Amazon DynamoDB: Building a Serverless Web Application

**SPL-TF-200-DBDYL6 - Version 1.0.7**

© 2024 Amazon Web Services, Inc. or its affiliates. All rights reserved. This work may not be reproduced or redistributed, in whole or in part, without prior written permission from Amazon Web Services, Inc. Commercial copying, lending, or selling is prohibited. All trademarks are the property of their owners.

Note: Do not include any personal, identifying, or confidential information into the lab environment. Information entered may be visible to others.

Corrections, feedback, or other questions? Contact us at [*AWS Training and Certification*](https://support.aws.amazon.com/#/contacts/aws-training).

**Lab Overview**

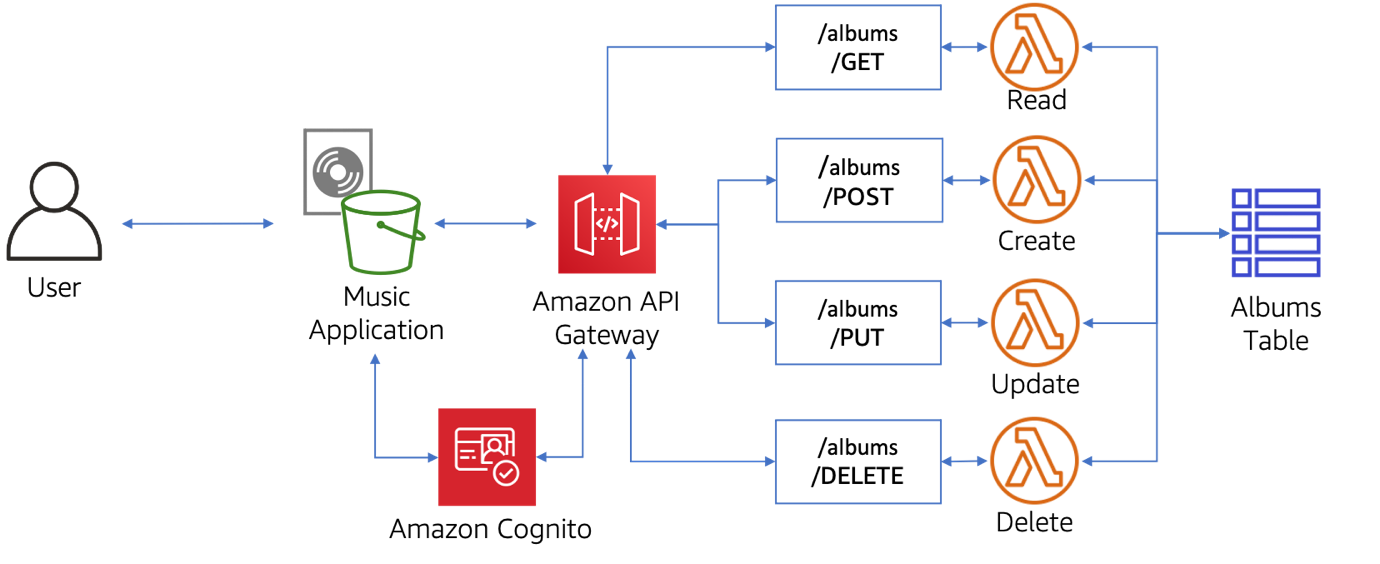
This lab is intended to be the final lab in the Developing with DynamoDB curriculum. It builds upon skills already covered in those labs. As such, it is expected to be more challenging, with opportunities to complete development by referencing documentation and leveraging prior experience rather than just using provided answers. However, complete answers are provided. Refer to the [Answer Key](https://labs.skillbuilder.aws/sa/lab/arn%3Aaws%3Alearningcontent%3Aus-east-1%3A470679935125%3Ablueprintversion%2FSPL-TF-200-DBDYL6-1%3A1.0.7-b7ad0e80/en-US#AnswerKey) in case you reach a point where you need them.

AnyCompany is deploying a new web-based music application to complement its movie database application. They have chosen to implement this using Amazon DynamoDB and a serverless AWS Lambda based architecture.

This application will have four general functions: It will allow users to show all music albums in the database, or those those of a certain genre. It will allow users to add or delete entries to the database, as well as update the non-key attributes of a given album. You have been tasked with creating and implementing this functionality combining Amazon DynamoDB, AWS Lambda, and Amazon API Gateway in conjunction with a front-end website you have been given. There is an optional task of integrating Amazon Cognito with the application to authorize users.

The music application will have the following components:

* A static front-end written in HTML/Javascript hosted on Amazon S3.
* A serverless backend leveraging Amazon API Gateway, AWS Lambda, and Amazon Cognito.
* A DynamoDB table for the persistence layer.



TOPICS COVERED

By the end of this lab, you will be able to:

* Create and deploy Lambda functions written in Python to perform operations on a DynamoDB table.
* Leverage DynamoDB conditional expressions.
* Create and deploy API Gateway endpoints to proxy Lambda functions.
* Create and configure Amazon Cognito user pools for authentication.

TECHNICAL KNOWLEDGE PREREQUISITES

To successfully complete this lab, you should be familiar with navigating the AWS Management Console, editing scripts using an integrated development environment (IDE), and implementing basic Python scripts to perform CRUD operations with DynamoDB. You should also have a basic understanding of the Amazon Simple Storage Service (Amazon S3), AWS Lambda, and Amazon API Gateway services.

ICON KEY

Various icons are used throughout this lab to call attention to certain aspects of the guide. The following list explains the purpose for each one:

* The keyboard icon specifies that you must run a command.
* The clipboard icon indicates that you can verify the output of a command or edited file by comparing it to the provided example.
* The note icon specifies important hints, tips, guidance, or advice.
* The exclamation icon draws special attention to actions that are irreversible and could potentially impact the failure of an action.
* The “i” circle icon specifies where to find more information.
* The person with a check mark icon indicates an opportunity to check your knowledge and test what you have learned.

**Start lab**

1. To launch the lab, at the top of the page, choose **Start lab**.

**Caution:** You must wait for the provisioned AWS services to be ready before you can continue.

1. To open the lab, choose **Open Console**.

You are automatically signed in to the AWS Management Console in a new web browser tab.

**WARNING:** **Do not change the Region unless instructed.**

COMMON SIGN-IN ERRORS

**Error: You must first sign out**



If you see the message, **You must first log out before logging into a different AWS account:**

* Choose the **click here** link.
* Close your **Amazon Web Services Sign In** web browser tab and return to your initial lab page.
* Choose **Open Console** again.

**Error: Choosing Start Lab has no effect**

In some cases, certain pop-up or script blocker web browser extensions might prevent the **Start Lab** button from working as intended. If you experience an issue starting the lab:

* Add the lab domain name to your pop-up or script blocker’s allow list or turn it off.
* Refresh the page and try again.

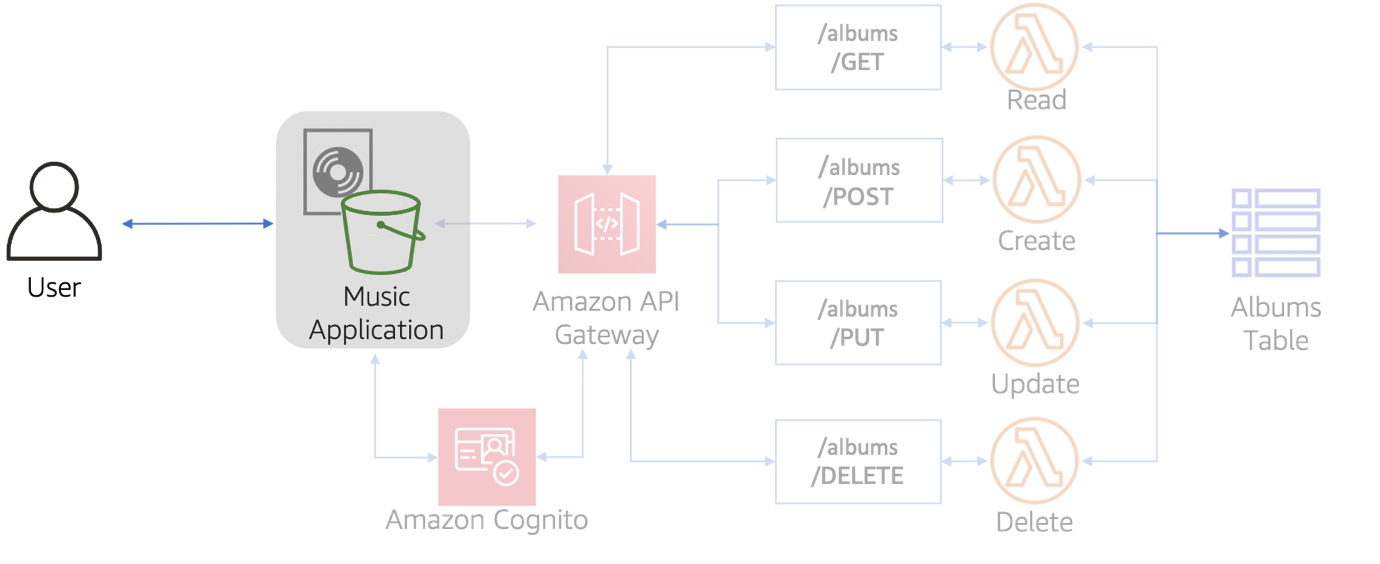
**Task 1: Deploying the music application**

As a first step, you will deploy the front-end of the music application to S3.

Amazon S3 comes with a simple web interface that you can leverage to store and retrieve any amount of data, at any time, from anywhere on the web. For more information, refer to *What is Amazon S3?* in the [Additional Resources](https://labs.skillbuilder.aws/sa/lab/arn%3Aaws%3Alearningcontent%3Aus-east-1%3A470679935125%3Ablueprintversion%2FSPL-TF-200-DBDYL6-1%3A1.0.7-b7ad0e80/en-US#AdditionalResources) section.

In this task you will use the AWS CLI to upload the static assets of a web-based music application to an existing S3 bucket configured for website hosting.

This task covers the “Music Application” section of the architecture diagram:



TASK 1.1: UPLOADING CONTENT TO THE S3 BUCKET PROGRAMMATICALLY

1. If you have not already done so, follow the steps in the [Start Lab](https://labs.skillbuilder.aws/sa/lab/arn%3Aaws%3Alearningcontent%3Aus-east-1%3A470679935125%3Ablueprintversion%2FSPL-TF-200-DBDYL6-1%3A1.0.7-b7ad0e80/en-US#StartLab) section to log into the AWS Management Console.
2. In the AWS Management Console, in the unified search bar, search for and choose

Cloud9

1. On the **Environments** page, on the **Lab\_Cloud9\_IDE**, choose **Open**.

The AWS Cloud9 environment opens in a new browser tab.

