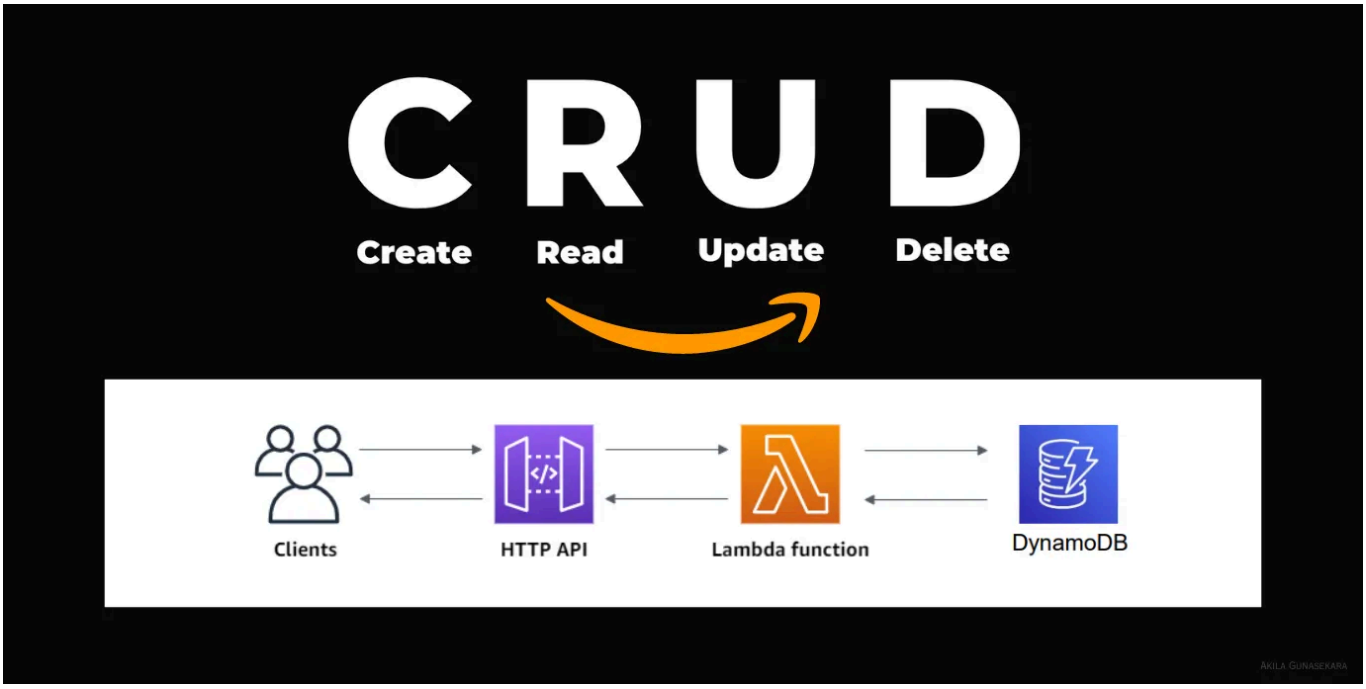
Build a CRUD API with Lambda and DynamoDB

Akila Gunasekara · Follow

7 min read · Jun 13, 2023



https://youtu.be/AZmG-9YE6Go

In this tutorial, I will walk through the process of building a serverless API that enables the ***creates, reads, updates, and deletes*** of items stored in a DynamoDB table. By following the steps outlined below, you will be able to

develop a scalable and efficient API using AWS Lambda, API Gateway, and DynamoDB. Let's get started with the instructions.

You can do it within the **AWS Free Tier**.

Here are the high-level steps we'll follow:

> **Step 1:** *Create a DynamoDB table*
>
> **Step 2:** *Create a Lambda function*
>
> **Step 3:** *Create an HTTP API*
>
> **Step 4:** *Create routes*
>
> **Step 5:** *Create an integration*
>
> **Step 6:** *Attach your integration to routes*
>
> **Step 7:** *Test your API*

Let's dive into the details: https://youtu.be/AZmG-9YE6Go

1: Create a DynamoDB table

Your API's data is stored in a DynamoDB table. Each item in the table has a unique ID, which is used as the partition key.

