



Open in app

Get started

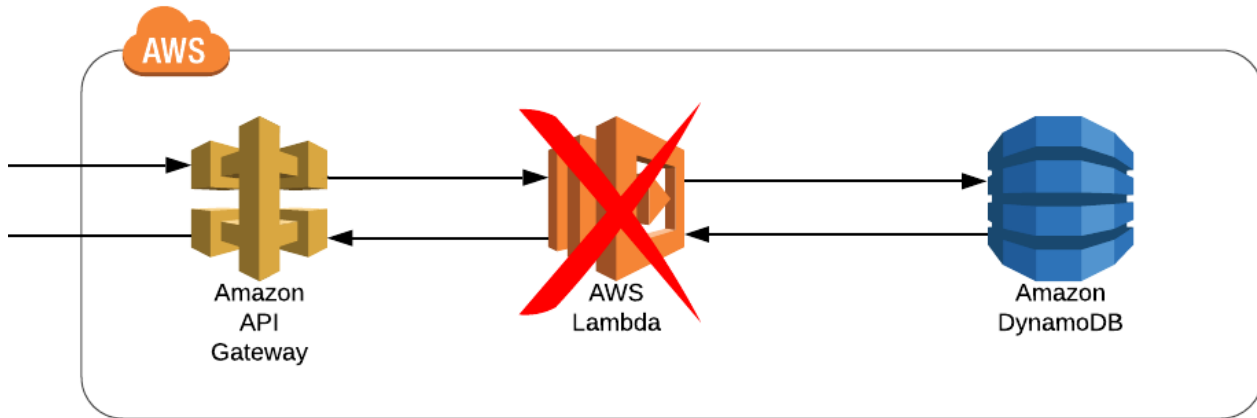


Published in BRLink · Follow



Rafael Campana · Follow

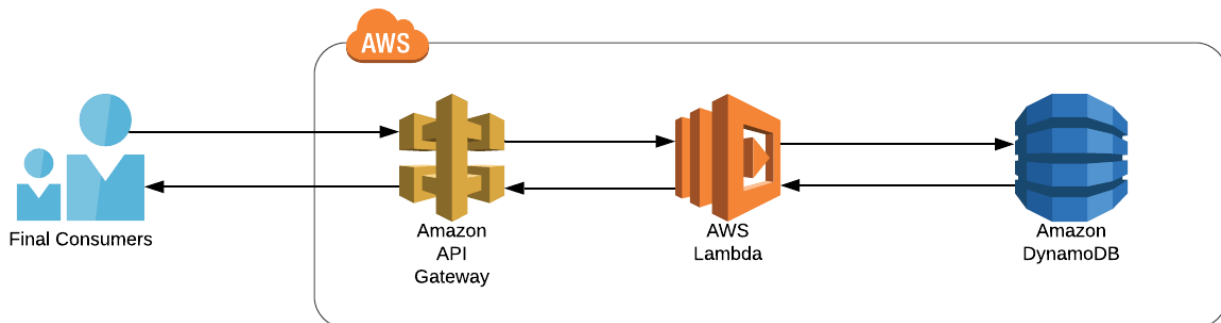
Aug 17, 2019 · 4 min read



## Rest Api with ApiGateway and DynamoDb

The common approach of software development is use a middleware function to manipulate the data between api and database.

The serverless way to achieve this in AWS world is use a lambda function to do this job as following sample:

