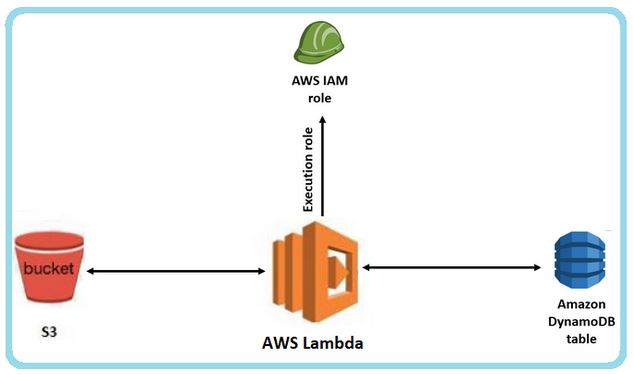
# Use Lambda Functions within a Stateless Architecture on AWS

**Create a Lambda Function with a Role**

Your company is using Amazon S3 for file storage (using an S3 bucket) where all users can put and delete their files. When a user uploads or deletes a file, your company requires an entry to be immediately created in the database table. This entry contains the file metadata, like the file’s name, creation time, size of the file, Bucket, Event, and so on.

Here is the diagram which illustrates the lab configuration:﻿

**Note:** You are expected to know or be able to figure out the details of how to use the S3 Console UI/X (as per [**Lab Details**](https://app.pluralsight.com/labs/detail/92bc7a07-400e-4cbf-98ab-79dc22f568d4) > **Prerequisites**). This includes but is not limited to uploading files to and deleting files from an existing S3 bucket.

1. Log in to the AWS Console using the **user name**, **Password**, and **Open AWS console** button provided by this lab.

**Note:** Make sure during this entire lab in the upper-right of the AWS Console your region is **Oregon** (US West(Oregon ) us-west-2)**.**

1. At the top of the page in the search bar, type in and click on **Lambda**.
2. Click **Create function**.
3. At the **Create function** page, enter the following:
   * Function name: my-function
   * Runtime: **Python 3.8**

Click **Create function**.

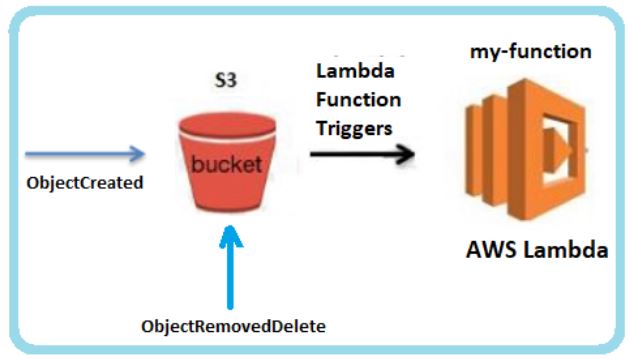
**Note:** An AWS Lambda function named my-function will be created with a basic execution role.

1. At the top of the page in the search bar, type in and click on **IAM**.
2. In the navigation pane on the left, click on **Roles**.
3. Click **Create role**.
4. Choose **Lambda** **Allow Lambda functions to call AWS Services on your behalf.** and click on **Next**.
5. Under **Attach permissions policies**, search for and check the following two policies:
   * **AmazonDynamoDBFullAccess**
   * **AmazonS3ReadOnlyAccess**
6. Click **Next**.
7. For the **Role name**, enterlambda-s3-dynamodb-role.
8. Click **Create role**.
9. At the top of the page in the search bar, type in and click on **Lambda**. Click the **my-function** link, then click on **Configuration > Permissions**.
10. Beside **Execution role**, click **Edit**, and enter the following:
    * Description: lambda-s3-dynamodb-role
    * Existing role: **lambda-s3-dynamodb-role**

Click **Save**.

You will be back at the **my-function** page, at the **Permissions** tab, and it will show a new **Role name** of **lambda-s3-dynamodb-role**.

