# Serverless App Development with AWS

### **EXERCISE 1 – Build a CRUD API with Lambda and DynamoDB**

#### **Introduction**

AWS Lambda is a serverless compute service that allows you to run code without provisioning or managing servers. It executes your code only when triggered, scaling automatically from a few requests per day to thousands per second. You are charged only for the compute time used—there’s no cost when the code isn’t running.

Lambda enables you to build backend services or applications without any administrative overhead. It operates on a high-availability compute infrastructure, handling tasks like server maintenance, operating system updates, capacity provisioning, scaling, monitoring, and logging. All you need is to supply your code in a supported language.

**Key Features:**

* **Event-driven execution:** Trigger Lambda functions using events like data changes in Amazon S3 or DynamoDB, HTTP requests via API Gateway, or custom API calls using AWS SDKs.
* **Integration:** Build workflows for AWS services, process streaming data from Amazon Kinesis, or create backend services with AWS-level performance, scalability, and security.

#### **Amazon DynamoDB**

Amazon DynamoDB is a fully managed NoSQL database designed for fast, predictable performance and seamless scalability. It offloads the administrative tasks of managing distributed databases, such as hardware provisioning, setup, replication, software patching, and scaling.

**Key Benefits:**

* **Operational efficiency:** No need to manage infrastructure.
* **Security:** Encryption at rest ensures sensitive data is protected.
* **Performance:** Consistently fast read and write operations.

#### **Amazon API Gateway**

Amazon API Gateway is a fully managed service for creating, deploying, and managing REST, HTTP, and WebSocket APIs at any scale. It allows developers to integrate APIs with AWS services or other web resources.

**API Gateway Features:**

* Supports **stateless client-server communication** using standard HTTP methods like GET, POST, PUT, PATCH, and DELETE.
* Provides secure, scalable, and monitorable API endpoints.
* Enables API integration with backend services like Lambda and DynamoDB.

### **Objective**

The goal of this exercise is to learn the foundational components of serverless application development, particularly backend integration.

**Learning Outcomes:**

* Understand the basics of AWS Lambda.
* Learn to create and use API Gateway for managing HTTP APIs.
* Integrate Lambda with API Gateway to interact with DynamoDB.
* Build a serverless CRUD API to create, read, update, and delete items from a DynamoDB table.

### **Steps to Perform**

1. **Create a DynamoDB Table**
   * Use the DynamoDB console to define a table with appropriate attributes and a primary key.
2. **Create a Lambda Function**
   * Use the AWS Lambda console to define a function that interacts with the DynamoDB table to handle CRUD operations.
3. **Set Up an API Gateway**
   * Use the API Gateway console to create an HTTP API that routes client requests to the Lambda function.
4. **Test the API**
   * Invoke the HTTP API from a client application. API Gateway will route the request to the Lambda function, which will process the operation on DynamoDB and return a response.

### **Process Overview**

1. **Client Request:**  
   A client application sends an HTTP request to the API Gateway endpoint.
2. **Request Routing:**  
   API Gateway receives the request and routes it to the designated AWS Lambda function.
3. **Lambda Execution:**  
   The Lambda function processes the request by performing the required CRUD operation on the DynamoDB table.
4. **Response Generation:**  
   After completing the operation, the Lambda function sends a response back to API Gateway.
5. **Client Response:**  
   API Gateway formats and returns the response to the client application.

# 

# 

### **Step 1: Create a DynamoDB Table**

In this step, you’ll create a DynamoDB table to store data for your API. Each item in the table will have a unique **ID**, which will serve as the **partition key**.

#### **Steps to Create the DynamoDB Table:**

1. **Access DynamoDB Console:**  
   Open the DynamoDB console at <https://console.aws.amazon.com/dynamodb/>.
2. **Create a New Table:**
   * Click **Create table** on the console.
3. **Configure Table Settings:**
   * For **Table name**, enter: http-crud-tutorial-items.
   * For **Primary key**, enter: id.
4. **Finalize Creation:**
   * Click **Create** to create the table.

Your DynamoDB table is now ready to store data for your CRUD API operations. The unique id will ensure that each record is uniquely identifiable within the table.