1. Inside of the AWS Cloud9 environment, to the right of the **Welcome** tab, choose the plus  icon, and then select **New Terminal**.
2. In the AWS Cloud9 terminal, run the following command to install Python3 and Boto3, the AWS SDK for Python:

sudo yum -y install python38

sudo alternatives --set python /usr/bin/python3.8

pip install boto3

1. The AWS Cloud9 development environments come prepackaged with the AWS CLI and tooling for over 40 programming languages. Run the following command to verify the AWS CLI installation:

aws --version

 The output should show aws-cli 1.x.x, similar to the following:

aws-cli/1.19.94 Python/3.6.12 Linux/4.14.232-123.381.amzn1.x86\_64 botocore/1.20.96

 As of the writing of this lab, the default version of the AWS CLI installed on the AWS Cloud9 instance is 1.19.94. We recommend upgrading to version 2 for access to the latest features, which may not be made available in version 1. For more information, refer to *Installing, updating, and uninstalling the AWS CLI* in the [Additional Resources](https://labs.skillbuilder.aws/sa/lab/arn%3Aaws%3Alearningcontent%3Aus-east-1%3A470679935125%3Ablueprintversion%2FSPL-TF-200-DBDYL6-1%3A1.0.7-b7ad0e80/en-US#AdditionalResources) section.

1. Run the following commands to download, extract, and install the AWS CLI version 2 package:

curl "https://awscli.amazonaws.com/awscli-exe-linux-x86\_64.zip" -o "awscliv2.zip"

unzip awscliv2.zip

sudo ./aws/install

1. Run the following command to download, extract, and install the AWS CLI version 2 package:

aws --version

 The output should show aws-cli 2.x.x, similar to the following:

aws-cli/2.13.5 Python/3.11.4 Linux/4.14.320-168.534.amzn1.x86\_64 exe/x86\_64.amzn.2018 prompt/off

1. In the terminal window, navigate over to the

web-app

 directory with the following command:

cd lab-files/web-app/

1. Replace the

{ChangeMeS3Bucket}

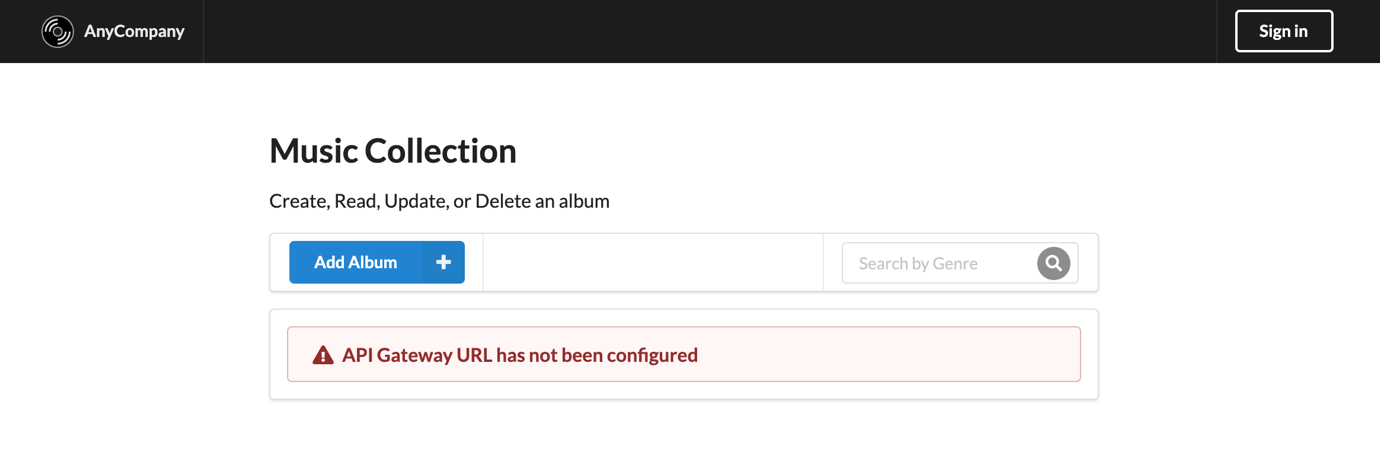
 in the following command with the value of **S3BucketName** listed to the left of these instructions. Run the edited command to upload the website to S3. Note that this command will sync the contents of your working directory (the “pwd” command) with the S3 bucket.

aws s3 sync $(pwd) s3://{ChangeMeS3Bucket}

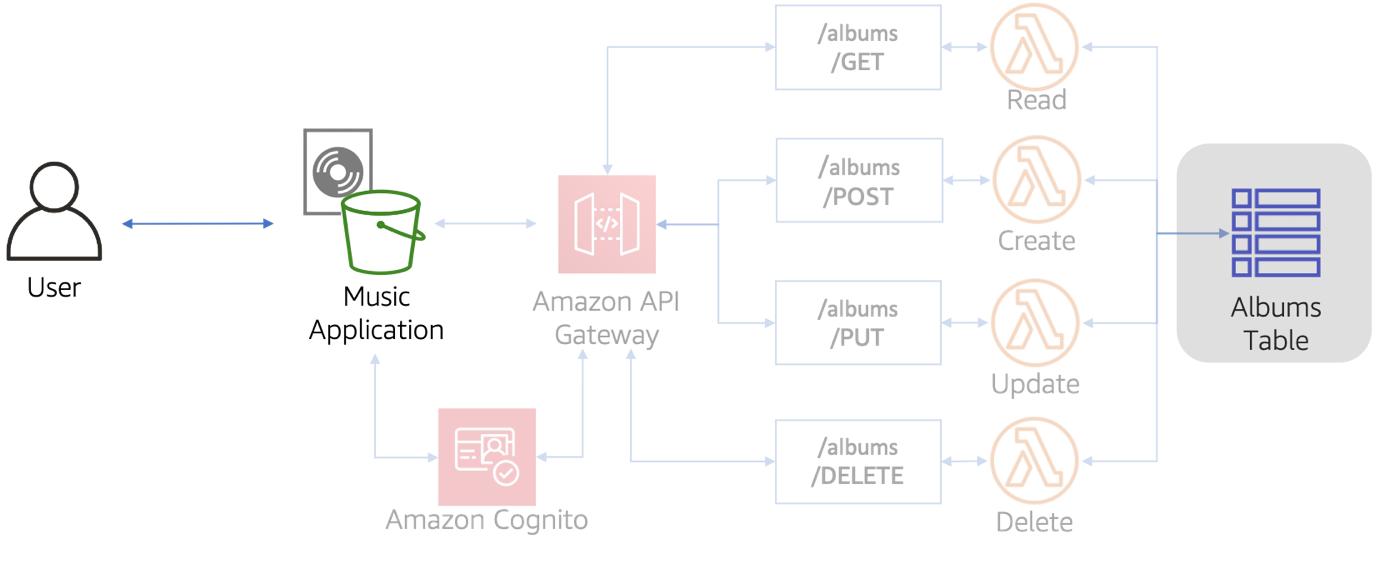
All the website files have now been uploaded to S3.

1. Copy the value of **WebsiteURL** listed to the left of these instructions, and paste to a new browser tab.

The music application will open in a new tab, showing a warning that the API Gateway URL has not been configured. The music application is now being hosted by Amazon S3.



**Task 2: Creating and managing a DynamoDB table**



The music application will use the data stored in a DynamoDB table created for this purpose. In this task, you will create a table named **Albums** using the following schema, and load sample data into that table.

* Artist – String (Partition or HASH key)
* Album – String (Sort or RANGE key)
* Genre – String
* Year – Number
* Rank – Number

This table will also need to use a Global Secondary Index (GSI) named **genre-index** based on the **Genre** attribute, in order to support querying on that attribute.

TASK 2.1: CREATE A DYNAMODB TABLE

 It is recommended that you take a moment to open and examine the Python script that will create the table. It can be found in your AWS Cloud9 environment as lab-files/sample-lambda-functions/CreateTable.py.

1. Run the following command in the AWS Cloud9 terminal to navigate back to the starting directory from the lab-files/web-app directory:

cd ../..

1. Run the following command to create the Albums table:

python lab-files/sample-lambda-functions/CreateTable.py Albums

1. Run the following command to verify successful creation of the Albums table:

aws dynamodb list-tables

 The output should show a JSON listing of the table created, similar to the following:

{

"TableNames": [

"Albums"

]

}

 For even more detail about the Albums table, consider trying the describe-table command with the AWS CLI. For more information on AWS CLI commands, refer to *AWS CLI Command Reference for DynamoDB* in the [Additional Resources](https://labs.skillbuilder.aws/sa/lab/arn%3Aaws%3Alearningcontent%3Aus-east-1%3A470679935125%3Ablueprintversion%2FSPL-TF-200-DBDYL6-1%3A1.0.7-b7ad0e80/en-US#AdditionalResources) section.

TASK 2.2 LOAD THE INITIAL DATA SET

Now you will load the Albums table with data for the application to use. This initial data set is provided in the albumlist.json file located in the lab-files folder in your AWS Cloud9 environment. You will need to run a script that reads the JSON file and adds the items to the table just created.

 It is recommended that you take a moment to open and examine the Python script that will load the data. It can be found in your AWS Cloud9 environment as lab-files/sample-lambda-functions/LoadAlbumData.py.

1. Run the following command to load the data into the Albums table:

python lab-files/sample-lambda-functions/LoadAlbumData.py Albums lab-files/albumlist.json

1. Run the following command to verify that the table has loaded the items:

aws dynamodb scan --table-name Albums | head

 The output should show the first 10 lines of a JSON-formatted dataset, similar to the following:

{

"Items": [

{

"Rank": {

"N": "224"

