Moving to Amazon API Gateway

**SPL-BE-200-DVMTAG-1 - Version 1.0.3**

© 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 demonstrates how to create the Amazon DynamoDB table and the Amazon API Gateway hosted API that’s used by the application.

The lab starts by guiding you through the process of creating a DynamoDB table that will be used to store the uniqueGridId data sent to the API. API Gateway acts as a frontend for the Lambda functions. You then update the Lambda function to store the mapping from uniqueGridId to an Amazon Simple Storage (Amazon S3) object by using a PUT operation.

After the AWS Lambda functions are updated, you deploy the application. Then, you create the API Gateway resource and add routes to the Lambda functions to copy images and create the grid image. API Gateway can then act as an initiator for the Lambda functions.

Finally, the lab guides you through the process of testing the functionality of the API by invoking it through API Gateway and verifying the results. You can observe how API Gateway activates the Lambda functions. You can also see how the Lambda functions perform the intended operations on the S3 bucket by creating a new grid image, creating an S3 presigned URL, and populating uniqueGridId data in the DynamoDB table.

OBJECTIVES

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

* Create a DynamoDB table.
* Update the application to save the mapping from uniqueGridId to an S3 object by using dynamodb.put\_item.
* Deploy the application.
* Create an API by using API Gateway.
* Run the API to create the grid image and an S3 presigned URL.

TECHNICAL KNOWLEDGE PREREQUISITES

To successfully complete this lab:

* You should be familiar with basic navigation of the AWS Management Console.
* You should be comfortable editing and running scripts by using an AWS Cloud9 code editor and terminal.
* You should have a basic understanding of and familiarity with Amazon S3.
* You should have a basic understanding of and familiarity with Amazon API Gateway.

DURATION

This lab requires *60* minutes to complete.

ICON KEY

Various icons are used throughout this lab to call attention to different types of instructions and notes. The following list explains the purpose for each icon:

* **Command:** A command that you must run.
* **Expected output:** A sample output that you can use to verify the output of a command or edited file.
* **Note:** A hint, tip, or important guidance.
* **Consider:** A moment to pause to consider how you might apply a concept in your own environment or to initiate a conversation about the topic at hand.

**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: Create a DynamoDB table**

In this task, you connect to the AWS Cloud9 environment and the run the AWS CLI command to create a DynamoDB table. The table stores the grid image IDs and information, in addition to the S3 presigned URL that you create later in the lab.

TASK 1.1: CONNECT TO AWS CLOUD9

In this task, you connect to the AWS Cloud9 environment that’s provisioned as part of this lab.

1. From the **Lab Information** section to the left of these instructions, copy the **Cloud9Environment** URL link and in a new browser tab, paste the URL.

The browser takes you to the AWS Cloud9 environment that you use during this lab.

You do not need the **Cloud9 Welcome screen** or any of the other default tabs that appear when you first launch **AWS Cloud9**.

1. To close each tab, choose the **X**.

This section of the IDE is where you update various files throughout this lab.

**Consider:** You are working in another AWS Cloud9 environment that’s similar to the previous lab. The only difference is the application files that you see in the file tree. If you need a refresher, take a moment to familiarize yourself with the **AWS Cloud9** IDE interface by expanding the *AWS Cloud9 review* section.

**AWS Cloud9 review**

TASK 1.2: CREATE A DYNAMODB TABLE

In this task, you create the DynamoDB table that stores the individual image’s gridIds, which are used to create the final grid image.

A **terminal pane** is at the bottom of the IDE. You can expand it up halfway to have more visibility when you run commands. You can also close it and open a new terminal session from the top menu. (To open a new terminal session, choose the  icon and choose **New terminal**.)

1. To create a **DynamoDB table** named **GridBuilder**, run the following AWS Command Line Interface (AWS CLI) command:

aws dynamodb create-table \

--table-name GridBuilder \

--attribute-definitions \

AttributeName=uniqueGridId,AttributeType=S \

AttributeName=s3Key,AttributeType=S \

--key-schema \

AttributeName=uniqueGridId,KeyType=HASH \

AttributeName=s3Key,KeyType=RANGE \

--provisioned-throughput \

ReadCapacityUnits=5,WriteCapacityUnits=5

**Expected output:**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\* This is OUTPUT ONLY. \*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

{

"TableDescription": {

"AttributeDefinitions": [

{

"AttributeName": "s3Key",

"AttributeType": "S"