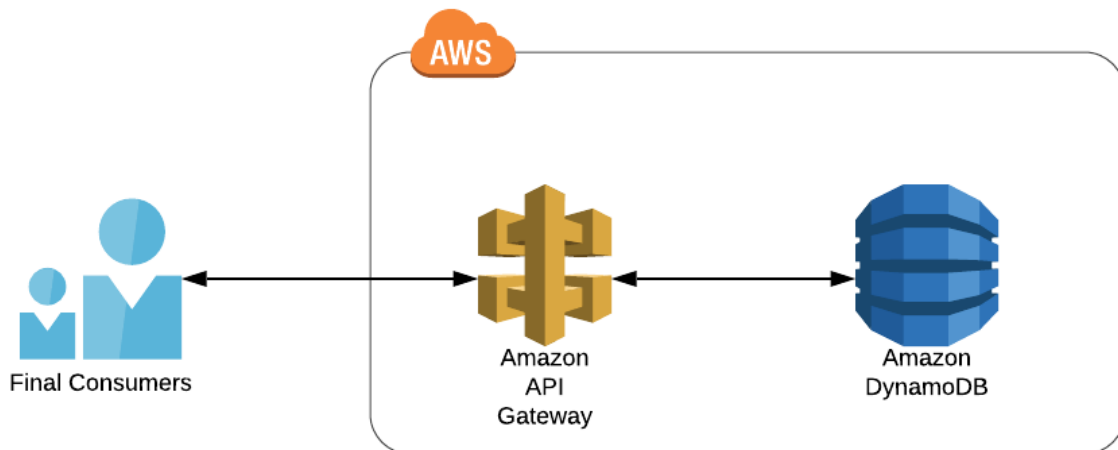


Sometimes (a lot of times to be honest) we just use this to as proxy to database. What about remove this useless code and keep the things simple and clean as a possible?



[Open in app](#)[Get started](#)

With a ApiGateway using mapping templates you can implement a restfull api, with almost zero line codes and delivery a good solution in a record time with lower cost possible . The schema you can see bellow:



Once you convinced , let's to this

## 1- Create a DynamoDB table

To this sample, I've create a super simple table just with **id**, **name** and **userName**. I won't detail how to do this, but you can see the steps here:

<https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/SampleData.CreateTables.html>

## 2- Create a Role

Your role should have access to your table, for this sample I've created a full dynamo db, sure in a real world is a good practice create a fine granulated access for each api,table and method





Open in app

Get started

Permissions Trust relationships Tags Access Advisor Revoke sessions

▼ Permissions policies (2 policies applied)

Attach policies

Policy name
▶ AmazonAPIGatewayPushToCloudWatchLogs
▶ AmazonDynamoDBFullAccess

▼ Permissions boundary (set)

Set a permissions boundary to control the maximum permissions this role can have. This is not a c

Change boundary Remove boundary

▶ AmazonDynamoDBFullAccess (AWS managed policy)

Polices of the Role

### 3- Create Your Api Gateway

Create a API Gateway with the following structure :

APIs > users (p3i31ivosk) > Resources > / (8yhedjuff6)

Resources Actions / Methods

- ▼ /
  - POST
  - ▼ /{username}
    - GET

POST

DynamoDB

Authorization None

API Key Not required

On method post, you should point the Dynamo db and indicate your role created in the



[Open in app](#)[Get started](#)Integration type  Lambda Function ⓘ HTTP ⓘ Mock ⓘ AWS Service ⓘ VPC Link ⓘAWS Region AWS Service AWS Subdomain HTTP method Action Type  Use action name Use path overrideAction Execution role Content Handling  ⓘUse Default Timeout  ⓘ

Once created, you should create a mapping template to convert a body content from the post rest request to a PutItem DynamoDb request, in the **Integration Request > Mapping Templates** add a new item with content type application/json with the following content:

```
{
  "TableName": "users",
  "Item": {
    "uid": {
      "S": "$context.requestId"
    },
    "name": {
      "S": "$input.path('$.name')"
    },
    "userName": {
      "S": "$input.path('$.userName')"
    }
  }
}
```





Open in app

Get started

with a requestId unique generated by api gateway request.

Once this done, let's do our first test and analyse the response:

Request Body

```

1 {
2   "name" : "Rafael Ortega Campana",
3   "userName" : "rocampana"
4 }