},

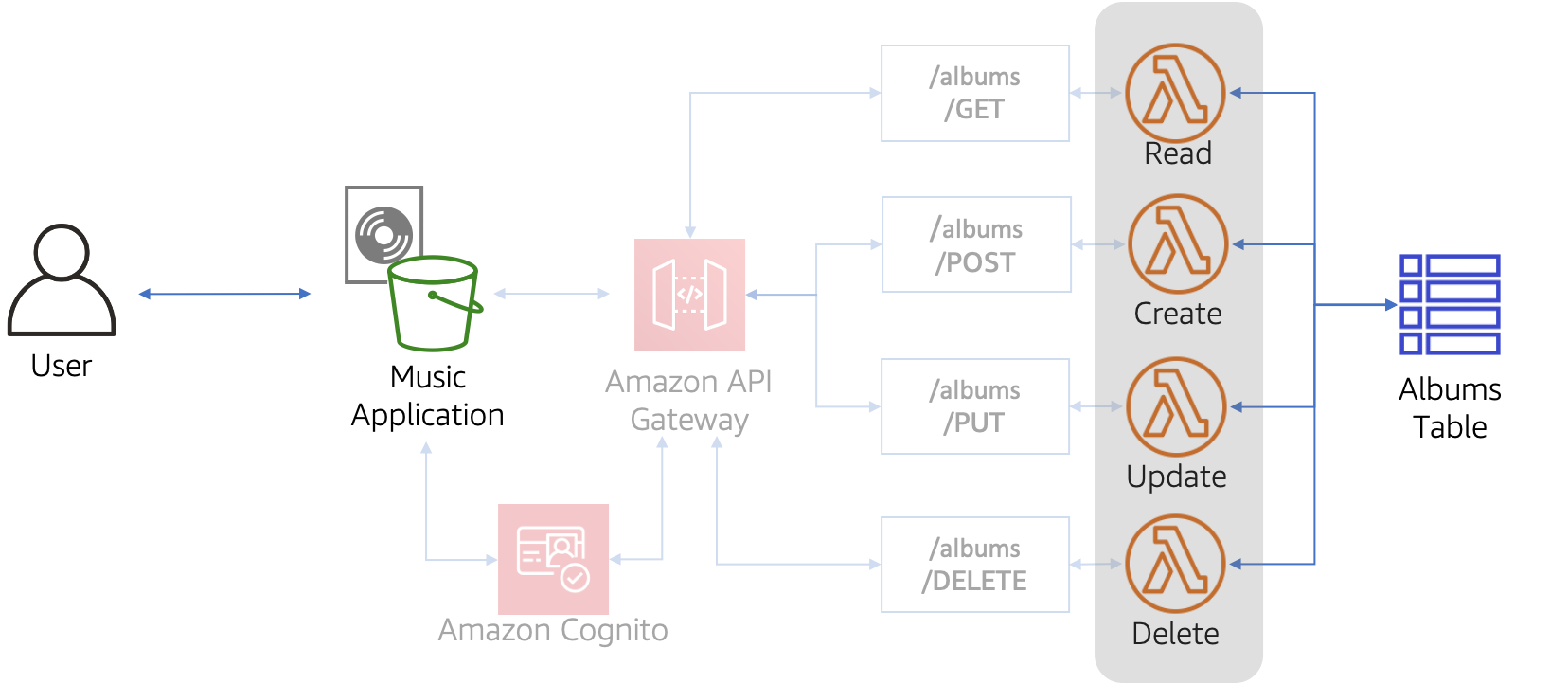
"Album": {

"S": "The Neil Diamond Collection"

},

"Artist": {

**Task 3: Developing AWS Lambda functions**



The music application will have four primary actions, each one of which will need to be supported by a corresponding AWS Lambda function to access the DynamoDB table holding the data. With a microservice architecture like this, the functions of an application run as their own individual services and communicate with the other parts of the application through a well-defined API. This allows one to more easily update existing components, or add entirely new components, to an application while minimizing downtime. The four primary actions used here are:

* Show some or all of the data from the table
* Add a table entry
* Edit a table entry
* Conditionally delete a table entry

In this task you will create AWS Lambda functions to perform these actions. AWS Lambda can use multiple languages, but this lab will use Python.

 Use the existing user role that includes the “LambdaRunRole” string for all Lambda functions you create in this lab. Creation of new IAM roles is not supported in this lab.

 Note that all sample Python scripts here use a Lambda environment variable named **tablename** with a value of **Albums** to specify the name of the DynamoDB table being accessed.

TASK 3.1: CREATING THE SHOW FUNCTION

The following steps will create a **show-albums** Lambda function that meets the following requirements:

* Input: An event sent to the Lambda function with single value for **Genre**, similar to the following:

{

"Genre": "Rock"

}

 The **Genre** value may be an empty string.

* Output: Depends on the value of the incoming **Genre** string — If the incoming **Genre** value is an empty string, scan and return a list of all items in the table. If the incoming **Genre** string has a value, query and return a list of all the albums for a given **Genre**, similar to the following:

[

{

"Rank": 158,

"Album": "Captain Fantastic and the Brown Dirt Cowboy",

"Artist": "Elton John",

"Genre": "Rock",

"Year": 1975

}

{

"Rank": 91,

"Album": "Goodbye Yellow Brick Road",

"Artist": "Elton John",

"Genre": "Rock",

"Year": 1973

}

...

]

1. In the **AWS Management Console**, in the unified search bar, search for and choose

Lambda

.

1. Choose **Create function**.
2. Choose **Author from scratch**, and configure the following in the **Basic information** section:

* For **Function name**, enter

show-albums

* For **Runtime**, choose

Python 3.11

1. Expand  **Change default execution role** , and configure:

* **Execution role** Choose **Use an existing role**
* **Existing role** Choose **xxxx-LambdaRunRole-xxxx**

1. Choose **Create function**.
2. In the **Code source** section, select **lambda\_function.py**, open the context(right-click) menu, and choose **Open**.
3. Delete the existing Python code and replace it with code that will accomplish the functionality necessary to meet the **show-albums** requirements specified earlier in this task.

 To help you, an incomplete Python script for this purpose is provided below. You may copy and insert this code, making sure to replace the ??? sections with appropriate new code. Remember that the **Albums** table has a GSI named **genre-index**.

import json

import boto3

from boto3.dynamodb.conditions import Key, Attr

import os

from botocore.exceptions import ClientError

def lambda\_handler(event, context):

region=boto3.session.Session().region\_name

dynamodb = boto3.resource('dynamodb', region\_name=region) #low-level Client

table = dynamodb.Table(os.environ['tablename']) #define which dynamodb table to access

if len(event['Genre']) > 0:

try:

totallist = table.???(

???

)

except ClientError as error:

return error.response['Error']['Message']

except BaseException as error:

raise error

return totallist['Items']

else :

try:

scanreturn = table.scan()

totallist = scanreturn['Items']

while 'LastEvaluatedKey' in scanreturn.keys(): # if lastevaluatedkey is present, we need to keep scanning

scanreturn = table.scan(

ExclusiveStartKey = scanreturn['LastEvaluatedKey']

)

totallist += scanreturn['Items']

return totallist

except ClientError as error:

return error.response['Error']['Message']

except Exception as error:

print(error)

 For assistance, refer to the documentation found here: [Getting Started with the AWS SDKs (Python and DynamoDB Step 4)](https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/GettingStarted.Python.04.html) and here: [Amazon DynamoDB API Python code examples](https://boto3.amazonaws.com/v1/documentation/api/latest/guide/dynamodb.html). If needed, the complete code is available to you as **ShowAlbums.py** in your AWS Cloud9 environment. Refer to the [Answer Key](https://labs.skillbuilder.aws/sa/lab/arn%3Aaws%3Alearningcontent%3Aus-east-1%3A470679935125%3Ablueprintversion%2FSPL-TF-200-DBDYL6-1%3A1.0.7-b7ad0e80/en-US#AnswerKey) section at the bottom of this lab for the specific location.

1. Choose **Deploy**. You should see a message that says **Changes deployed**.

 Note that the following few steps setup a AWS Lambda environment variable that specifies the name of the DynamoDB table that will be accessed by the function. This is necessary if you used the sample function as the basis for yours.

1. Choose the **Configuration** tab found just above the **Code source** pane to configure the environment variables.
2. In the left navigation pane, choose **Environment variables**.
3. In the **Environment variables** section, choose **Edit**.
4. In the **Edit environment variables** page, configure the following details:

* Choose **Add environment variable**.
* For **Key**, enter

tablename

* For **Value**, enter

Albums

1. Choose **Save**.

 While this is optional, it is a good practice to test a Lambda function before using it with outside applications. These next few steps walk you through a sample test run.

1. Select the **Code** tab to return to the Python script editing area.
2. Choose the **Test** drop-down and select **Configure test event**.
3. In the **Event name** dialog, enter

genretest

.

1. Replace the default text in the coding pane with the sample input you expect the function to receive:

{

"Genre": "Rock"

}

1. Choose **Save**.
2. Choose the **Test** button to start the test.

 The output should return a JSON list of all the albums in the database with the *Rock* genre.

 You may need to troubleshoot your script to ensure proper operation. Also, it is a good practice to try more than one test event. For example, you might try making the “Genre” input an empty string, to verify that the script returns all albums in that case.

TASK 3.2: CREATING THE ADD FUNCTION

Now you will need to create an **add-album** Lambda function that meets the following requirements:

* Input: An event sent to the Lambda function with an album Item object, similar to the following:  **Hint** use this as a test event to check your function

{

"Item": {

"Artist": "ExampleArtist",

"Album": "ExampleAlbum",

"Genre": "ExampleGenre",

"Rank": 100,

"Year": "1995"

}

}

* Output: The response from the service

{

"ResponseMetadata": {

"RequestId": "3PNR7OHFFPB890989Q108N9P63VV4KQNSO5AEMVJF66Q9ASUAAJG",

"HTTPStatusCode": 200,

"HTTPHeaders": {},

"RetryAttempts": 0

}

}

1. Create the **add-album** function. Follow the steps provided for creating the **show-albums** Lambda function.

 To help you, an incomplete Python script for this purpose is provided below. You may copy and insert this code, making sure to replace the ??? sections with appropriate new code.

import json

from pprint import pprint

import boto3

from boto3.dynamodb.conditions import Key, Attr

import time

from decimal import \*

import os

from botocore.exceptions import ClientError

def lambda\_handler(event,context):

region=boto3.session.Session().region\_name

dynamodb = boto3.resource('dynamodb', region\_name=region) #low-level Client

table = dynamodb.Table(os.environ['tablename']) #define which dynamodb table to access

try:

response = table.???(

Item=???["Item"]

)

return response

# handle error responses

except ClientError as error:

return error.response['Error']['Message']

except Exception as error:

print(error)

 For assistance, refer to the documentation found here: [Getting Started with the AWS SDKs (Python and DynamoDB Step 3)](https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/GettingStarted.Python.03.html) and here: [Amazon DynamoDB API Python code examples](https://boto3.amazonaws.com/v1/documentation/api/latest/guide/dynamodb.html). If needed, the complete code is available to you as **AddAlbum.py** in your AWS Cloud9 environment. Refer to the [Answer Key](https://labs.skillbuilder.aws/sa/lab/arn%3Aaws%3Alearningcontent%3Aus-east-1%3A470679935125%3Ablueprintversion%2FSPL-TF-200-DBDYL6-1%3A1.0.7-b7ad0e80/en-US#AnswerKey) section at the bottom of this lab for the specific location.

 Don’t forget to add the **tablename** environment variable!

TASK 3.3: CREATING THE UPDATE FUNCTION

Now you will need to create an **update-album** Lambda function that meets the following requirements:

* Input: An event sent to the Lambda function with an album Item object, similar to the following:  **Hint** use this as a test event to check your function

{

"Item": {

"Artist": "ExampleArtist",

"Album": "ExampleAlbum",

"Genre": "ExampleGenre",

"Rank": 105,

"Year": "1995"

}

}

* Output: The updated attributes

{

"Rank": 105,

"Genre": "ExampleGenre",

"Year": "1995"

}

* Only update attributes that are not part of the primary key schema (Artist & Album.)