1. Open the **DynamoDB** console by visiting https://console.aws.amazon.com/dynamodb/.

2. Click on the "**Create table**" button.
3. Provide a name for the table, such as "http-crud-tutorial-items".
4. Specify the partition key as "**id**".
5. Select the "**Create table**" option.

By following these steps, you will successfully create a DynamoDB table named "http-crud-tutorial-items" with the partition key set to "id".

2: Create a Lambda function

A single Lambda function is created for the backend of your API, responsible for performing CRUD operations on items in DynamoDB. The function utilizes events from API Gateway to determine the appropriate interaction with DynamoDB. Although this tutorial simplifies the process by using a single function, it is recommended to follow best practices and create separate functions for each API route.

1. Sign in to the Lambda console by visiting https://console.aws.amazon.com/lambda.
2. Click on the "**Create function**" button.
3. Enter a name for your function, such as "**http-crud-tutorial-function**".
4. Under the "**Permissions**" section, click on "**Change default execution role**".
5. Choose the option to "**Create a new role from AWS policy templates**".
6. Provide a name for the role, such as "**http-crud-tutorial-role**".

7. From the "**Policy templates**" list, select "**Simple microservice permissions**". This policy template grants the Lambda function the necessary permissions to interact with DynamoDB.
8. Choose **Create** function.
9. Open the console's code editor and navigate to the ***index.mjs*** file. Replace the existing code with the following code snippet.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import {
  DynamoDBDocumentClient,
  ScanCommand,
  PutCommand,
  GetCommand,
  DeleteCommand,
} from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});

const dynamo = DynamoDBDocumentClient.from(client);

const tableName = "http-crud-tutorial-items";

export const handler = async (event, context) => {
  let body;
  let statusCode = 200;
  const headers = {
    "Content-Type": "application/json",
  };

  try {
    switch (event.routeKey) {
      case "DELETE /items/{id}":
        await dynamo.send(
          new DeleteCommand({
            TableName: tableName,
            Key: {
              id: event.pathParameters.id,
            },
          })
        );
        body = `Deleted item ${event.pathParameters.id}`;
```

```
        break;
      case "GET /items/{id}":
        body = await dynamo.send(
          new GetCommand({
            TableName: tableName,
            Key: {
              id: event.pathParameters.id,
            },
          })
        );
        body = body.Item;
        break;
      case "GET /items":
        body = await dynamo.send(
          new ScanCommand({ TableName: tableName })
        );
        body = body.Items;
        break;
      case "PUT /items":
        let requestJSON = JSON.parse(event.body);
        await dynamo.send(
          new PutCommand({
            TableName: tableName,
            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;
  } finally {
    body = JSON.stringify(body);
  }