# Add an S3 Trigger to the Lambda Function

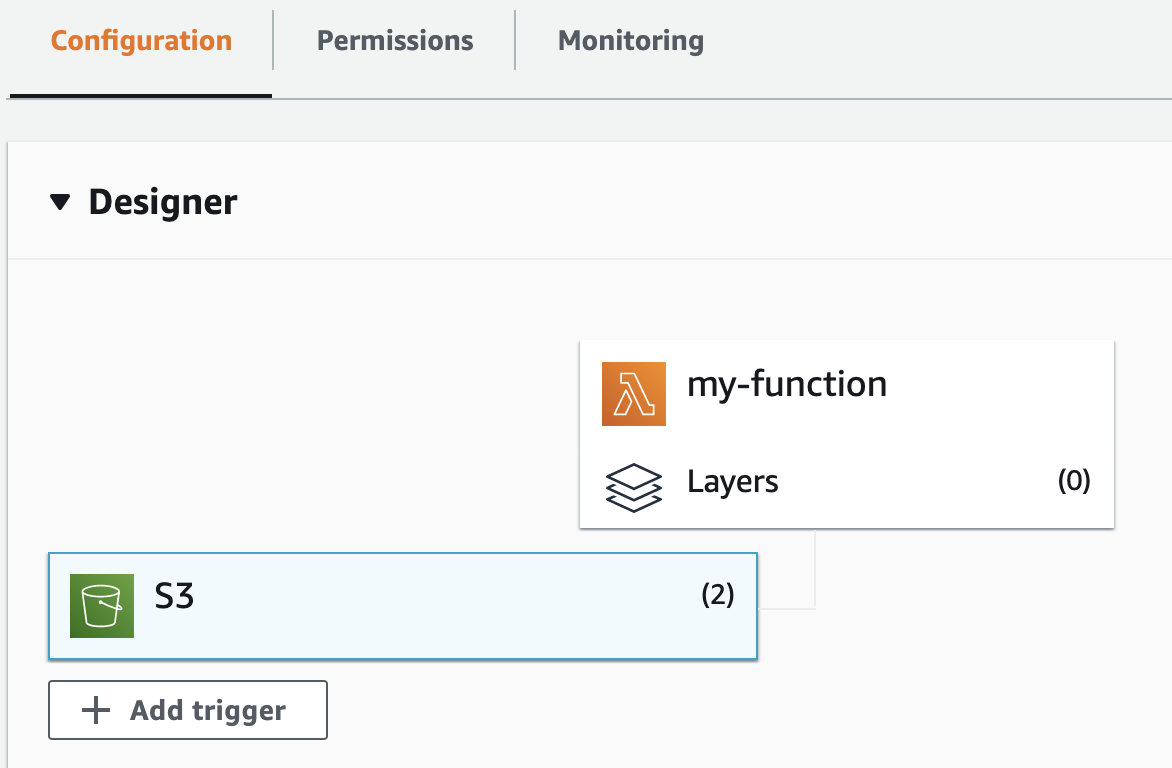
﻿﻿

1. Click **Configuration** **> Triggers**.
2. Click **Add trigger**.
3. From the **Trigger configuration**, type in and click **S3**. Then fill in the following values:
   * Bucket: **lambda-s3-plural-<random\_name>**
   * Recursive invocation: Check the **I acknowledge** box.

Click **Add**.

1. Click **+ Add trigger**, choose **S3** and the **lambda-s3-plural-<random\_name>** as before, but this time select an **Event type** of **Permanently deleted**. Check the box, then click **Add**.

You will see you have added two S3 triggers to your function.

﻿﻿

# Add Code to the Lambda Function

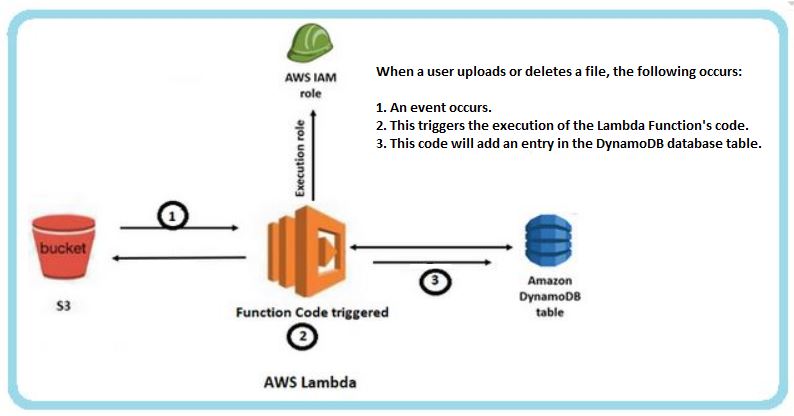
1. Click on the **Code** tab, scroll down to the **Code source** section, and double-click the **lambda\_function.py** file to open it.
2. Copy and paste the code below into the **lambda\_function** editor, overwriting ]any code that already exists:

import boto3 from uuid import uuid4 def lambda\_handler(event, context): s3 = boto3.client("s3") dynamodb = boto3.resource('dynamodb') for record in event['Records']: bucket\_name = record['s3']['bucket']['name'] object\_key = record['s3']['object']['key'] size = record['s3']['object'].get('size', -1) event\_name = record ['eventName'] event\_time = record['eventTime'] dynamoTable = dynamodb.Table('lambda-s3-table') dynamoTable.put\_item( Item={'RequestId': str(uuid4()), 'Bucket': bucket\_name, 'Object': object\_key,'Size': size, 'Event': event\_name, 'EventTime': event\_time})

1. Click **Deploy**.

The **Deploy** button will be greyed out and you will see a **Changes deployed** message.

# Trigger Lambda Function Code

﻿﻿

The code in your Lambda function, upon uploading to and deleting items from the S3, will write rows to a DynamoDB table.

You will trigger than code and see the results in the table.

1. At the top of the page in the search bar, type in and click on **S3**.
2. Click on the **lambda-s3-plural-<random\_name>** link.

**Note:** This bucket was created for you when you started this lab.

1. Click **Upload**, and add any small file from your local system. Sample\_Cloud\_Lab.docx is used in the examples here. Then at the bottom of the page click **Upload**.
2. Head back to the **lambda-s3-plural-<random\_name>** S3 bucket, and upload a new file. links.txt will be used as an example here. Then, delete the one (Sample\_Cloud\_Lab.docx) you uploaded before.
3. At the top of the page in the search bar, type in and click on **DynamoDB**.
4. In the left-hand menu, click **Tables**.
5. Click the **lambda-s3-table** link, then click **Explore table items**.

﻿

You will see three entries, caused by uploading the two files (probably shown as **PUT** events), and the one deletion.

**Note:** If you don't see three items, click that tab's refresh button.

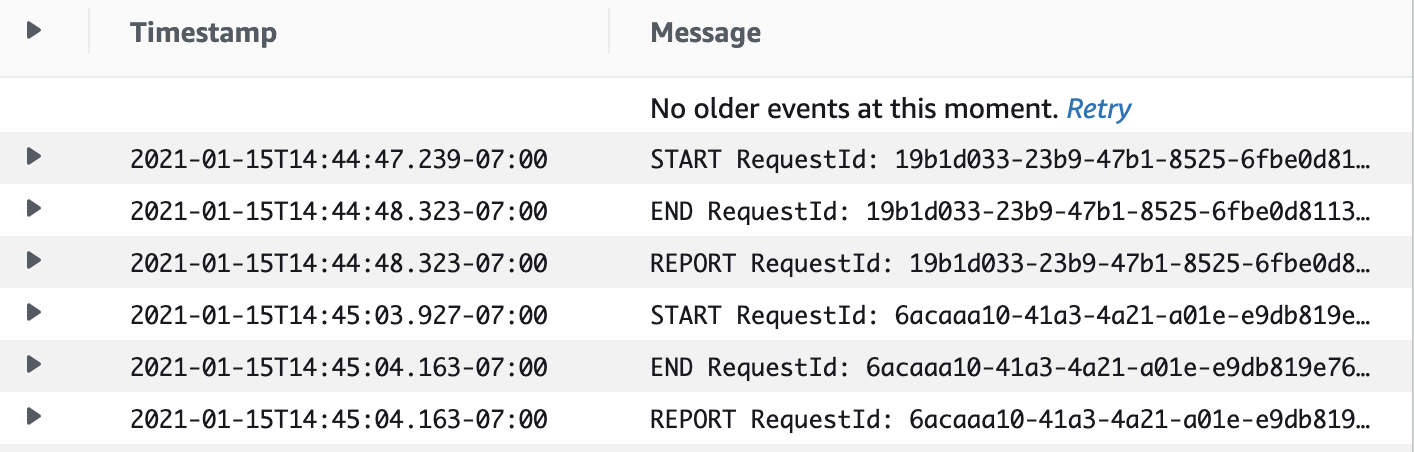
**Use AWS Lambda to Trigger Email Notification on S3 Events**