1. Create the **update-album** function. Follow the steps provided for creating the **show-albums** Lambda function.

 To help you, an incomplete Python script for this purpose is provided below. You may copy and insert this code, making sure to replace the ??? sections with appropriate new code.

import json

import boto3

from decimal import Decimal

from botocore.exceptions import ClientError

import os

def lambda\_handler(event, context):

region=boto3.session.Session().region\_name

dynamodb = boto3.resource('dynamodb', region\_name=region) #low-level Client

table = dynamodb.Table(os.environ['tablename']) #define which dynamodb table to access

try:

response = table.???(

Key={

'Artist': event["Item"]["Artist"],

'Album': event["Item"]["Album"]

},

???="set Genre=:g, #R=:r, #Y=:y",

??? = { '#R' : "Rank", '#Y' : "Year" },

???={

':r': Decimal(event["Item"]["Rank"]),

':g': event["Item"]["Genre"],

':y': event["Item"]["Year"]

},

ReturnValues="UPDATED\_NEW"

)

return response['Attributes']

# handle error responses

except ClientError as error:

return ClientError

except Exception as error:

print(error)

 For assistance, refer to the documentation found here: [Getting Started with the AWS SDKs (Python and DynamoDB Step 3)](https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/GettingStarted.Python.03.html) and here: [Amazon DynamoDB API Python code examples](https://boto3.amazonaws.com/v1/documentation/api/latest/guide/dynamodb.html). If needed, the complete code is available to you as **UpdateAlbumAttributes.py** in your AWS Cloud9 environment. Refer to the [Answer Key](https://labs.skillbuilder.aws/sa/lab/arn%3Aaws%3Alearningcontent%3Aus-east-1%3A470679935125%3Ablueprintversion%2FSPL-TF-200-DBDYL6-1%3A1.0.7-b7ad0e80/en-US#AnswerKey) section at the bottom of this lab for the specific location.

 Don’t forget to add the **tablename** environment variable!

TASK 3.4: CREATING THE DELETE FUNCTION

Now you will need to create an **delete-album** Lambda function that meets the following requirements:

* Input: An event sent to the Lambda function with a specified Artist and Album, similar to the following:  **Hint** use this as a test event to check your function

{

"Artist":"ExampleArtist",

"Album":"ExampleAlbum"

}

* Output: The response from the service

{

"ResponseMetadata": {

"RequestId": "V39D7VGL6OL8LBD42NO01S7HIBVV4KQNSO5AEMVJF66Q9ASUAAJG",

"HTTPStatusCode": 200,

"HTTPHeaders": {},

"RetryAttempts": 0

}

* If the given album’s **Rank** is greater then 100 or does not exist, the item gets deleted.
* If the given album’s **Rank** is less than or equal to 100, the item is not deleted. A message is returned indicating the conditional operation has failed.

1. Create the **delete-album** function. Follow the steps provided for creating the **show-albums** Lambda function.

 To help you, an incomplete Python script for this purpose is provided below. You may copy and insert this code, making sure to replace the ??? sections with appropriate new code.

import json

from pprint import pprint

import boto3

from boto3.dynamodb.conditions import Key, Attr

import time

from decimal import \*

from botocore.exceptions import ClientError

import os

def lambda\_handler(event,context):

region=boto3.session.Session().region\_name

dynamodb = boto3.resource('dynamodb', region\_name=region) #low-level Client

table = dynamodb.Table(os.environ['tablename']) #define which dynamodb table to access

try:

delstatus = table.delete\_item( # perform delete

Key={

'Artist': event["Artist"],

'Album': event["Album"]

},

???

)

return delstatus

except ClientError as error:

return error.response['Error']['Message']

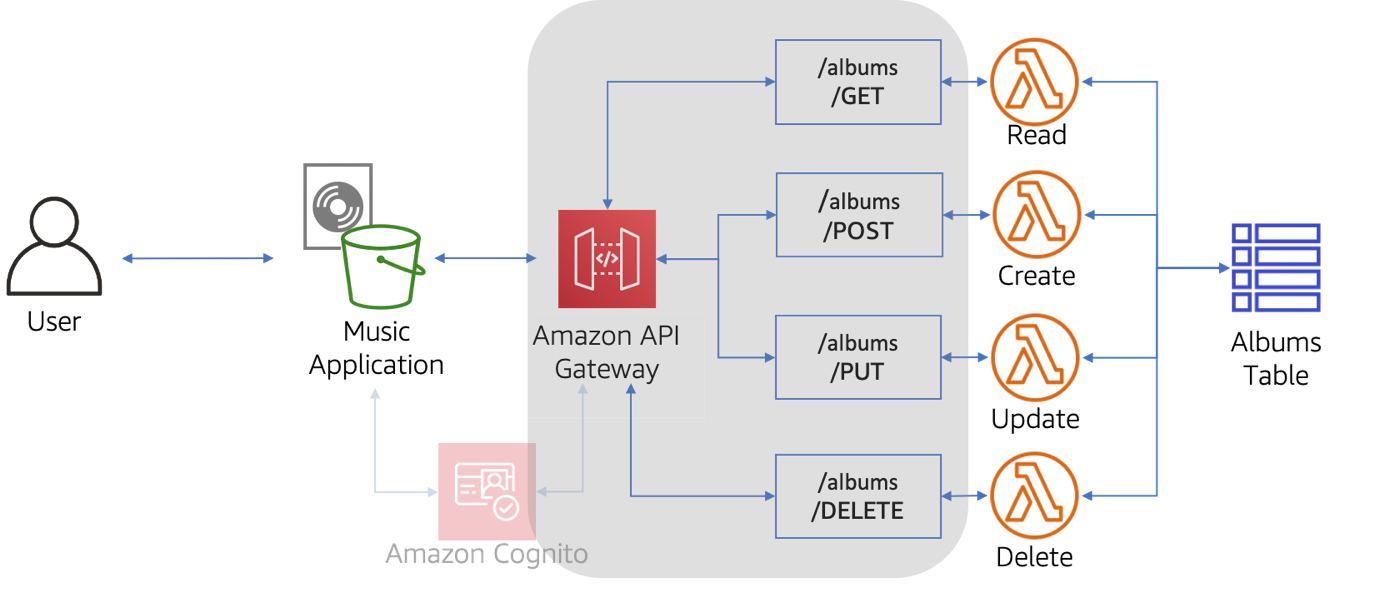
except Exception as error:

print(error)

 For assistance, refer to the documentation found here: [Getting Started with the AWS SDKs (Python and DynamoDB Step 3)](https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/GettingStarted.Python.03.html) and here: [Amazon DynamoDB API Python code examples](https://boto3.amazonaws.com/v1/documentation/api/latest/guide/dynamodb.html). If needed, the complete code is available to you as **DeleteAlbum.py** in your AWS Cloud9 environment. Refer to the [Answer Key](https://labs.skillbuilder.aws/sa/lab/arn%3Aaws%3Alearningcontent%3Aus-east-1%3A470679935125%3Ablueprintversion%2FSPL-TF-200-DBDYL6-1%3A1.0.7-b7ad0e80/en-US#AnswerKey) section at the bottom of this lab for the specific location.

 Don’t forget to add the **tablename** environment variable!

**Task 4: Creating, configuring, and deploying an API using AWS API Gateway**



In this task, you will create the API and methods to communicate between the music application and the Lambda functions that access the DynamoDB table.

TASK 4.1: CREATING THE API

1. In the **AWS Management Console**, in the unified search bar, search for and choose

API Gateway

.

1. In the **Choose an API type** section, choose **Build** from the **Rest API** block.

 There is a similar option labeled as **Rest API Private**. Please be mindful not to select this option.

1. Under **API details**, select **New API**.
2. Enter the following details:

* For **API name**, enter

MusicAPI

* For **Description**, enter

MusicAPI

* For **Endpoint Type**, select

Regional

1. Choose **Create API**.
2. In the left navigation pane, choose **Resources** if not already selected.
3. Under **Resources**, select **/**.
4. In the **Resources** pane, select **Create Resource**.
5. In the **Resource details** section, for the **Resource Name**, enter the following:

albums

 The **Resource Path** will automatically populate with **albums**.

1. Choose **Create resource**.

With the resource created, you can now start defining methods on the **albums** resource.

TASK 4.2: CREATING THE GET METHOD

1. If not already selected, choose the **Resources** link and select **/albums**.
2. From the **Methods** section, select **Create method**.

A blank dropdown menu will appear for **Method type**.

1. Select **GET** from the blank dropdown menu.
2. For **Integration type**, select  **Lambda Function**.
3. For **Lambda Function**, type a lowercase

s

, then select your **show-albums** function.

1. Choose **Create method**.

The **API Gateway** console will navigate you the **GET - Method Execution** flowchart.

There are a couple more things you need to do before you are ready to deploy and test your newly created **GET** method. First, you need to instruct the API to pass in a value for **Genre** from a query parameter to your Lambda function’s event object. You will accomplish this with a mapping template.

1. Choose **Integration Request**.
2. In **Integration request settings** section, select **Edit**.
3. Expand the  **Mapping templates** section (you may need to scroll down) and configure the following:

* Select **Add mapping template**.
* For **Content-Type**, enter

application/json

 in the text field (*you will need to enter this even though it is already displayed*).

1. In the text editor below **Generate template**, enter the following:

{

"Genre": "$input.params('Genre')"

}

 This template instructs the API to take a value from a query parameter and transforms it into JSON, which your Lambda function can then consume via the event object.

1. Choose **Save**.

As a final step in the configuration, you need to enable cross-origin resource sharing (CORS) on your **/albums** resource. CORS enables cross-origin HTTP requests that are initiated from scripts running in the browser. Since your REST API will receive non-simple cross-origin HTTP requests via the music application, you need to enable CORS.

 For more information, refer to *Enabling CORS for a REST API resource* in the [Additional Resources](https://labs.skillbuilder.aws/sa/lab/arn%3Aaws%3Alearningcontent%3Aus-east-1%3A470679935125%3Ablueprintversion%2FSPL-TF-200-DBDYL6-1%3A1.0.7-b7ad0e80/en-US#AdditionalResources) section.

1. Choose the **/albums** resource.
2. In **Resource details** section, select **Enable CORS**.
3. For **Gateway responses**, select the following options:

* **Default 4XX**
* **Default 5XX**

1. For **Access-Control-Allow-Methods**, select the following options:

* **GET**
* **OPTIONS**

1. Select **Save**.

With CORS enabled, you are ready to deploy your API.

1. Choose **Deploy API**, then configure the following options:

* **Stage**:

\*New Stage\*

* **Stage name**:

dev

1. Choose **Deploy**.
2. Copy the **Invoke URL** value into a separate file or clipboard manager. You will be referencing this value throughout the lab.

 While this is optional, it is a good practice to test an API before using it with outside applications. These next few steps walk you through using curl functionality to test the API.