  return {
    statusCode,
    body,
    headers,
  };
};
```

10. Once you've made the necessary changes, click on "Deploy" to update your Lambda function.

By following these steps, you will create a Lambda function named "http-crud-tutorial-function" and set up a new role named "http-crud-tutorial-role" with the appropriate permissions to interact with DynamoDB.

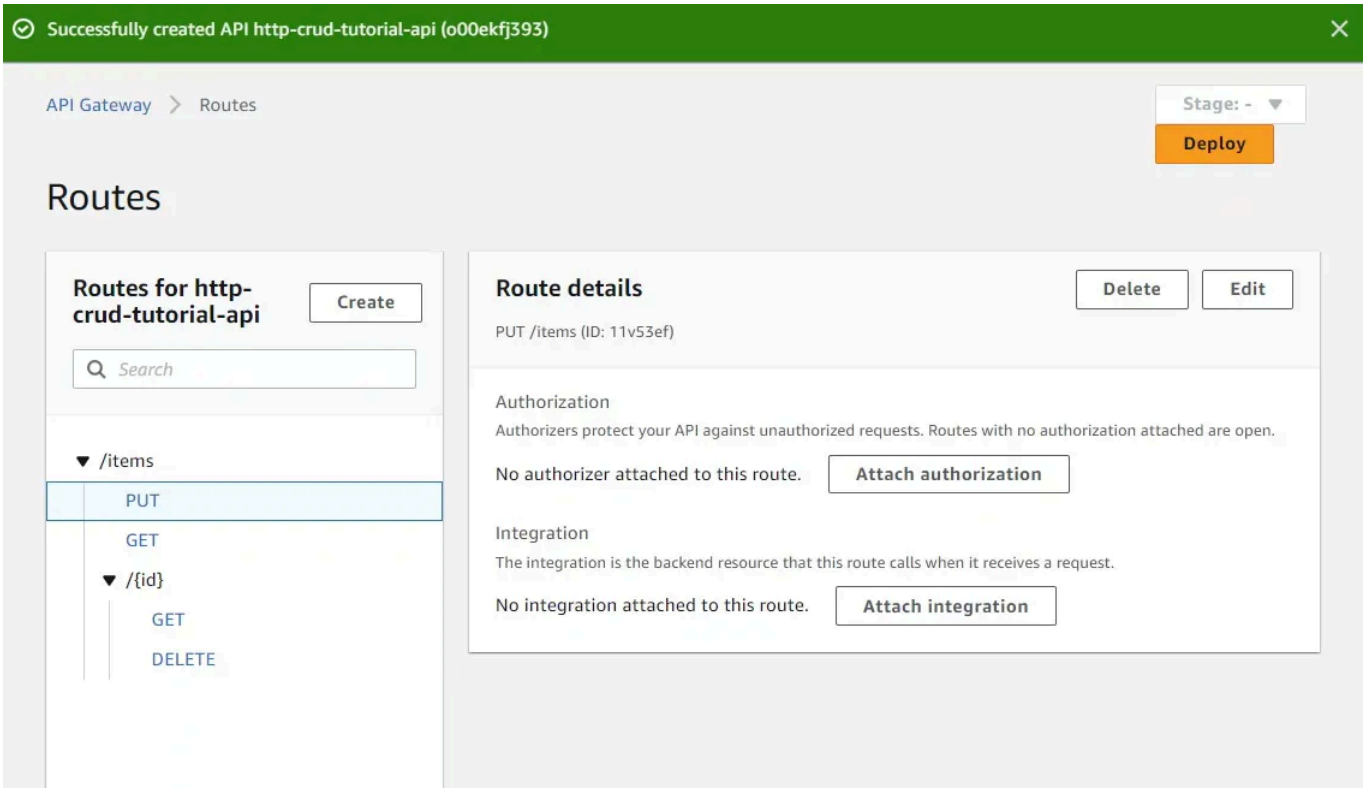
3: Create an HTTP API

To create an HTTP API and establish an HTTP endpoint for your Lambda function, follow these steps:

1. Sign in to the API Gateway console by visiting *https://console.aws.amazon.com/apigateway*.
2. Click on "**Create API**" and select the "Build" option for creating an HTTP API.
3. Enter a name for your API, such as "**http-crud-tutorial-api**", in the "API name" field.
4. Proceed to the next step by clicking on "**Next**" without configuring routes at this point. You will create routes later.
5. Review the automatically generated stage by API Gateway and ensure its settings meet your requirements. Click on "**Next**" to proceed.
6. Finally, click on "**Create**" to create the HTTP API.

