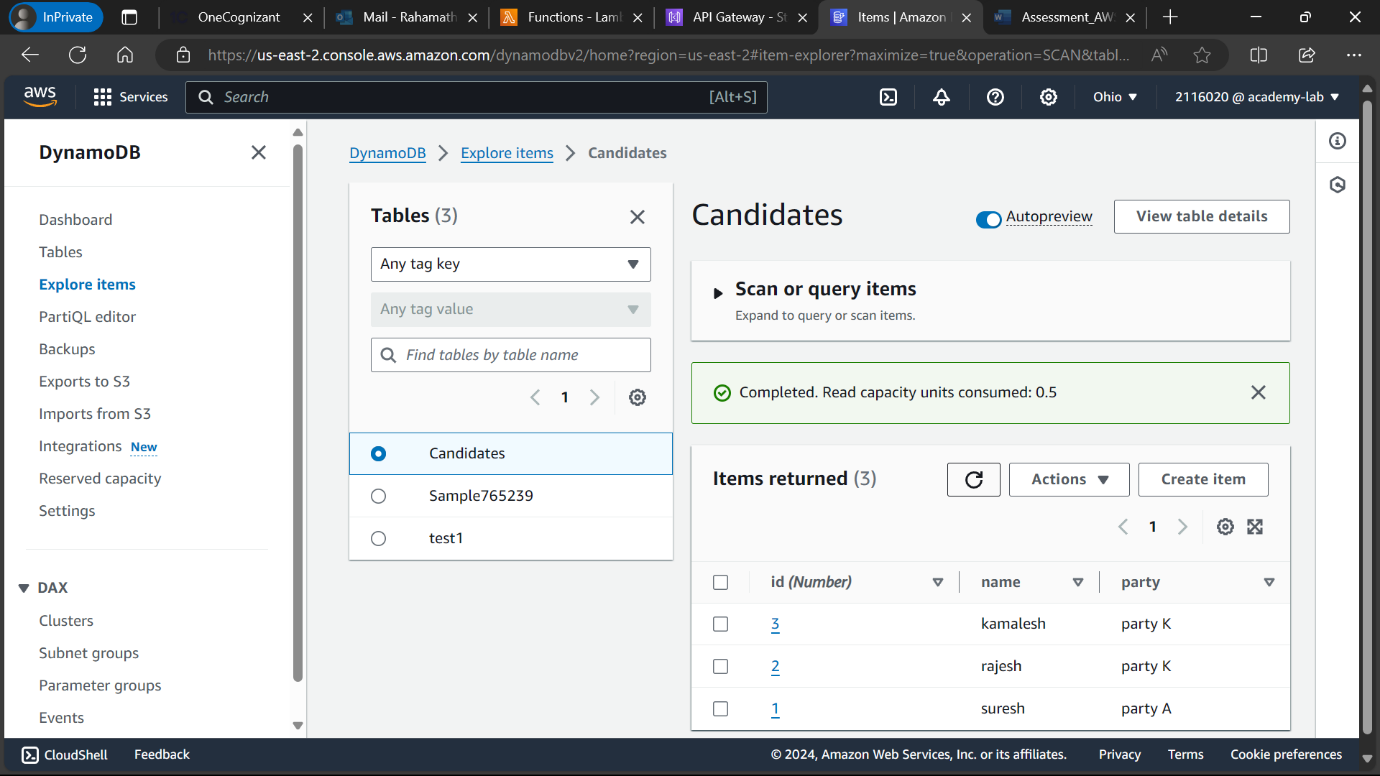
# DESIGN DECISIONS

* To reduce the latency and also to keep the cost in check, I used DynamoDB as a Database to store real-time data and AWS Lambda functions will be triggered whenever data is ingested into DynamoDB.
* The lambda functions are strategically divided into 3 separate to keep the developer experience high and to make sure only the required code runs thus reducing cost
* To make sure the application is always available even during peak loads, but at the same time to keep the aws cost in mind, I used DynamoDB in provisioned mode with autoscaling option thus serving our application purpose fully.
* RESTful API are created using AWS API Gateway to allow accessing, modifying, updating and deleting candidates’ data in DynamoDB.
* AWS Java SDK is used to for creating, updating and deleting candidates data in DynamoDB which makes the application reliable and secure
* To keep the candidates data secure, the data in DynamoDB is encrypted at rest, thus making this solution secure