1. Leave the **API Gateway** console browser tab open and navigate back over to your AWS Cloud9 environment.
2. Replace the

{ChangeMeInvokeURL}

 in the following command with the **Invoke URL** you saved earlier. Run the edited command to test your API.

curl {ChangeMeInvokeURL}/albums

 The output should contain a list of all the albums in the **Albums** table, similar to following:

[{"Rank": 158, "Album": "Captain Fantastic and the Brown Dirt Cowboy", "Artist": "Elton John", "Genre": "Rock", "Year": 1975}...

The **GET /albums** endpoint is working as expected. The API is serving as a proxy to the **show-albums** function and returning the expected result. The only thing left to test is your mapping template.

1. Replace the

{ChangeMeInvokeURL}

 in the following command with the **Invoke URL** you saved earlier. Run the edited command in the terminal window to test your API with the query functionality.

curl {ChangeMeInvokeURL}/albums?Genre=Pop

 The output should contain a list of all the albums in the **Albums** where the **Genre** attribute is

Pop

, similar to following:

[{"Rank": 30, "Album": "Blue", "Year": 1971, "Artist": "Joni Mitchell", "Genre": "Pop"}]

TASK 4.3: CREATING THE POST METHOD

To create the POST method, you will reuse most of the steps performed to create the GET method, only using POST instead.

1. Navigate to the API Gateway browser tab.
2. Choose the **Resources** link and select **/albums**.
3. From **Methods** section, select **Create method**.
4. For **Method type**, select **POST** from the dropdown menu.
5. For **Integration type**, select  **Lambda Function**.
6. For **Lambda Function**, type a lowercase

a

, then select your **add-album** function.

1. Choose **Create method**

The **API Gateway** console will navigate you to the **POST - Method Execution** flowchart.

1. Choose the **/albums** resource.
2. In **Resource details** section, select **Enable CORS**.
3. For **Gateway responses**, select the following options:

* **Default 4XX**
* **Default 5XX**

1. For **Access-Control-Allow-Methods**, select the following options:

* **GET**
* **OPTIONS**
* **POST**

1. Select **Save**.
2. Deploy the API using the same steps as shown during the GET task. Do not create a new stage of deployment. Re-use the “dev” stage.

 While this is optional, it is a good practice to test an API before using it with outside applications. These next few steps walk you through using curl functionality to test the API.

 If you see an error such as “Missing Authentication Token” while trying the subsequent curl commands, ensure you’ve properly enabled CORS, re-deployed the API, and also wait half a minute for the changes to take effect.

1. Leave the **API Gateway** console browser tab open and navigate back over to your AWS Cloud9 environment.
2. Replace the

{ChangeMeInvokeURL}

 in the following command with the **Invoke URL** you saved earlier. Run the edited command in the terminal window. This command will add a new album to the database.

curl -X POST -d '{"Item": {"Artist":"NewTalent", "Album":"CurlLabTest1", "Genre":"LabTest"}}' {ChangeMeInvokeURL}/albums

 The output should return some basic Metadata, similar to the following:

{"ResponseMetadata": {"RequestId": "R4FCIN3TC6DK8B0J51PPCGE9CVVV4KQNSO5AEMVJF66Q9ASUAAJG", "HTTPStatusCode": 200, "HTTPHeaders": {"server": "Server", "date": "Wed, 12 May 2021 19:03:01 GMT", "content-type..."}

1. Test the addition of the new item by using the GET method similarly to the previous task:

curl {ChangeMeInvokeURL}/albums?Genre=LabTest

 The output should contain the item just added to the database, similar to the following:

{"Items": [{"Album": "CurlLabTest1", "Artist": "NewTalent", "Genre": "LabTest"}

TASK 4.4: CREATING THE PUT METHOD

For this method, you will repeat the actions performed to create the POST method, only using the PUT method with the **update-album** Lambda function.

1. Use the AWS console to create a PUT method on the MusicAPI API that integrates the **update-album** Lambda function.
2. Choose the **/albums** resource.
3. In **Resource details** section, select **Enable CORS**.
4. For **Gateway responses**, select the following options:

* **Default 4XX**
* **Default 5XX**

1. For **Access-Control-Allow-Methods**, select the following options:

* **GET**
* **OPTIONS**
* **POST**
* **PUT**

1. Select **Save**.
2. Re-deploy the API.

 Optional: Test the PUT method with these steps:

1. Leave the **API Gateway** console browser tab open and navigate back over to your AWS Cloud9 environment.
2. Replace the

{ChangeMeInvokeURL}

 in the following command with the **Invoke URL** you saved earlier. Run the edited command in the terminal window. This command will update the value of the year attribute of this database item.

curl -X PUT -d '{"Item": {"Artist":"NewTalent", "Album":"CurlLabTest1", "Genre":"NewLabTestSound", "Rank": 555, "Year": 2021}}' {ChangeMeInvokeURL}/albums

 The output should return the attributes updated (along with Metadata), similar to the following:

{"Attributes": {"Rank": 555, "Genre": "NewLabTestSound", "Year": 2021}

1. Test the update of the item attribute by using the GET method similarly to the previous task:

curl {ChangeMeInvokeURL}/albums?Genre=NewLabTestSound

TASK 4.5: CREATING THE DELETE METHOD

For this method, you will repeat the actions performed to create the DELETE method, only using the DELETE method with the **delete-album** Lambda function.

1. Use the AWS console to create a DELETE method on the MusicAPI API that integrates the **delete-album** Lambda function.
2. Choose the **/albums** resource.
3. In **Resource details** section, select **Enable CORS**.
4. For **Gateway responses**, select the following options:

* **Default 4XX**
* **Default 5XX**

1. For **Access-Control-Allow-Methods**, select the following options:

* **DELETE**
* **GET**
* **OPTIONS**
* **POST**
* **PUT**

1. Select **Save**.
2. Re-deploy the API.

 Optional: Test the DELETE method with these steps:

1. Leave the **API Gateway** console browser tab open and navigate back over to your AWS Cloud9 environment.
2. Replace the

{ChangeMeInvokeURL}

 in the following command with the **Invoke URL** you saved earlier. Run the edited command in the terminal window. This command will delete the specified item from the database.

curl -X DELETE -d '{"Artist":"NewTalent", "Album":"CurlLabTest1"}' {ChangeMeInvokeURL}/albums

 The output should return some Metadata, similar to the following:

{"ResponseMetadata": {"RequestId": "D2DMNG53G487M34P93LI53293BVV4KQNSO5AEMVJF66Q9ASUAAJG", "HTTPStatusCode": 200, "HTTPHeaders": {"server": "Server", "date": "Wed, 12 May 2021 20:01:45 GMT", "content-type": "application/x-amz-json-1.0", "content-length": "2", "connection": "keep-alive", "x-amzn-requestid": "D2DMNG53G487M34P93LI53293BVV4KQNSO5AEMVJF66Q9ASUAAJG", "x-amz-crc32": "2745614147"}, "RetryAttempts": 0}

1. Test the deletion of the new item by using the GET method similarly to the previous task:

curl {ChangeMeInvokeURL}/albums?Genre=NewLabTestSound

 This time the output should return an empty Items list.

**Task 5: Integrating the backend with the music application**

You are now ready to integrate the music application with the backend (Lambda functions and the DynamoDB table) you have created. In this task, you will make changes to the music application to use the API created earlier.

TASK 5.1: CONFIGURING THE MUSIC APPLICATION

The first thing you will do is modify the front-end’s configuration file so that it can consume your newly created API.

1. Navigate to the AWS Cloud9 environment.
2. In the left navigation pane, choose **lab-files**, and then choose **web-app**.
3. Right-click the config.js file and choose Open
4. In the AWS Cloud9 editor, edit the line with

ApiGatewayStageUrl

 as shown below, only replacing

{ChangeMe-InvokeURL}

 with the value of **Invoke URL** you noted earlier.

var ApiGatewayStageUrl = '{ChangeMe-InvokeURL}';

1. Save your changes by choosing the **File** menu and selecting **Save**
2. In the terminal window, navigate to the

web-app

 directory with the following command:

cd ~/environment/lab-files/web-app

1. Replace

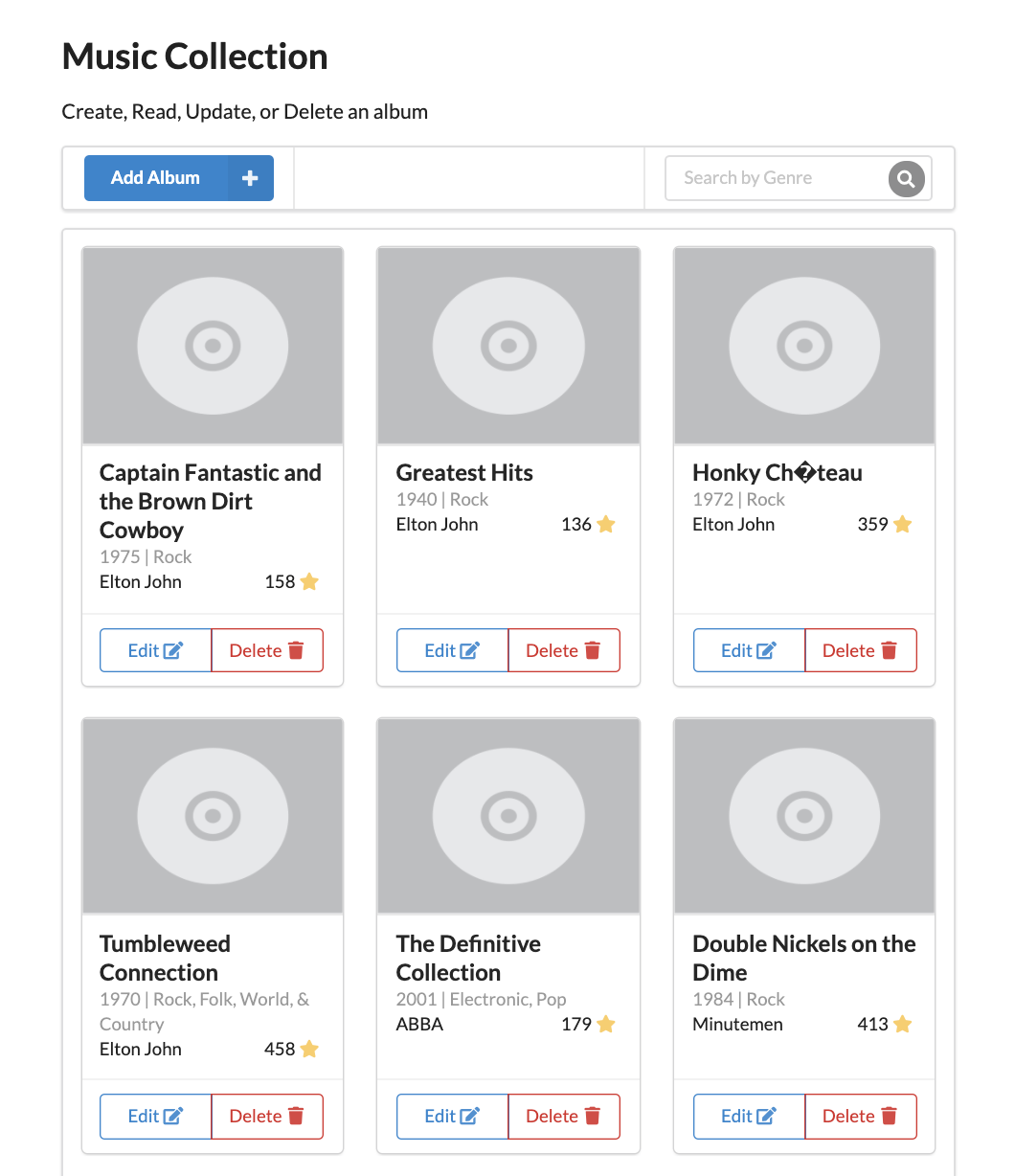
{ChangeMeS3Bucket}

 in the following command with the value of **S3BucketName** listed to the left of these instructions, then run the edited command to upload the modified file to S3.

aws s3 cp config.js s3://{ChangeMeS3Bucket}

1. Navigate to the browser tab with the music application to test the updated configuration file.
2. Refresh the browser.