By following these steps, you will successfully create an HTTP API named "**http-crud-tutorial-api**" that serves as an HTTP endpoint for your Lambda function. In the subsequent steps, you can configure routes and integrations to connect your API with the Lambda function.

4: Create routes



To create routes for your API,

- `GET /items/{id}`
- `GET /items`
- `PUT /items`
- `DELETE /items/{id}`

which determine how incoming requests are directed to backend resources, follow these steps:

1. Sign in to the API Gateway console by visiting *https://console.aws.amazon.com/apigateway*.

2. Select your API from the available APIs.
3. Navigate to the "**Routes**" section.
4. Click on "**Create**" to start creating a new route.
5. Choose the desired HTTP method for the route. For example, select "**GET**".
6. Enter the path for the route, such as "**/items/{id}**". The "{id}" in the path represents a path parameter that API Gateway extracts from the request path when a client makes a request.
7. Click on "**Create**" to finalize the creation of the route.
8. Repeat steps 4 to 7 for each of the remaining routes: "GET /items", "DELETE /items/{id}", and "PUT /items".

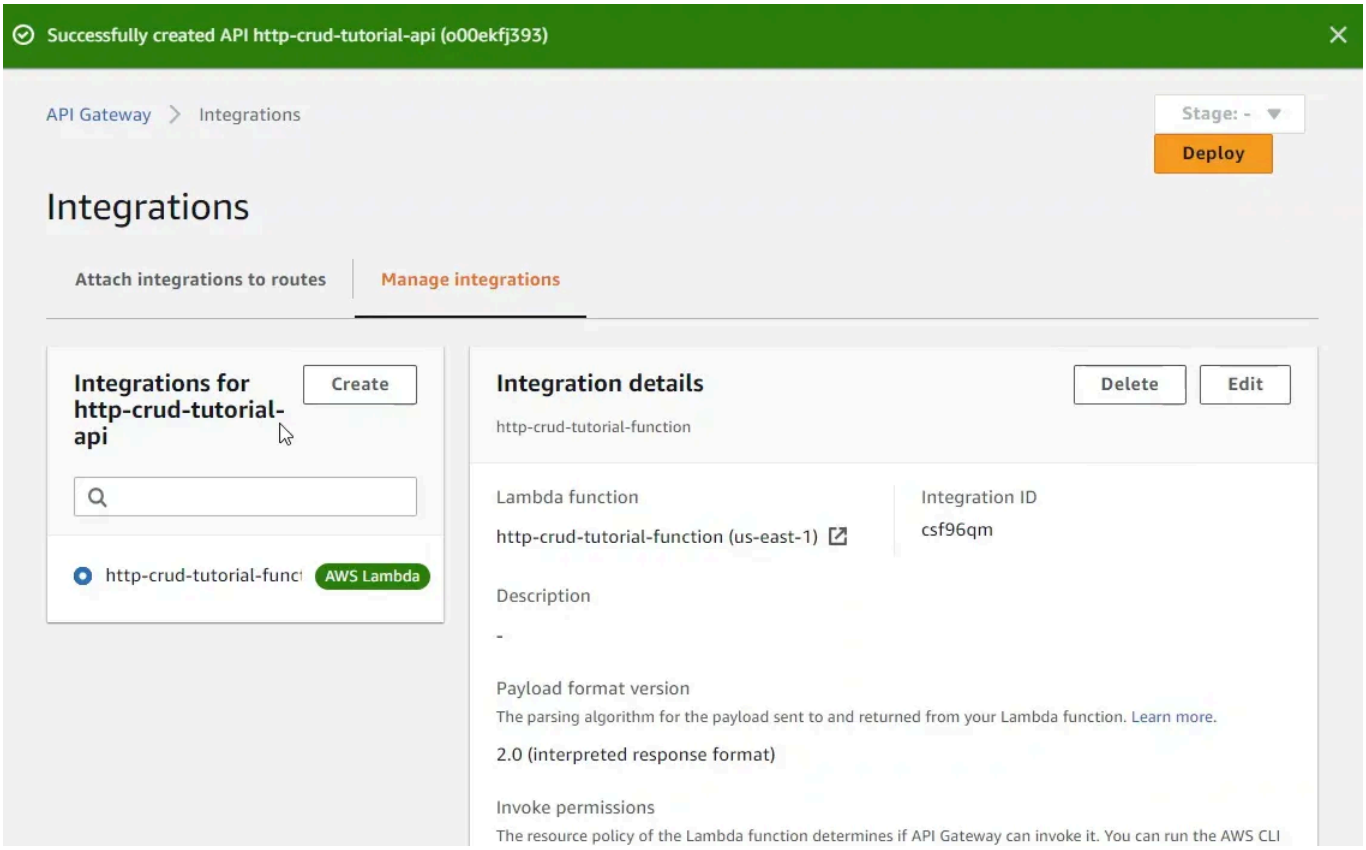