### **Step 2: Create a Lambda Function**

In this step, you'll create a Lambda function to serve as the backend for your API. This function will handle creating, reading, updating, and deleting items from the DynamoDB table. The Lambda function uses events from API Gateway to determine the desired CRUD operation.

**Note:** While this lab uses a single function for simplicity, the best practice is to create separate Lambda functions for each API route to improve maintainability and scalability.

#### **Steps to Create the Lambda Function:**

1. **Access Lambda Console:**  
   Sign in to the Lambda console at <https://console.aws.amazon.com/lambda>.
2. **Create a New Function:**
   * Click **Create function**.
3. **Configure Function Settings:**
   * For **Function name**, enter: http-crud-tutorial-function.
   * Under **Permissions**, choose **Change default execution role**.
4. **Set Execution Role:**
   * Select **Create a new role from AWS policy templates**.
   * For **Role name**, enter: http-crud-tutorial-role.
   * For **Policy templates**, select **Simple microservice permissions** to grant the Lambda function permission to interact with DynamoDB.
5. **Finalize Function Creation:**
   * Click **Create function** to generate the Lambda function.
6. **Add Function Code:**
   * Open the **index.js** file in the Lambda console's code editor.
   * Replace its contents with the provided code snippet (specific to the CRUD operations).
   * Click **Deploy** to save and activate your function.

Your Lambda function is now ready and configured to handle requests from API Gateway and perform CRUD operations on the DynamoDB table.

**Nodejs18X**import { DynamoDBClient } from "@aws-sdk/client-dynamodb";

import pkg from "@aws-sdk/lib-dynamodb";

const { PutCommand, DeleteCommand, GetCommand, ScanCommand } = pkg;

const client = new DynamoDBClient();

export const handler = async (event) => {

  let body;

  let statusCode = 200;

  const headers = {

    "Content-Type": "application/json",

  };

  try {

    switch (event.routeKey) {

      case "DELETE /items/{id}":

        await client.send(

          new DeleteCommand({

            TableName: "http-crud-tutorial-items",

            Key: { id: event.pathParameters.id },

          })

        );

        body = `Deleted item ${event.pathParameters.id}`;

        break;

      case "GET /items/{id}":

        body = await client.send(

          new GetCommand({

            TableName: "http-crud-tutorial-items",

            Key: { id: event.pathParameters.id },

          })

        );

        break;

      case "GET /items":

        body = await client.send(

          new ScanCommand({ TableName: "http-crud-tutorial-items" })

        );

        break;

      case "PUT /items":

        const requestJSON = JSON.parse(event.body);

        await client.send(

          new PutCommand({

            TableName: "http-crud-tutorial-items",

            Item: {

              id: requestJSON.id,

              price: requestJSON.price,

              name: requestJSON.name,

            },

          })

        );

        body = `Put item ${requestJSON.id}`;

        break;

      default:

        throw new Error(`Unsupported route: "${event.routeKey}"`);

    }

  } catch (err) {

    statusCode = 400;

    body = err.message;

  }

  return {

    statusCode,

    body: JSON.stringify(body),

    headers,

  };

};

### **Step 3: Create an HTTP API**

In this step, you’ll create an HTTP API using Amazon API Gateway. The API provides an endpoint to connect your Lambda function with clients. Initially, you’ll create an empty API, and routes will be configured later to define how requests are handled.

#### **Steps to Create the HTTP API:**

1. **Access API Gateway Console:**  
   Sign in to the API Gateway console at <https://console.aws.amazon.com/apigateway>.
2. **Create a New API:**
   * Choose **Create API**.
   * Under **HTTP API**, select **Build**.
3. **Configure the API:**
   * For **API name**, enter: http-crud-tutorial-api.
   * Click **Next**.
4. **Skip Route Creation:**
   * Under **Configure routes**, click **Next** to skip this step. Routes will be added later.
5. **Review and Finalize:**
   * Review the **Stage** created by API Gateway, and click **Next**.
   * Click **Create** to finalize your API setup.

Your HTTP API is now created and ready for route configuration.

### **Step 4: Create Routes**

Routes define how incoming HTTP requests are processed and forwarded to backend resources. In this step, you’ll create routes for CRUD operations using specific HTTP methods and resource paths.