The music application is now loading all of the albums from the **Albums** table, having been integrated with the API you created.



TASK 5.2: TESTING THE MUSIC APPLICATION

Now that the music application is configured correctly, it is time to test the end-to-end integration with your backend.

1. Choose **Add Album**.
2. In the **Add Album** dialog box, enter the following:

* Artist:

Test

* Album:

Test

* Genre:

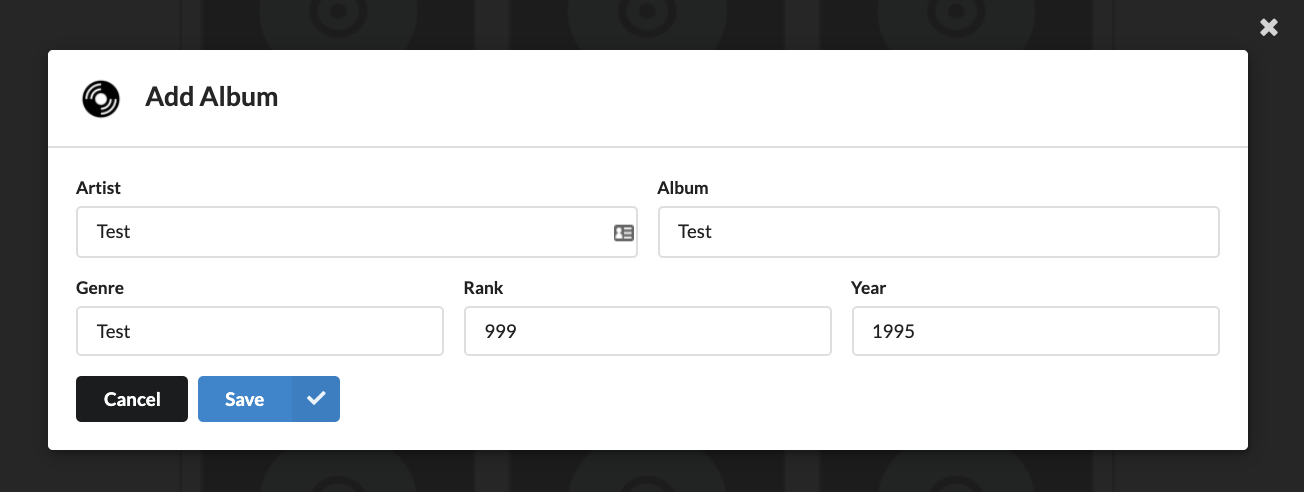
Test

* Rank:

999

* Year:

1995



1. Choose **Save**.

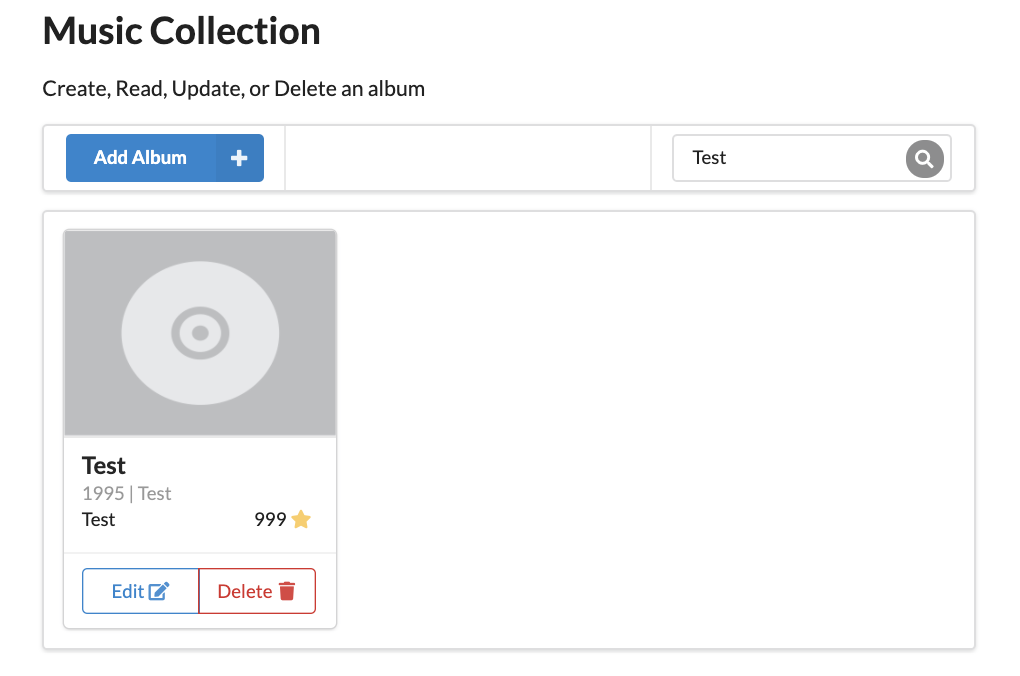
The music application will reload. This action is using the **POST /albums** method of the API you created to add an item to the database.

Now search for the newly created album by using the search feature.

1. In the **Search By Genre** textfield, enter

Test

 and choose



This search feature is reaching out to your **GET /albums** method and providing the **Genre** as a query parameter. Behind the scenes, that query parameter is then passed on to your **show-albums** function.

Now that you’ve verified that both the **GET** and **POST** endpoints are working as expected, its time to test the **PUT** method.

1. On the

Test

 album card, choose Edit .

1. Change the value for **Year** to

2000

1. Choose **Save**.

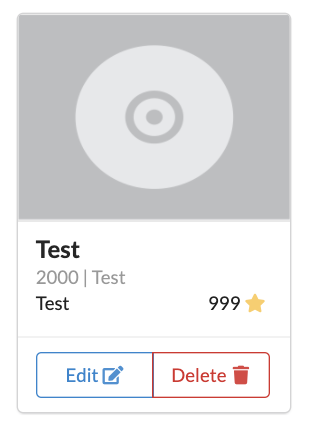
Once again, the music application will reload.

1. Search for the album by entering

Test

 in the search box and choosing

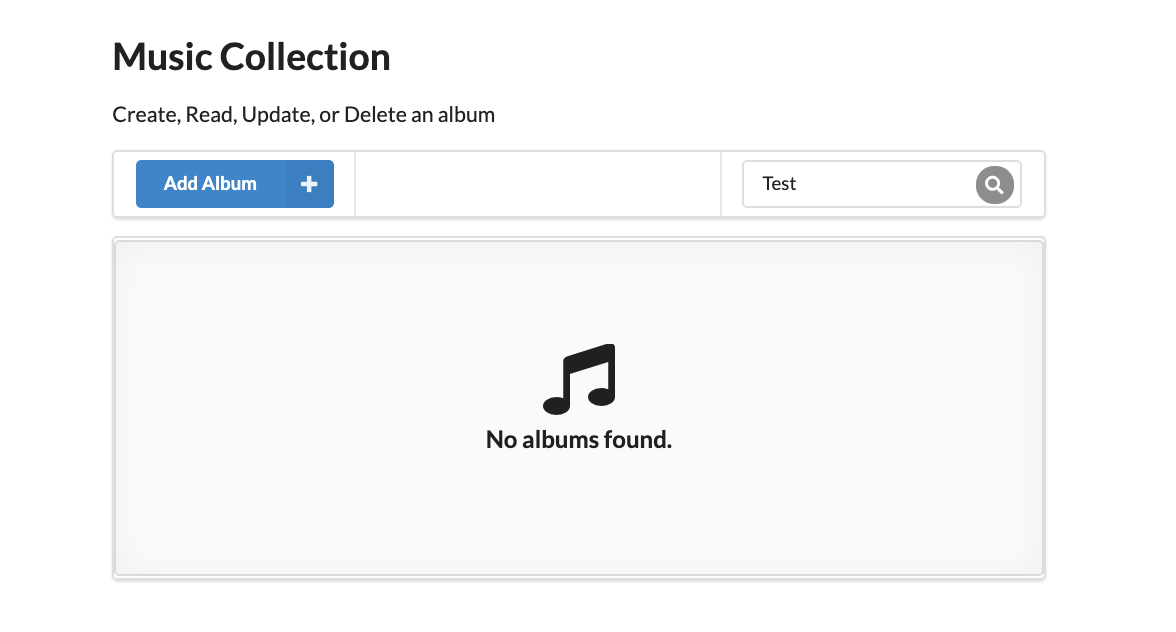
1. Notice that the value for the **Year** attribute has been updated.



1. To test the DELETE method, choose Delete .
2. After the music application reloads, search for the

Test

 genre once again.



The album that you have created for testing purposes no longer appears, indicated that it has been successfully deleted from the **Albums** table.