By following these steps, you will successfully create the necessary routes for your API. Each route specifies an HTTP method and a resource path, allowing API Gateway to appropriately handle incoming requests and direct them to the corresponding backend resources.

5: Create an integration

To create an integration that connects a route to backend resources, follow these steps:

1. Sign in to the API Gateway console by visiting *https://console.aws.amazon.com/apigateway*.
2. Select your API from the available APIs.
3. Navigate to the "**Integrations**" section.

4. Click on “**Manage integrations**” and then select “**Create**” to begin creating a new integration.
5. Skip the step to attach the integration to a route for now. You will complete this step later.
6. Choose “**Lambda function**” as the integration type.
7. Enter the name of your Lambda function, such as “**http-crud-tutorial-function**”, in the “**Lambda function**” field.
8. Click on “**Create**” to finalize the creation of the integration.

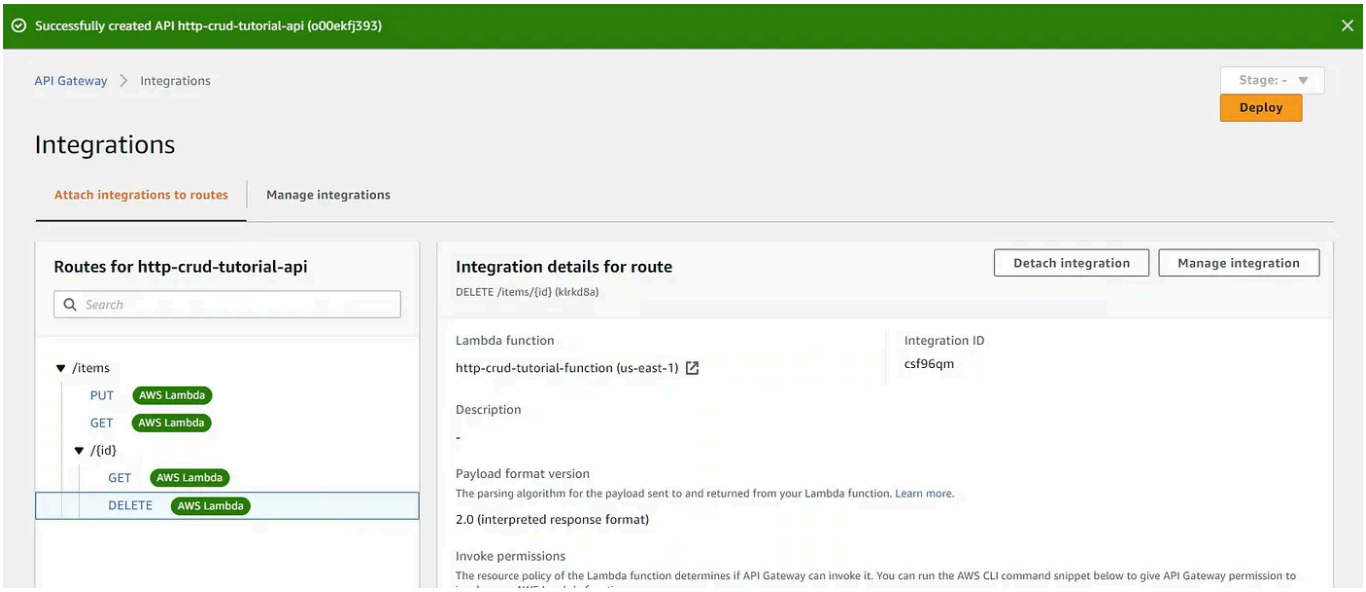


By following these steps, you will create an integration that connects your API’s routes to the backend resources. In this example, a single Lambda integration is used for all routes. In the subsequent steps, you will attach this integration to the appropriate routes.

6: Attach your integration to routes