},

{

"AttributeName": "uniqueGridId",

"AttributeType": "S"

}

],

"TableName": "GridBuilder",

"KeySchema": [

{

"AttributeName": "uniqueGridId",

"KeyType": "HASH"

},

{

"AttributeName": "s3Key",

"KeyType": "RANGE"

}

],

"TableStatus": "CREATING",

"CreationDateTime": "2023-03-09T19:42:47.981000+00:00",

"ProvisionedThroughput": {

"NumberOfDecreasesToday": 0,

"ReadCapacityUnits": 5,

"WriteCapacityUnits": 5

},

"TableSizeBytes": 0,

"ItemCount": 0,

"TableArn": "arn:aws:dynamodb:us-west-2:111111111111:table/GridBuilder",

"TableId": "eaf90c7b-adf3-4988-afa2-e32b30709cfb"

}

}

 Congratulations! You have successfully created the *GridBuilder* DynamoDB table.

**Task 2: Update the application to use DynamoDB and create a Lambda function**

In this task, you update the application to save the mapping from *uniqueGridId* to an S3 object by using the *dynamodb.put\_item* method. You also create a Lambda function based on your application.

CHALLENGE A: UPDATE THE APPLICATION TO USE DYNAMODB

In this task, you update the application so it uses DynamoDB to store the images that it needs to create the grid image. This process is a more manual way to do this task, but you learn how to automate it later. After you update the application, you create a deployment package of the application and its dependencies. Lambda can then consume the deployment package to create a Lambda function.

1. Open the file tree and then open the **/api-backend-manual/add\_image/app.py** file.
2. Update the **dynamodb.put\_item** method to save the mapping from **uniqueGridId** to an **S3 object**.

**Hint**

**Solution**

1. To create a compressed .zip deployment package of the application and its dependencies, run the following command:

cd ~/environment/api-backend-manual/add\_image ; zip ~/environment/api-backend-manual/add\_image app.py

**Expected output:**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\* This is OUTPUT ONLY. \*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

updating: app.py (deflated 50%)

TASK 2.2: CREATE A LAMBDA FUNCTION

In this task, you create the *add\_image* Lambda function based on the deployment package that you just created.

**Note:** The AWS CLI command to create the *add\_image* Lambda function references two variables. One variable is for the *Lambda role* that’s used when the *add\_image* Lambda function is called. The other variable stores the value for the *source bucket name* that was created as part of the resources for this lab.

1. To create the **add\_image** Lambda function by using the deployment package, run the following command:

aws lambda create-function \

--function-name add\_image \

--runtime python3.9 \

--timeout 30 \

--handler app.lambda\_handler \

--role $LAMBDA\_ROLE \

--environment Variables={SOURCE\_BUCKET=$SOURCE\_BUCKET} \

--zip-file fileb://~/environment/api-backend-manual/add\_image.zip

**Expected output:**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\* This is OUTPUT ONLY. \*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

{

"FunctionName": "add\_image",

"FunctionArn": "arn:aws:lambda:us-west-2:111111111111:function:add\_image",

"Runtime": "python3.9",

"Role": "arn:aws:iam::111111111111:role/LambdaApplicationRole",

"Handler": "app.lambda\_handler",

"CodeSize": 723,

"Description": "",

"Timeout": 30,

"MemorySize": 128,

"LastModified": "2023-03-09T20:18:50.090+0000",

"CodeSha256": "E4gBd56Wks8faPlDXZkMFE8YJFanA8SbXkGr4S+IKR8=",

"Version": "$LATEST",

"Environment": {

"Variables": {

"SOURCE\_BUCKET": "source-images-us-west-2-6376353"

}

},

"TracingConfig": {

"Mode": "PassThrough"

},

"RevisionId": "5a82dfde-5c6f-420d-8927-97a59680e7e6",

"State": "Pending",

"StateReason": "The function is being created.",

"StateReasonCode": "Creating",

"PackageType": "Zip",

"Architectures": [

"x86\_64"

],

"EphemeralStorage": {

"Size": 512

},

"SnapStart": {

"ApplyOn": "None",

"OptimizationStatus": "Off"

},

"RuntimeVersionConfig": {

"RuntimeVersionArn": "arn:aws:lambda:us-west-2::runtime:07a48df201798d627f2b950f03bb227aab4a655a1d019c3296406f95937e2525"

}

}

 Congratulations! You successfully updated the application, created a deployment package, and used it to create the *add\_image* Lambda function.