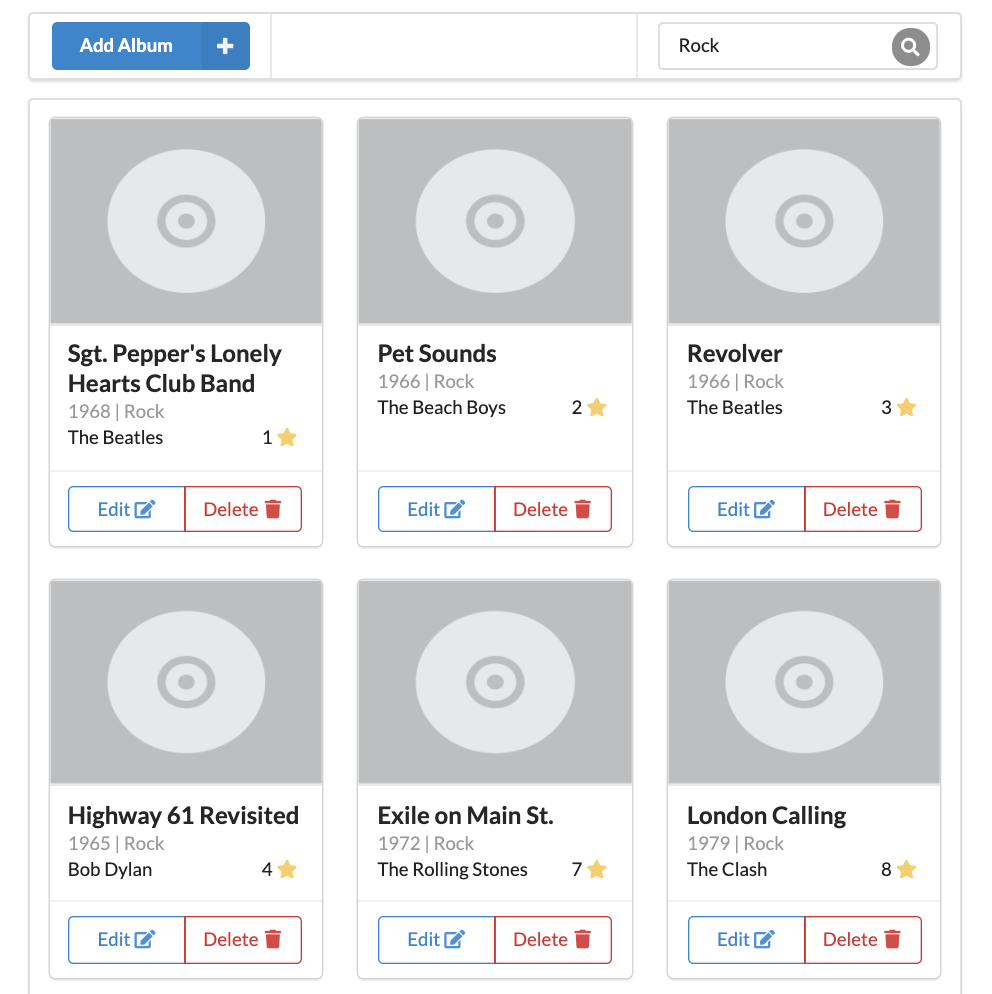
1. Filter the list of albums for the

Rock

 genre by entering

Rock

 into the search box and choosing



1. Try deleting an album with a **Rank** of 100 or less (the value next to the star symbol), by choosing Delete  on the corresponding album card.

The operation will not succeed due to the conditional expression you implemented earlier in the **delete-album** function.

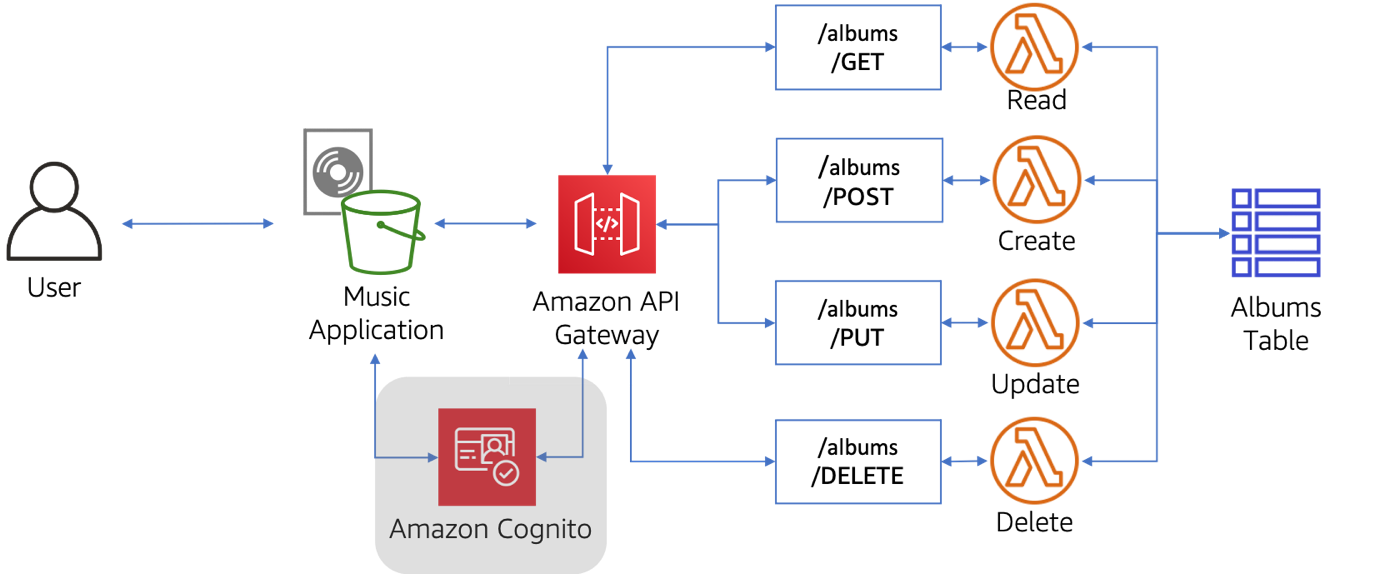
**Task 6: Challenge activity – adding authentication via Amazon Cognito**

If you have time, try this challenge activity that adds a layer of security to your online application. Currently, anyone with the URL to the music application can access all of its capabilities. For certain use-cases this may be fine, but for the music application you only want authenticated users leveraging its features.

Instead of creating and managing your own user directory service, you have decided to leverage Amazon Cognito user pools. A user pool is a user directory in Amazon Cognito. With a user pool, your users can sign in to your web or mobile app through Amazon Cognito. For more information, refer to *Amazon Cognito User Pools* in the [Additional Resources](https://labs.skillbuilder.aws/sa/lab/arn%3Aaws%3Alearningcontent%3Aus-east-1%3A470679935125%3Ablueprintversion%2FSPL-TF-200-DBDYL6-1%3A1.0.7-b7ad0e80/en-US#AdditionalResources) section.

In this task, you will:

* Create an Amazon Cognito user pool, which you are going to use to authenticate your end users and control access to the API, making sure that your endpoint is secure.
* Create and confirm an Amazon Cognito user. You will use this user to test the end to end application.



TASK 6.1: CREATING AN AMAZON COGNITO USER POOL

1. At the top of the AWS Management Console, to the right of the **Services** menu, in the search bar, search for

Cognito

 and then choose that service from the list.

1. Choose **Create user pool**

**Note:** You may safely ignore the error message *[AccessDeniedException] Failed to fetch ACM certificates*.

1. In the **Configure sign-in experience** step, complete the following settings for **Authentication providers**:

* **Provider types**: Cognito user pool
* **Cognito user pool sign-in options**: User name
* **User name requirements**: Allow users to sign in with a preferred user name

1. Choose **Next**
2. In the **Configure security requirements** step, complete the following settings:

* In the **Password policy** section, select **Custom**.
* **Password minimum length**: 8 by default
* **Password requirements**: clear the check box for the following options: - **Contains at least 1 number** - **Contains at least 1 special character** - **Contains at least 1 uppercase letter** - **Contains at least 1 lowercase letter**
* In the **Multi-factor authentication** section, select **MFA enforcement**: No MFA
* In the **User account recovery** section, unselect the default option.

**CAUTION:** Selecting **No MFA** is not best security practice. You are only choosing this option for the sake of this lab demonstration. Using MFA in a real world use case is best practice.

1. Choose **Next**
2. In the **Configure sign-up experience** step, complete the following settings:

* In the **Verifying attribute changes** section for **Keep original attribute value active when an update is pending - Recommended** deselect the checkbox. Leave the remaining settings as is.

1. Choose **Next**
2. In the **Configure message delivery** step, complete the following settings:

* Under **Email**, in the **Email provider** section, choose **Send email with Cognito**.
* Leave the remaining settings as is.

1. Choose **Next**
2. In the **Integrate your app** step, complete the following settings:

* In the **User pool name** section, enter

MusicAppPool

 for **User pool name**.

* In the **Initial app client** section, enter

MusicAppClient

 for **App client name**.

* In **Client secret**, Don’t generate a client secret is selected by default.

1. Choose **Next**.
2. Choose **Create user pool**.
3. Copy the **User pool ID** into a separate file or clipboard manager. You will reference this in a later step.
4. Choose the link for **MusicAppPool**.
5. Choose the **App Integration** tab.
6. Scroll down to the **App client list** section.
7. Copy the **Client ID** in the same manner that you copied the **User pool ID**.

Example Values:

* **User pool ID:** us-west-2\_aAjVJjJHr
* **Client ID:** 3ao6g7eq5soc7baqna6ks5l9mv

TASK 6.2: CREATING A USER IN THE AMAZON COGNITO USER POOL

You will create and confirm an Amazon Cognito user by using the AWS CLI.

1. Navigate to the AWS Cloud9 environment, leaving the **Amazon Cognito** console tab open.
2. Replace

{ChangeMe-ClientID}

 with your **Client ID** in the following command, then run the adjusted command in the AWS Cloud9 terminal.

aws cognito-idp sign-up --client-id {ChangeMe-ClientID} --username testuser --password testuser

 Expected output:

{

"UserConfirmed": false,

"UserSub": "abc70c2f-52v7-4abc-c195-012abc8560ab"

}

 The command added a new user to the user pool with the username

testuser

 and a password of

testuser

. For more information on Amazon Cognito CLI commands, refer to *Amazon Cognito Command Reference* in the [Additional Resources](https://labs.skillbuilder.aws/sa/lab/arn%3Aaws%3Alearningcontent%3Aus-east-1%3A470679935125%3Ablueprintversion%2FSPL-TF-200-DBDYL6-1%3A1.0.7-b7ad0e80/en-US#AdditionalResources) section.

1. Replace

{ChangeMe-UserPoolID}

 with the value of the **User Pool ID** that you noted earlier. Run the adjusted command to confirm the user that you created.

aws cognito-idp admin-confirm-sign-up --user-pool-id {ChangeMe-UserPoolID} --username testuser

1. Navigate to the **Amazon Cognito** browser tab.
2. In the left navigation pane, choose **User pools**.
3. Choose the **MusicAppPool**.

You should see the username testuser and the that the status is **CONFIRMED**. If you do not see it, choose the refresh icon in the top-right corner.

 Usually, the sign-up process would be done at the application level. The music application does not support this, so you created the user manually.

TASK 6.3: CREATING THE AMAZON COGNITO AUTHORIZER

Your Amazon Cognito user pool is ready to be used. You need to update your API’s configuration to use it by adding an Amazon Cognito Authorizer. The Authorizer will allow you to control access to your API with the **MusicAppPool**.