To attach the created integration to your API's routes, follow these steps:

1. Sign in to the API Gateway console by visiting *https://console.aws.amazon.com/apigateway*.
2. Select your API from the available APIs.
3. Navigate to the "**Integrations**" section.
4. Choose the route you want to attach the integration to.
5. Under the "**Choose an existing integration**" section, select "**http-crud-tutorial-function**" as the integration.
6. Click on "**Attach integration**" to associate the integration with the selected route.
7. Repeat steps 4-6 for all the remaining routes of your API.



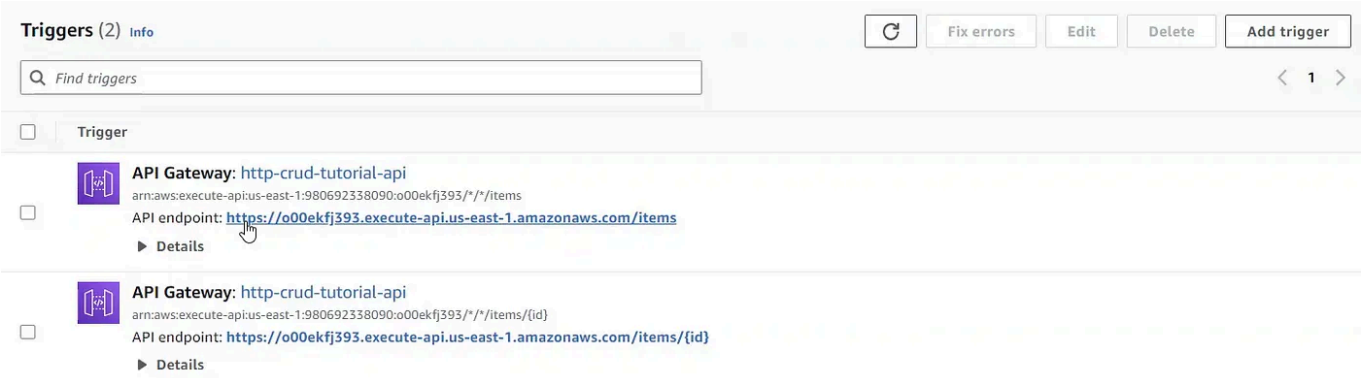
By following these steps, you will attach the integration to each route of your API. This ensures that when a client calls any of the routes, the associated

Lambda function will be invoked, enabling the desired functionality and interaction with your backend resources.

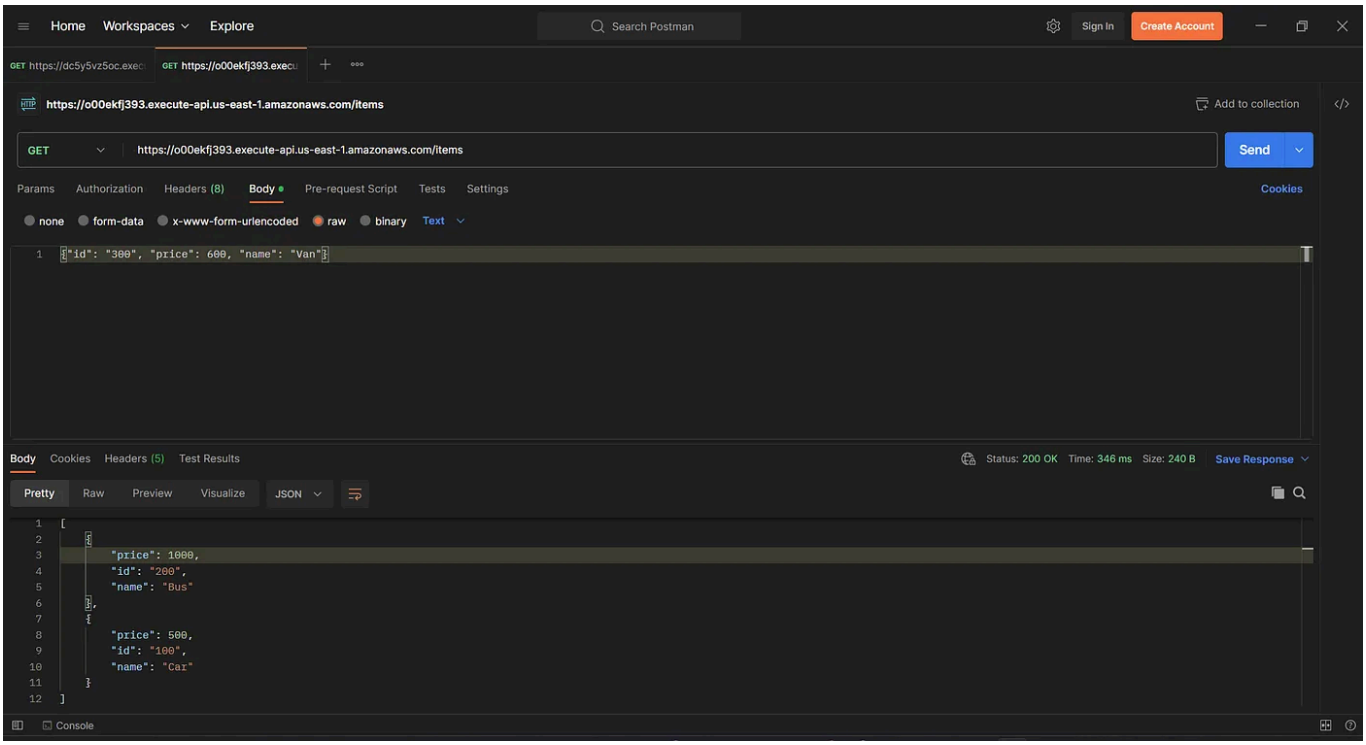
7: Test your API

To obtain the URL for invoking your API and test its functionality, follow these steps:

1. Sign in to the API Gateway console by visiting https://console.aws.amazon.com/apigateway.



1. Select your API from the available APIs.