#### **Routes to Create:**

* **GET /items/{id}**: Retrieve an item by ID.
* **GET /items**: Retrieve all items.
* **PUT /items**: Add or update an item.
* **DELETE /items/{id}**: Delete an item by ID.

#### **Steps to Create Routes:**

1. **Access Your API:**
   * In the API Gateway console, select your API (e.g., http-crud-tutorial-api).
2. **Open Routes:**
   * Click **Routes** in the left menu.
3. **Add a Route:**
   * Click **Create**.
   * For **Method**, choose: GET.
   * For **Path**, enter: /items/{id}.
     + The {id} at the end is a **path parameter** that API Gateway extracts from the client’s request.
   * Click **Create**.
4. **Repeat for Other Routes:**
   * Repeat steps 3–4 to create the following routes:
     + **GET /items**
     + **PUT /items**
     + **DELETE /items/{id}**

Your API now has routes configured to handle requests and forward them to your Lambda function. These routes enable the HTTP API to perform CRUD operations on the DynamoDB table effectively.

### **Step 5: Create an Integration**

Integrations link API routes to backend resources, such as Lambda functions. In this step, you'll create a single Lambda integration and use it across all routes.

#### **Steps to Create the Integration:**

1. **Access Your API in API Gateway:**
   * Sign in to the API Gateway console at <https://console.aws.amazon.com/apigateway>.
   * Select your API (e.g., http-crud-tutorial-api).
2. **Navigate to Integrations:**
   * In the left-hand menu, select **Integrations**.
3. **Create the Integration:**
   * Click **Manage integrations** and then select **Create**.
4. **Skip Route Attachment:**
   * For **Attach this integration to a route**, choose **Skip**.
     + You’ll attach the integration to routes in a later step.
5. **Specify Integration Details:**
   * For **Integration type**, choose: **Lambda function**.
   * For **Lambda function**, enter: http-crud-tutorial-function.
6. **Finalize the Integration:**
   * Click **Create** to save your integration.

You’ve successfully created a Lambda integration. Next, you'll attach it to the routes.

### **Step 6: Attach Your Integration to Routes**

To make your API functional, the integration must be attached to each route. In this step, you’ll attach the previously created Lambda integration to all four routes:

* **GET /items/{id}**
* **GET /items**
* **PUT /items**
* **DELETE /items/{id}**

#### **Steps to Attach the Integration to Routes:**

1. **Access Integrations:**
   * In the API Gateway console, select **Integrations** under your API.
2. **Attach to a Route:**
   * Select a route (e.g., **GET /items/{id}**).
   * Under **Choose an existing integration**, select: http-crud-tutorial-function.
   * Click **Attach integration**.
3. **Repeat for All Routes:**
   * Repeat Step 2 for the remaining routes (**GET /items**, **PUT /items**, **DELETE /items/{id}**).

#### **Verification:**

Once all routes have been updated, verify that each route displays the message:  
**AWS Lambda integration is attached.**

### **Step 7: Test Your API**

Testing your API ensures that the routes, integrations, and backend functionality are working correctly. You can use curl, a command-line tool, to send HTTP requests to your API.

#### **Step 7.1: Retrieve Your API’s Invoke URL**

1. **Log in to API Gateway Console:**
   * Go to <https://console.aws.amazon.com/apigateway>.
2. **Select Your API:**
   * From the list of APIs, choose your API (e.g., http-crud-tutorial-api).
3. **Locate the Invoke URL:**
   * On the **Details** page, look for the **Invoke URL** under the **API Overview** section.  
     Example: https://abcdef123.execute-api.us-west-2.amazonaws.com.
4. **Copy the Invoke URL:**
   * Save the URL for use in subsequent curl commands.

#### **Step 7.2: Create or Update an Item**

To test the **PUT /items** route, use the following command:

curl -v -X "PUT" -H "Content-Type: application/json" \

-d "{\"id\": \"abcdef234\", \"price\": 12345, \"name\": \"myitem\"}" \

https://abcdef123.execute-api.us-west-2.amazonaws.com/items

* **Command Breakdown:**
  + -v: Enables verbose output for debugging.
  + -X "PUT": Specifies the HTTP method (PUT).
  + -H "Content-Type: application/json": Sets the request header to indicate JSON data.
  + -d "{\"id\": \"abcdef234\", \"price\": 12345, \"name\": \"myitem\"}": Specifies the JSON body containing item details.
  + Replace https://abcdef123.execute-api.us-west-2.amazonaws.com/items with your API’s **Invoke URL** + /items.