1. At the top of the page in the search bar, type in and click on **IAM**.
2. In the left-hand menu click on **Roles**, then click **Create role**.
3. Choose **Lambda Allow Lambda functions to call AWS Services on your behalf.**, and click on **Next**.
4. In the **Attach permissions** **policies** search bar, type in and select the two policies below:
   * **AmazonSESFullAccess**
   * **AWSLambdaExecute**
5. Click **Next**, then enter a **Role name** of lambda-ses-role.
6. Click **Create role**.
7. Go back to the **Lambda Functions** page, click **my-function**,and then click on **Configurations >** **Permissions**.
8. Beside **Execution role**, click **Edit**, and enter the following:
   * Description: lambda-s3-ses-role
   * Existing role: **lambda-ses-role**

Click **Save**.

1. Click the **Code** tab, and scroll down to the **Code source** section. (The **lambda\_function** should already be opened in the code editor.)
2. Copy and paste the result into the code editor, overwriting the existing code. import json import boto3 def lambda\_handler(event, context): for i in event["Records"]: action = i["eventName"] ip = i["requestParameters"]["sourceIPAddress"] bucket\_name = i["s3"]["bucket"]["name"] object =i["s3"]["object"]["key"] client = boto3.client("ses") subject = str(action) + 'Event from ' + bucket\_name body = """ <br> This is a notification mail to inform you regarding {} event. The object {} is deleted. Source IP: {} """.format(action, object, ip) message = {"Subject": {"Data": subject}, "Body": {"Html": {"Data": body}}} response = client.send\_email(Source = "<your email>", Destination = {"ToAddresses": ["<your email>"]}, Message = message)
3. In the code editor, in the last line replace both occurrences of <your email> with your email address. Click **Deploy**.
4. At the top of the page in the search bar, type in and click **Simple Email Service**.
5. In the navigation pane on the left, click **Verified identities**.
6. Click **Create Identity**, then, on the following page, select **Email address** and enter your email address.
7. Click **Create identity**.
8. Check your email, and in the **Email Address Verification Request** message follow the instructions.
9. Go to the **S3** page, and click on the **lambda-s3-plural-<random\_name>** link.
   * Upload a new file pics.jpg, and delete the existing one links.txt**.** (Just examples, use files from your own system, links.txt is the file you uploaded previously.)
10. Check your mail, you will get two emails, one for an **ObjectCreated** event, the other for an **ObjectRemoved** event.
11. At the top of the page in the search bar, type in and click **CloudWatch**.
12. In the left-hand menu. click **Log groups**, then click the **/aws/lambda/my-function** link.
13. At the bottom of the page in the **Log streams** section, click the **Log stream** link of the most recent item. (There will likely only be one item.)

You will see entries similar to the following:



# Deploy an AWS Lambda Using a Container Image

Lab 2

# Use the Cloud9 Environment

The company you work for is looking for a quick way to deploy web services through containers running on a serverless environment. One of the ways AWS allows us to do that is by deploying container images to AWS Lambda.

In this lab, you’ll walk through the steps necessary to deploy a simple web service, written in Go and packaged as a container image, to AWS Lambda.

In this first challenge, you will get started with Cloud9, the web-based environment you will use to build and publish the container image.

1. Use the **user name**, **password**, and **Open AWS console** to the right of this content to log into the AWS Console.
2. Once in the AWS Console, navigate to the AWS **Cloud9** service.
3. On the AWS **Cloud9** dashboard, there should be one environment.

Note: The name of the environment should start with PS-Labs- and be followed by 5 random characters. For example: PS-Labs-LFWHN or PS-Labs-GVUOV, etc.