**Task 3: Create an API by using API Gateway**

In this task, you create the API.

1. At the top of the AWS Management Console, in the search bar, search for and choose

API Gateway

, then choose **Open Link in New Tab**.

1. On the **Choose an API type** page, in the **HTTP API** section, choose **Build** .
2. On the **Create an API** page, in the **Create and configure integrations** section:

* Choose **Add integration**
* For **Integrations**, choose **Lambda**.
* For **AWS Region**, choose the **region value** that’s specified in the lab details at the left of this screen.
* For **Lambda function**, choose the function name that ends with **function:add\_image**.
* Choose **Add integration**.
* For **Integrations**, choose **Lambda**.
* For **AWS Region**, choose the **region value** that’s specified in the lab details to the left of this screen.
* For **Lambda function**, choose the function name that ends with **function:generate\_grid**.
* For **API name**, enter

generate-grid

.

1. Choose **Next** .
2. On the **Configure routes** page, in the **Configure routes** section:

* For both entries that are listed, for the HTTP **Method**, choose **POST**.

1. Choose **Next** .
2. On the **Define stages** page, keep the default values and choose **Next** .
3. On the **Review and create** page, choose **Create** .

**Expected output:**

*Successfully created API generate-grid…*

1. On the left navigation pane, choose **Stages**.
2. Choose the **$default** stage.
3. Copy the invoke URL to an editor. It should look similar to *https://vf3acap6h0.execute-api.us-west-2.amazonaws.com*.

You use the invoke URL in a later step.

 Congratulations! You have successfully created an API by using API Gateway. This API can receive *POST* HTTP methods from the *add\_image* and *generate\_grid* Lambda functions.

**Task 4: Create the grid image by using API Gateway**

In this task, you engage the API to create the grid image by using images that are stored in the *source-images* S3 bucket. After the API processes all images, a final grid image is created and stored in the *destination-images* S3 bucket.

1. Return to the **AWS Cloud9** environment browser tab.
2. Create the **uniqueGridId** variable value (which is based on the **timestamp** when the command is run) by running the following command:

uniqueGridId=`date +%s` ; echo ${uniqueGridId}

**Expected output:** Your value will differ from what is shwon below.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\* This is OUTPUT ONLY. \*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

1683045233

You now create the *baseURL* variable by using the *invoke URL* value from API Gateway that you copied and saved previously.

1. To create the **baseURL** variable, update the placeholder text with the *invoke URL* value and then run the following command:

baseUrl='placeholder-for-invoke-url'

**Expected output:**

*None, unless an error occurs.*

Now, you invoke the API using the following curl -X POST method to send data to it. The command sends an image file named *image01.jpg* as binary data to the *source-images* S3 bucket by using the HTTP POST method. Next, it passes the parameter named *uniqueGridId* with the *baseUrl* value, and stores this data in the DynamoDB table.

1. Invoke the API Gateway by using the following commands:

cd ~/environment/api-backend-manual/source

curl -X POST --data-binary @image01.jpg "${baseUrl}/add\_image?uniqueGridId=${uniqueGridId}"

**Expected output:**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\* This is OUTPUT ONLY. \*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

{"message": "image saved", "image\_size": 196799}

1. Repeat this step three more times for images named **image02.jpg**, **image03.jpg**, and **image04.jpg**.

**Hint**

**Note:** If you encounter an error with these commands, check your code in the */api-backend-manual/add\_image/app.py* file against the *app\_solution.py* file.

* If you need to make a fix, update the *app.py*. Repackage the application, and use

aws lambda update-function-code

 to deliver the updated package.

* **Additional Information:** Learn more about how to use [update-function-code](https://us-west-2-tcprod.s3.us-west-2.amazonaws.com/courses/SPL-BE-200-DVMTAG/v1.0.3.prod-d32f9881/instructions/en_us/#additional-resources).