2. Make a note of your API's invoke URL. You can find this URL under the "**Invoke URL**" section on the Details page of your API.
3. Copy your API's invoke URL. The full URL looks like https://abcdef123.execute-api.us-west-2.amazonaws.com.



https://youtu.be/AZmG-9YE6Go

By following these steps, you will obtain the invoke URL for your API, which you can use to test its functionality using tools like curl or any other HTTP client of your choice.

Written by Akila Gunasekara

8 Followers

Follow

More from Akila Gunasekara

Akila Gunasekara

Customizing email verification messages Amazon Cognito

To verify a user's email address using Amazon Cognito, you have two options: sending the...

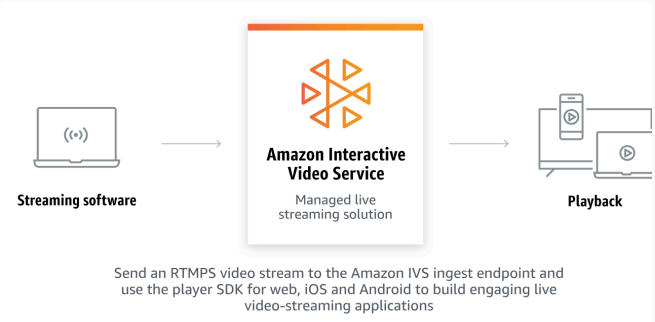
Oct 23, 2023

Akila Gunasekara

How to Integrate PayHere into React.js (Sandbox)-2023/2024

Connecting PayHere Payment Gateway to a React Application

Nov 30, 2023 11 1

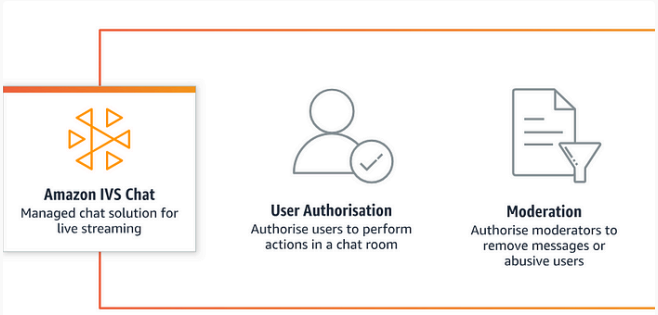


Akila Gunasekara

What is Amazon IVS?

Amazon IVS (Interactive Video Service) is a fully managed live video streaming solution...

Jun 19, 2023 28



Akila Gunasekara

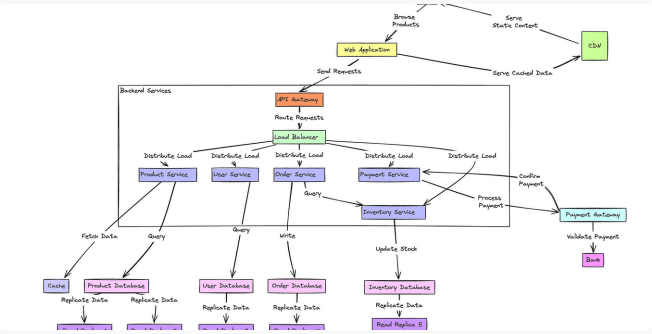
How cCzzzhat works

The diagram represents a use case for live video stream chat. It consists of three...