# IMPLEMENTATION DETAILS

1. Create a DynamoDB table named Candidates with id as partition key
2. Add an item in the table
3. Create a Maven project in an IDE and import AWS JAVA Sdk in pom.xml
4. Create a model class named Candidates.java with getters and setters method
5. Create a new class named CandidatesLambdaHandler.java
6. Create a new method named handleRequest() in it and write code to do CRUD operations in DynamoDB using AWS java sdk
7. Build the code and create a JAR package
8. Import the JAR package in AWS lambda and make sure to change the Handler function in AWS console.
9. Test the Lambda function in AWS console to make sure it works as expected
10. Now create an API in AWS Gateway named Election-API
11. Create a new resource called Candidates and then create a new GET method.
12. Make sure to select AWS Lambda function before created GET method.
13. Now repeat step 6 to 12 for other lambda functions too
14. Now we have a application that processes data that is secure, reliable, scalable and cost efficient.

# RESULT



# AWS CLOUDFORMATION TEMPLATE

Transform: AWS::Serverless-2016-10-31

Resources:

  ElectionAPI:

    Type: AWS::Serverless::Api

    Properties:

      Name: !Sub

        - ${ResourceName} From Stack ${AWS::StackName}

        - ResourceName: ElectionAPI

      StageName: Prod

      DefinitionBody:

        openapi: '3.0'

        info: {}

        paths:

          /candidates/{id}:

            get:

              x-amazon-apigateway-integration:

                httpMethod: POST

                type: aws\_proxy

                uri: !Sub arn:${AWS::Partition}:apigateway:${AWS::Region}:lambda:path/2015-03-31/functions/${GetCandidate.Arn}/invocations

              responses: {}

            delete:

              x-amazon-apigateway-integration:

                httpMethod: POST

                type: aws\_proxy

                uri: !Sub arn:${AWS::Partition}:apigateway:${AWS::Region}:lambda:path/2015-03-31/functions/${DeleteCandidate.Arn}/invocations

              responses: {}

          /candidates:

            put:

              x-amazon-apigateway-integration:

                httpMethod: POST

                type: aws\_proxy

                uri: !Sub arn:${AWS::Partition}:apigateway:${AWS::Region}:lambda:path/2015-03-31/functions/${UpdateCandidate.Arn}/invocations

              responses: {}

      EndpointConfiguration: REGIONAL

      TracingEnabled: true

      Cors:

        MaxAge: 5

  GetCandidate:

    Type: AWS::Serverless::Function

    Properties:

      Description: !Sub

        - Stack ${AWS::StackName} Function ${ResourceName}

        - ResourceName: GetCandidate

      CodeUri: src/Function

      Handler: com.shakil.aws.CandidateLambdaHandler::handleRequest

      Runtime: java21

      MemorySize: 3008

      Timeout: 30

      Tracing: Active

      Events:

        ElectionAPIGETcandidatesid:

          Type: Api

          Properties:

            Path: /candidates/{id}

            Method: GET

            RestApiId: !Ref ElectionAPI

      Environment:

        Variables:

          CANDIDATES\_TABLE\_NAME: !Ref Candidates

          CANDIDATES\_TABLE\_ARN: !GetAtt Candidates.Arn

      Policies:

        - DynamoDBCrudPolicy:

            TableName: !Ref Candidates

  GetCandidateLogGroup:

    Type: AWS::Logs::LogGroup

    DeletionPolicy: Retain

    Properties:

      LogGroupName: !Sub /aws/lambda/${GetCandidate}

  UpdateCandidate:

    Type: AWS::Serverless::Function

    Properties:

      Description: !Sub

        - Stack ${AWS::StackName} Function ${ResourceName}

        - ResourceName: UpdateCandidate

      CodeUri: src/Function2

      Handler: com.shakil.aws.CandidateLambdaHandler::handlePutRequest

      Runtime: java21

      MemorySize: 3008

      Timeout: 30

      Tracing: Active

      Events:

        ElectionAPIPUTcandidates:

          Type: Api

          Properties:

            Path: /candidates

            Method: PUT

            RestApiId: !Ref ElectionAPI

      Environment:

        Variables:

          CANDIDATES\_TABLE\_NAME: !Ref Candidates

          CANDIDATES\_TABLE\_ARN: !GetAtt Candidates.Arn

      Policies:

        - DynamoDBCrudPolicy:

            TableName: !Ref Candidates

  UpdateCandidateLogGroup:

    Type: AWS::Logs::LogGroup

    DeletionPolicy: Retain

    Properties:

      LogGroupName: !Sub /aws/lambda/${UpdateCandidate}

  DeleteCandidate:

    Type: AWS::Serverless::Function

    Properties:

      Description: !Sub

        - Stack ${AWS::StackName} Function ${ResourceName}

        - ResourceName: DeleteCandidate

      CodeUri: src/Function3

      Handler: com.shakil.aws.CandidateLambdaHandler::handleDeleteRequest

      Runtime: java21

      MemorySize: 3008

      Timeout: 30

      Tracing: Active

      Events:

        ElectionAPIDELETEcandidatesid:

          Type: Api

          Properties:

            Path: /candidates/{id}

            Method: DELETE

            RestApiId: !Ref ElectionAPI

      Environment:

        Variables:

          CANDIDATES\_TABLE\_NAME: !Ref Candidates

          CANDIDATES\_TABLE\_ARN: !GetAtt Candidates.Arn

      Policies:

        - DynamoDBCrudPolicy:

            TableName: !Ref Candidates

  DeleteCandidateLogGroup:

    Type: AWS::Logs::LogGroup

    DeletionPolicy: Retain

    Properties:

      LogGroupName: !Sub /aws/lambda/${DeleteCandidate}

  Candidates:

    Type: AWS::DynamoDB::Table

    Properties:

      AttributeDefinitions:

        - AttributeName: id

          AttributeType: 'N'

      BillingMode: PAY\_PER\_REQUEST

      KeySchema:

        - AttributeName: id

          KeyType: HASH

      StreamSpecification:

        StreamViewType: NEW\_AND\_OLD\_IMAGES

# CODE SAMPLES

**Project Repo:** [GitHub - shakilahamedr/election-api](https://github.com/shakilahamedr/election-api)

**package** com.shakil.aws;

**import** java.io.BufferedReader;

**import** java.io.IOException;

**import** java.io.InputStream;

**import** java.io.InputStreamReader;

**import** java.io.OutputStream;

**import** java.io.OutputStreamWriter;

**import** org.json.simple.JSONObject;

**import** org.json.simple.parser.JSONParser;

**import** com.amazonaws.services.dynamodbv2.AmazonDynamoDB;

**import** com.amazonaws.services.dynamodbv2.AmazonDynamoDBClientBuilder;

**import** com.amazonaws.services.dynamodbv2.document.DynamoDB;

**import** com.amazonaws.services.dynamodbv2.document.Item;

**import** com.amazonaws.services.dynamodbv2.document.spec.PutItemSpec;

**import** com.amazonaws.services.lambda.runtime.Context;

**import** com.amazonaws.services.lambda.runtime.RequestStreamHandler;

**import** com.shakil.aws.model.Candidate;

**public** **class** CandidateLambdaHandler **implements** RequestStreamHandler {

**private** String DYNAMO\_TABLE = "Candidates";

@SuppressWarnings("unchecked")

@Override

**public** **void** handleRequest(InputStream input, OutputStream output, Context context) **throws** IOException {

OutputStreamWriter writer = **new** OutputStreamWriter(output);

BufferedReader reader = **new** BufferedReader(**new** InputStreamReader(input));

JSONParser parser = **new** JSONParser();

JSONObject responseObject = **new** JSONObject();

JSONObject responseBody = **new** JSONObject();

AmazonDynamoDB client = AmazonDynamoDBClientBuilder.*defaultClient*();