```

Test

```

Sat Aug 17 04:17:52 UTC 2019 : Method request body before transformations: {
  "name" : "Rafael Ortega Campana",
  "userName" : "rocampana"
}
Sat Aug 17 04:17:52 UTC 2019 : Endpoint request URI: https://dynamodb.us-east-1.amazonaws.com/putitem
Sat Aug 17 04:17:52 UTC 2019 : Endpoint request headers: {Authorization=*****
*****
*****941251, X-Amz-Date=20190817T04
mzn-apigateway-api-id-p3i11vosk, Accept=application/json, User-Agent=AmazonAPIGateway_p
X-Amz-Security-Token=Ago7b3JpZ2luX2VjEL3////////wEaCXVzLWVhc3QtMSJIMEYCIQDf6BvktTdseD5
CYtw+9/OFq8cPqbXfCB/AIhALdqQL5sHJqcmjo+6hh13rPHCiROVncCZ+GIp1GLH7I1KoEDCFUQABoMOTk3MTAxM
d8/FJfMLa+Hd8pXEg3gkL4SF6E54ckv5QmN4K3IrsGu9rd43eFagBcPTGswq/zNHsxZ/pN5ErLsqZ0LN8UynT0w2
kjNE1QLbJNaDVvcPM8H+MlyOEFagZstEBvaVrv4IV7rwU8SYoD72qJ+3pZrjTAoBoNlysrVX0aUT6uMwZJRhJkRRw
4TCBHOzTAoHskwCbzKVkOF3J1+foja+GV1M2MERzRxbIawb1KRdd2V0Zfu8Zw20ahUdmflvqpJ0EKGZms1ECsb6
wXGPFyQ4ynnR6tdzEscwits26LC628ebeJKD1LwstXBiYG/VgBBiud8NvDECZe006j3Ln5s32I [TRUNCATED]
Sat Aug 17 04:17:52 UTC 2019 : Endpoint request body after transformations: {
  "TableName": "Users",
  "Item": {
    "uid": {
      "S": "faf57d93-c0a5-11e9-b7fb-af04296c5c9d"
    },
    "name": {
      "S": "Rafael Ortega Campana"
    },
    "userName": {
      "S": "rocampana"
    }
  }
}
Sat Aug 17 04:17:52 UTC 2019 : Sending request to https://dynamodb.us-east-1.amazonaws.c
om/putitem
Sat Aug 17 04:17:52 UTC 2019 : Received response. Status: 200, Integration latency: 23 m
Sat Aug 17 04:17:52 UTC 2019 : Endpoint response headers: {Server=Server, Date=Sat, 17 A
ug 2019 04:17:52 GMT, Content-Type=application/x-amz-json-1.0, Content-Length=2, Connection=keep-
alive, x-amz-crc32=27456141-
}

```

As you can see, the rest call is converted to putitem invoke to dynamo and the response was 200, the item has been created and you can check on dynamo db console:)

### 4- Create a Resource and Get Method

Create a **Resource** with resource path {username} and a get method associated.

Configure the get method almost as the post, but changing the "PutItem" to "Query", once the action now needed is that query in the database.



[Open in app](#)[Get started](#)

GET

POST

Integration type  Lambda Function ⓘ HTTP ⓘ Mock ⓘ AWS Service ⓘ VPC Link ⓘAWS Region AWS Service AWS Subdomain HTTP method Action Type  Use action name Use path overrideAction Execution role Content Handling  ⓘUse Default Timeout  ⓘ

Now we need to create again the **Mapping Templates on Integration Request**, but now with the following translation:

```
{
  "TableName": "users",
  "IndexName": "userName-index",
  "KeyConditionExpression": "userName = :val",
  "ExpressionAttributeValues": {
    ":val": {
      "S": "$input.params('username')"
    }
  }
}
```

This maps the resource param **username** to dynamoDb with the condition “userName” and does the query.



[Open in app](#)[Get started](#)

```
#set($inputRoot = $input.path('$'))
{
  "users": [
    #foreach($elem in $inputRoot.Items) {
      "uid": "$elem.uid.S",
      "name": "$elem.name.S",
      "userName": "$elem.userName.S"
    }#if($foreach.hasNext),#end
  ]
}#end
```

Let`s test and analyse the response:

#### ← Method Execution /{username} - GET - Method Test

Make a test call to your method with the provided input

Path

{username}

rocampana

Query Strings

{username}

param1=value1&param2=value2

Headers

{username}

Use a colon (:) to separate header name and value, and new lines to declare multiple headers. eg. Accept:application/json.

Stage Variables

No [stage variables](#) exist for this method.

Request: /rocampana

Status: 200

Latency: 56 ms

Response Body

```
{
  "users": [
    {
      "uid": "c889ffba-c0a3-11e9-bc04-7740352d9541",
      "name": "Rafael Campana",
      "userName": "rocampana"
    }
  ]
}
```

Response Headers

```
{"X-Amzn-Trace-Id": "Root=1-5d57832e-ddcc0b1fcc82c5d8525e"
```

Logs

```
Execution log for request e9d25ddc-c0a7-11e9-a229-8d6f48
Get Api 17 04/21/2020 UTC 2020-04-21T17:04:21.000Z
```

We have passed the username created early “rocampana” witch contains a array (with just one item) with all users with this profile registered on database.

And now we have **two methods**:

**post** — *users/*

**get** — *users/{username}*





Open in app

Get started

Now, ensure that your api gateways resources is accessed securely and be happy