#### **Expected Output:**

* **HTTP Response Code:**
  + 200 OK or similar if the request succeeds.
* **Response Body:**
  + JSON confirmation of the item's creation or update.

Example:

{

"id": "abcdef234",

"price": 12345,

"name": "myitem"

}

### **Next Steps:**

1. **Test Other Routes:**
   * Use similar curl commands for the other routes (**GET**, **DELETE**) to validate the complete API functionality.
2. **Monitor Logs:**
   * Use AWS CloudWatch to inspect Lambda execution logs for troubleshooting if any issues arise.
3. **Iterate and Improve:**
   * Based on the test results, make necessary adjustments to your Lambda function or API Gateway configurations.

# 

### **Step-by-Step Guide for Completing the Lab**

### **Step 7.3: Additional API Commands**

Use the following commands to test other HTTP methods for your API.

#### **Get All Items**

To list all items in the DynamoDB table:

curl -v https://abcdef123.execute-api.us-west-2.amazonaws.com/items

#### **Get a Specific Item**

To retrieve an item by its id:

curl -v https://abcdef123.execute-api.us-west-2.amazonaws.com/items/abcdef234

#### **Delete an Item**

To delete an item by its id:

curl -v -X "DELETE" https://abcdef123.execute-api.us-west-2.amazonaws.com/items/abcdef234

#### **Verify Deletion**

To ensure the item is deleted, list all items again:

curl -v https://abcdef123.execute-api.us-west-2.amazonaws.com/items

### **Step 8: Clean Up Resources**

#### **Delete the DynamoDB Table**

1. Open the DynamoDB console: [AWS DynamoDB Console](https://console.aws.amazon.com/dynamodb/).
2. Select your table (http-crud-tutorial-items).
3. Choose **Delete table**.
4. Confirm deletion by clicking **Delete**.

#### **Delete the HTTP API**

1. Open the API Gateway console: [AWS API Gateway Console](https://console.aws.amazon.com/apigateway/).
2. Select your API (http-crud-tutorial-api).
3. Click **Actions** > **Delete**.
4. Confirm deletion.

#### **Delete the Lambda Function**

1. Open the Lambda console: [AWS Lambda Console](https://console.aws.amazon.com/lambda/).
2. Select your function (http-crud-tutorial-function).
3. Click **Actions** > **Delete**.
4. Confirm deletion.

#### **Delete the Lambda Log Group**

1. Open the Amazon CloudWatch console: [AWS CloudWatch Console](https://console.aws.amazon.com/cloudwatch/).
2. Navigate to **Log groups**.
3. Find and select /aws/lambda/http-crud-tutorial-function.
4. Click **Actions** > **Delete log group**.
5. Confirm deletion.

#### **Delete the Lambda Execution Role**

1. Open the IAM console: [AWS IAM Console](https://console.aws.amazon.com/iam/).
2. Navigate to **Roles**.
3. Select the role (http-crud-tutorial-role).
4. Click **Delete role**.
5. Confirm deletion.

### **Results**

Students will be able to:

* Define an AWS Lambda function.
* Create and configure a DynamoDB table.
* Develop a serverless API triggered by HTTP requests from the client side.
* Handle API calls to perform CRUD operations on a database.
* Implement basic HTTP methods like GET, PUT, and DELETE.

### **Observations**

During this lab, students should focus on:

* Configurations for AWS Lambda functions.
* Setting up and managing an AWS DynamoDB table.
* Integrating and managing API Gateway routes and integrations.

### **References**

* [AWS Lambda Documentation](https://docs.aws.amazon.com/lambda/latest/dg/welcome.html)
* [AWS API Gateway Documentation](https://docs.aws.amazon.com/apigateway/latest/developerguide/welcome.html)
* [AWS DynamoDB Documentation](https://docs.aws.amazon.com/dynamodb/latest/developerguide/Introduction.html)

This comprehensive guide ensures students understand the end-to-end process of creating a scalable, serverless backend using AWS services.