DynamoDB dynamoDB = **new** DynamoDB(client);

**int** id;

Item resItem = **null**;

**try** {

JSONObject reqObject = (JSONObject) parser.parse(reader);

**if** (reqObject.get("pathParameters") != **null**) {

JSONObject pps = (JSONObject) reqObject.get("pathParameters");

**if** (pps.get("id") != **null**) {

id = Integer.*parseInt*((String) pps.get("id"));

resItem = dynamoDB.getTable(DYNAMO\_TABLE).getItem("id", id);

}

} **else** **if** (reqObject.get("queryStringParameters") != **null**) {

JSONObject qps = (JSONObject) reqObject.get("queryStringParameters");

**if** (qps.get("id") != **null**) {

id = Integer.*parseInt*((String) qps.get("id"));

resItem = dynamoDB.getTable(DYNAMO\_TABLE).getItem("id", id);

}

}

**if** (resItem != **null**) {

Candidate candidate = **new** Candidate(resItem.toJSON());

responseBody.put("candidate", candidate);

responseObject.put("statusCode", 200);

} **else** {

responseBody.put("message", "No candidates found");

responseObject.put("statusCode", 404);

}

responseObject.put("body", responseBody.toString());

} **catch** (Exception e) {

context.getLogger().log("ERROR : " + e.getMessage());

}

writer.write(responseObject.toString());

reader.close();

writer.close();

}

@SuppressWarnings("unchecked")

**public** **void** handlePutRequest(InputStream input, OutputStream output, Context context) **throws** IOException {

OutputStreamWriter writer = **new** OutputStreamWriter(output);

BufferedReader reader = **new** BufferedReader(**new** InputStreamReader(input));

JSONParser parser = **new** JSONParser();

JSONObject responseObject = **new** JSONObject();

JSONObject responseBody = **new** JSONObject();

AmazonDynamoDB client = AmazonDynamoDBClientBuilder.*defaultClient*();

DynamoDB dynamoDB = **new** DynamoDB(client);

**try** {

JSONObject reqObject = (JSONObject) parser.parse(reader);

**if** (reqObject.get("body") != **null**) {

Candidate candidate = **new** Candidate((String) reqObject.get("body"));

dynamoDB.getTable(DYNAMO\_TABLE)

.putItem(**new** PutItemSpec().withItem(**new** Item().withNumber("id", candidate.getId())

.withString("name", candidate.getName()).withString("party", candidate.getParty())));

responseBody.put("message", "New Candidate created");

responseObject.put("statusCode", 200);

responseObject.put("body", responseBody.toString());

}

} **catch** (Exception e) {

responseObject.put("statusCode", 400);

responseObject.put("error", e);

}

writer.write(responseObject.toString());

reader.close();

writer.close();

}

@SuppressWarnings("unchecked")

**public** **void** handleDeleteRequest(InputStream input, OutputStream output, Context context) **throws** IOException {

OutputStreamWriter writer = **new** OutputStreamWriter(output);

BufferedReader reader = **new** BufferedReader(**new** InputStreamReader(input));

JSONParser parser = **new** JSONParser();

JSONObject responseObject = **new** JSONObject();

JSONObject responseBody = **new** JSONObject();

AmazonDynamoDB client = AmazonDynamoDBClientBuilder.*defaultClient*();

DynamoDB dynamoDB = **new** DynamoDB(client);

**try** {

JSONObject reqObject = (JSONObject) parser.parse(reader);

**if** (reqObject.get("pathParameters") != **null**) {

JSONObject pps = (JSONObject) reqObject.get("pathParameters");

**if** (pps.get("id") != **null**) {

**int** id = Integer.*parseInt*((String) pps.get("id"));

dynamoDB.getTable(DYNAMO\_TABLE).deleteItem("id", id);

}

}

responseBody.put("message", "Candidate deleted");

responseObject.put("statusCode", 200);

responseObject.put("body", responseBody.toString());

} **catch** (Exception e) {

responseObject.put("statusCode", 400);

responseObject.put("error", e);

}

writer.write(responseObject.toString());

reader.close();

writer.close();

}

}