1. View the entries that were created in DynamoDB by using the

aws dynamodb scan --table-name GridBuilder

 command below:

aws dynamodb scan --table-name GridBuilder

**Expected output:**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\* This is OUTPUT ONLY. \*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

{

"Items": [

{

"uniqueGridId": {

"S": "1679419021"

},

"s3Key": {

"S": "4541c45e0cbe761f38539aa79d8a07a2.jpg"

}

},

{

"uniqueGridId": {

"S": "1679419021"

},

"s3Key": {

"S": "92f5e8ac5f183332a5a1eabf04594e3f.jpg"

}

},

{

"uniqueGridId": {

"S": "1679419021"

},

"s3Key": {

"S": "9a75b10395415fea979ebf9e05e968b2.jpg"

}

},

{

"uniqueGridId": {

"S": "1679419021"

},

"s3Key": {

"S": "be0f0c7fb685709a3bf3888929972905.jpg"

}

}

],

"Count": 4,

"ScannedCount": 4,

"ConsumedCapacity": null

}

1. To engage the API to create the **grid image** and **S3 presigned URL** while formatting the output using jQuery, run the following command:

curl -s -X POST "${baseUrl}/generate\_grid?uniqueGridId=${uniqueGridId}" | jq -r '"\nMessage: " + .message, "\nPresigned\_URL: " + .presigned\_url, "\n"'

**Expected output:**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\* This is OUTPUT ONLY. \*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Message: built grid

Presigned\_URL: https://destination-images-us-west-2-4806916.s3.amazonaws.com/443905deebdb93ce140b1f3bf4cd6269.jpg?AWSAccessKeyId=ASIAVZP34UDT5FIS3JBO&Signature=s3Rl27UbSKOI%2B9UqFCz4DPZ3k1s%3D&x-amz-security-token=IQoJb3JpZ2luX2VjECYaCXVzLXdlc3QtMiJHMEUCIElV35jEIbLAjEVTzg3U35lpl94YMbWGm3n760H1NyJCAiEAuVC69TKrwxdHGWdFMTvanq36H%2BCkTsTwcHdxaPo9Icwq7gIITxAEGgwzOTgzNDk2MDcxNDMiDFVZVxqL%2B8Uf4j9P7SrLAmqcxrnMuVIv8%2FqtbZyBuI3paB4grnJFAImerFvxOK%2BshGZ4QFVGNKg8QqQl7IxYOy8oc35l%2Fr65bIkvrfMRM8AVKZ77rVR6hbDu69fyJm9F%2BEawJZ0afB54GSd7%2BkXvKsgPyq02Oqc1N%2BUbOER9qa56AdQTZzu8OoewOzs6xIOyzRKfLImIhsgDiU5AT1rmO0IrBRVCfqNgj9HF6DxZZgzlaQ5gNoQTyBoApmBjCaIfqH3o57YZPz%2BRBKlbSYoNrRIJUpSc7iEzdlJA8%2BaAjJbiw%2BEWPpddoW2hPsIxK0VrU9JRixHhy9y4xx%2FkU2BJWvHDVhElOdQt7kBolOaqlB6dBwfxiew7LRIuGPnYK1iMyHkFd%2Fe8BgPN%2FX9Mvwr8%2BSfC9%2BnAu9KT%2BPxczbe9vvMGNTxuf9bApfmMthlznD%2BIkGam%2BnR7nya5dbkwmZ2VowY6ngGlFMjv0tfvFLG6hpE4%2FlHuLC4CwThj%2FezDcUCzHrfZfV1uq9MNH7MyHUZEYdsMqGF24oSnIQAInabBT04tK3UmBw%2BqAWVu3ZwASiFeqnkqi8DEo0E2n2VkZ7fvz6KpXJnbVfD14pjKf%2BE53hVt%2BFw6IRvFBWYxSE9xTgFAlnYCUHamfwJDNiM3A24rA9Zbmy6HQLFwY7kw7z%2BY7J26oA%3D%3D&Expires=1684361161

Now, you can see the *grid-image.jpg* that was created from the images by using Lambda as the compute environment

1. Open the **S3 presigned URL** in a new browser tab.



**Note:** To add additional photos to the grid image, you can continue to run the command to add additional images, and then run the command to generate the grid image.

 Congratulations! You successfully engaged the API to create the grid image based on images that were stored in the GridBuilder DynamoDB table. Additionally, the API generated an S3 presigned URL, where the image is stored in the *destination-images* S3 bucket. Then, you viewed the *grid-image.jpg* file in a browser tab by using the *S3 presigned URL*.

**Conclusion**

 Congratulations! You now have successfully:

* Created a DynamoDB table.
* Updated the application to save the mapping from uniqueGridId to an S3 object using dynamodb.put\_item.
* Deployed the application.
* Created an API by using API Gateway.
* Run the API to create the grid image and an S3 presigned URL.

**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.

**Additional Resources**

* Learn more information about [Lambda update-function-code](https://docs.aws.amazon.com/cli/latest/reference/lambda/update-function-code.html).

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).