1. In the **AWS Management Console**, in the unified search bar, search for and choose

API Gateway

.

1. Choose MusicAPI.
2. In the left navigation menu, choose **Authorizers**.
3. Choose **Create authorizer**.
4. Update the details as follows:

* For **Authorizer name**, enter

MusicAppPool

* For **Authorizer type**, select  Cognito
* For **Cognito user pool**, Choose **MusicAppPool**
* For **Token source**, enter

Authorization

* For **Token validation**, leave it empty.

1. Choose **Create authorizer**.

TASK 6.4: ENABLING AUTHORIZATION ON THE API ENDPOINTS

Now that the Amazon Cognito authorizer is created, you need to enable it on your API methods.

1. In the left navigation pane, choose **Resources**.
2. In the **Resources** tree, choose **GET**.
3. Inside the flowchart, choose **Method request**.
4. In the **Method request settings** section, choose **Edit**.
5. For **Authorization**, choose **MusicAppPool**.
6. Choose **Save**.

You will now repeat the above steps for the other methods.

1. Repeat the steps used for the GET method to add authorization to the POST, DELETE, and PUT methods.
2. Choose the **/albums** resource.
3. Choose **Deploy API**.
4. In the **Deploy API** dialog box, for **Stage** choose

dev

1. Choose **Deploy**.

TASK 6.5: APPLICATION INTEGRATION

You are now ready to use the music application with the Amazon Cognito user pool.

1. Navigate to the AWS Cloud9 environment.
2. In the left navigation pane, choose **lab-files**, and then choose **web-app**.
3. Right-click the config.js file and choose Open
4. In the AWS Cloud9 editor, in the

var poolData

 section, add the values for the **User Pool ID** and **Client ID** you noted earlier. The updated file should look similar to the following:

var poolData = {

UserPoolId: 'us-west-2\_aAjVJjJHr',

ClientId: '3ao6g7eq5soc7baqna6ks5l9mv'

};

1. Save your changes by choosing the **File** menu and selecting **Save**
2. Replace

{ChangeMeS3Bucket}

 in the following command with the value of **S3BucketName** listed to the left of these instructions. Run the adjusted command in the AWS Cloud9 terminal window to re-upload the

config.js

 file to S3.

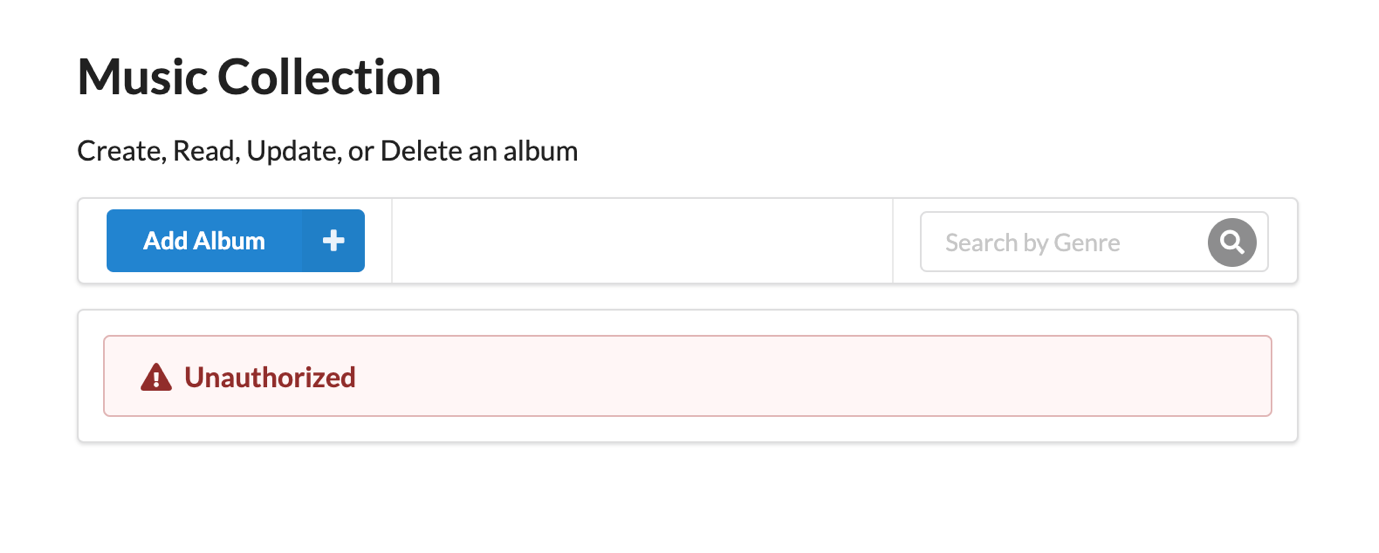
aws s3 cp config.js s3://{ChangeMeS3Bucket}

 You will need to run this command in the

web-app

 directory.

1. Navigate to the browser tab with the music application.
2. Refresh the browser, and notice that the music application is now displaying an authorization error or network error similar to this:

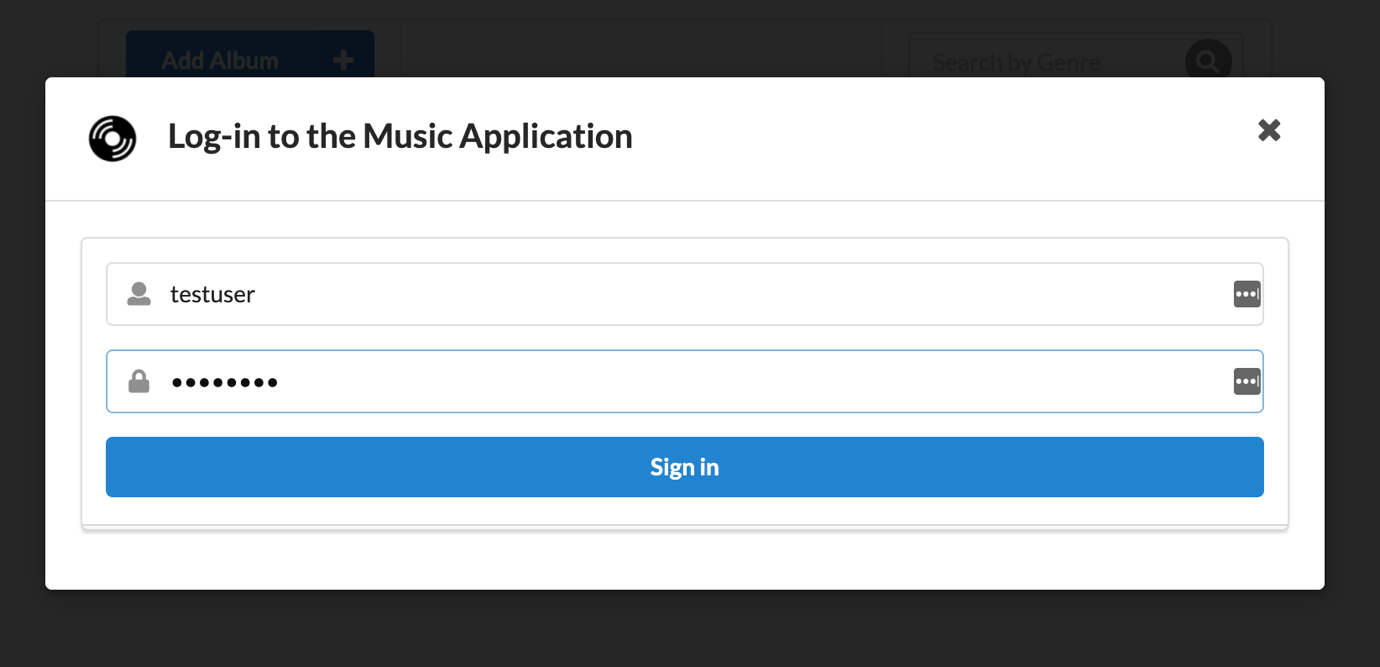


You are now ready to test the end-to-end flow for the authorization that you have configured.

1. Choose the **Sign in** button in the top-right corner of the music application (If it is showing “Sign out”, choose to sign out then choose “Sign in” when it appears)
2. In the login modal, enter

testuser

 for both the username and password fields.



1. Choose **Sign in**.
2. The page will reload, and you should now see the list of all albums.

Since you have logged in with the

testuser

 user, the music application is sending the token that Amazon Cognito generated upon log-in with every request to the **MusicAPI**. Behind the scenes, the API reaches out to Amazon Congito to validate each token that it receives. If the token is valid the request is then authorized.

1. **Optional** Test the other operations (POST, PUT, DELETE) supported by the music application.

**Conclusion**

 Congratulations! You now have successfully:

* Created and deployed Lambda functions that perform operations on a DynamoDB table.
* Leveraged DynamoDB conditional expressions.
* Created and Deployed an API Gateway to proxy Lambda functions.
* Created and Configured Amazon Cognito user pools for authentication.

**Additional Resources**

* [What is Amazon S3?](https://docs.aws.amazon.com/AmazonS3/latest/userguide/Welcome.html)
* [Installing, updating, and uninstalling the AWS CLI](https://docs.aws.amazon.com/cli/latest/userguide/cli-chap-install.html)
* [AWS CLI Command Reference for DynamoDB](https://awscli.amazonaws.com/v2/documentation/api/latest/reference/dynamodb/index.html)
* [Getting Started with DynamoDB and AWS SDKs](https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/GettingStarted.html)
* [Amazon DynamoDB API Reference](https://docs.aws.amazon.com/amazondynamodb/latest/APIReference/API_Operations_Amazon_DynamoDB.html)
* [Amazon DynamoDB API Python code examples](https://boto3.amazonaws.com/v1/documentation/api/latest/guide/dynamodb.html)
* [Enabling CORS for a REST API resource](https://docs.aws.amazon.com/apigateway/latest/developerguide/how-to-cors.html)
* [Amazon Cognito user pools](https://docs.aws.amazon.com/cognito/latest/developerguide/cognito-user-identity-pools.html)
* [Amazon Cognito Command Reference](https://awscli.amazonaws.com/v2/documentation/api/latest/reference/cognito-idp/index.html)

**Answer key**

You can find sample Python scripts, both incomplete and completed, in the lab-files/sample-lambda-functions folder in the AWS Cloud9 instance of the lab.

**End lab**

Follow these steps to close the console and end your lab.

1. Return to the **AWS Management Console**.
2. At the upper-right corner of the page, choose **AWSLabsUser**, and then choose **Sign out**.
3. Choose **End lab** and then confirm that you want to end your lab.

For more information about AWS Training and Certification, see [*https://aws.amazon.com/training/*](https://aws.amazon.com/training/).

*Your feedback is welcome and appreciated.*  
If you would like to share any feedback, suggestions, or corrections, please provide the details in our [*AWS Training and Certification Contact Form*](https://support.aws.amazon.com/#/contacts/aws-training).