Jul 12, 2023 20

See all from Akila Gunasekara

Recommended from Medium



Kevin Wong

E-commerce Platform like Amazon

Draft Notes

May 16

Ravi Patel

Building a Secure User Registration and Login API with...

Introduction:

Mar 13 10 1

Lists

Staff Picks

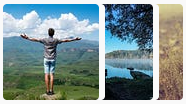
723 stories · 1261 saves

Stories to Help You Level-Up at Work

19 stories · 767 saves

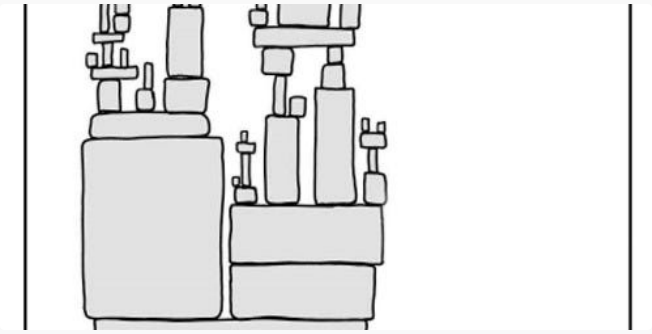
Self-Improvement 101

20 stories · 2641 saves



Productivity 101

20 stories · 2269 saves



Louis Trinh

Mongoose Schema Modeling for E-commerce Products: Best...

Mongoose Schema Model for Products in E-commerce

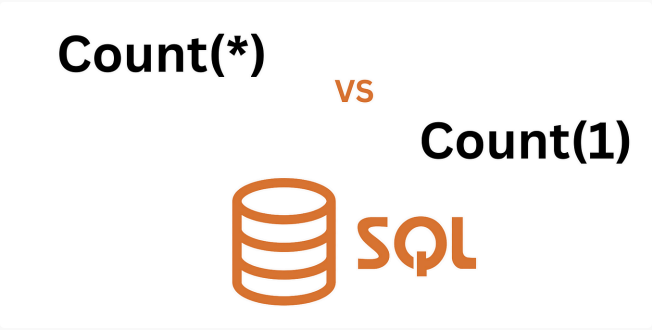
Jun 7

Abhigyana Satpathy

Understanding Amazon DynamoDB: A Comprehensive...

Amazon DynamoDB is a fully managed NoSQL database service provided by Amazo...

Jul 11 3

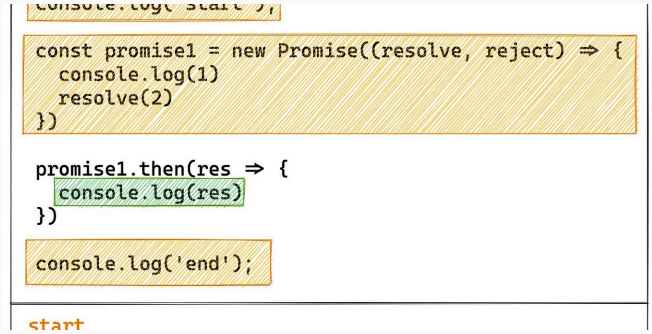


Vishal Barvaliya in Data Engineer

Count(*) vs Count(1) in SQL.

If you've spent any time writing SQL queries, you've probably seen both `COUNT(*)` and...

Mar 8 1.1K 31



Shuai Li in Programming Domain

Can You Answer This Senior Level JavaScript Promise Interview...

Most interviewees failed on it.

Aug 12 1.1K 13

See more recommendations