1. Click the **Open IDE** button (either one). A new browser tab will open with a browser-based IDE. Wait a few moments so that the project files are loaded.
2. Once loaded, click the code folder. You’ll be editing files inside this folder shortly. But before you do that, you will run a few Docker commands.
3. There are a few terminal tabs towards the bottom of the screen. Use the right-most one. That’s where you’ll be running the Docker CLI commands.
4. Inside the terminal tab, type the following command to list the number of images available:

docker images

Note it's fine if nothing is displayed; the following is done just to ensure no previous images exist.

1. Run the following command to delete all current container images:

docker rmi $(docker images -qa)

1. To make sure all images were deleted, run the docker images command again. This time, no images should be listed.

If the second time you ran the docker images command no images were listed, congratulations! You’ve finished setting up Cloud9 and are ready to move on to the next challenge.

Commands

aws ecr get-login-password --region us-west-2 | docker login --username AWS --password-stdin 973789387811.dkr.ecr.us-west-2.amazonaws.com

# Create an ECR (Elastic Container Registry) Repository

In order to deploy a container image to Lambda, the image must be stored in Amazon ERC. In this challenge, you will create a private ECR repository.

1. Go to the AWS Console and navigate to the **Elastic Container Registry** service.
2. Click **Get Started**. This should take you to a **Create Repository** page**.**
3. Make sure the **Visibility Settings** is **Private**.
4. Type ps-labs-app for repository name.
5. Scroll down to the bottom of the form and click **Create repository**. This should create the new repository and then take you to the repository list page.
6. Find the new repository on the list (it should be the only one!) and then click on it to go to its details page.
7. On the top right hand-side, click the **View push commands** button. A modal should open. Make sure the **macOS / Linux** tab is selected and copy the first command. The command should start with:

aws ecr get-login-password...

Leave the modal open, as you’ll come back to it later to copy the other commands.

Note: ERC uses a separate authentication mechanism which will require you to run a few commands before pushing new images to this repository.

1. Go back to **Cloud9**. In the terminal tab, run the command you copied from the previous step, aws ecr get-login-password...

If you can see the **Login Succeeded** message, congratulations! You’ve successfully created a new ERC repository and are ready to move on to the next challenge.

# Push an Image to ECR

With the ECR repository in place and authentication out of the way, now it’s time to build the container image and push it to the repository you just created.

1. Still on the **Cloud9** editor and in the command line, run the following command to change into the code directory:

cd ~/environment/code

1. Inside the **code** folder, run the following command to build the container image:

docker build -t ps-labs-app .

**Note:** Don’t forget the dot (.) at the end of the previous command.

This command should take a minute or two to finish, and the tail end of the output should display the following message:

**Successfully tagged ps-labs-app:latest**

1. The image is built. Now, you need to create a remote tag for this image on the Amazon ERC repository. Go back to **Amazon ECR**. On the modal with the list of commands, find and copy command number 3. This command should start with docker tag ...
2. Go back to **Cloud9** and run the command you copied from the previous step in a terminal tab. Running this command should not display any results.
3. Go back to **Amazon ECR** one more time, and copy the last command on the modal. It should start with docker push ...
4. Back to Cloud9, run the push command from a terminal tab. It should take a few seconds to run, and the output should finish with something like:

**latest: digest:** sha256:8699be62436423... size: 1368

1. Go back to the **Amazon ECR** page one last time. Close the modal and refresh the page for the **ps-labs-app** repo.

You should now see the image you just pushed from Cloud9 listed on the page, and with an image tag set to latest.

Congratulations! You’ve successfully pushed a container image to a private ERC Repository. Move on to the next and final challenge.

# Deploy a Container to AWS Lambda

Your application’s container image is properly stored in the private ECR repository you created. Now, you will create a Lambda function that will pull the image from this repository, and run it as a process.

1. Go to the AWS Console and navigate to the **Lambda** service.
2. On the Lambda page, click **Create function.**
3. Select the option **Container Image.**
4. For **Function Name**, enter hello-function
5. For **Container Image URI**, click **Browse Images**. A modal should pop up.
6. On the drop-down, select the ECR repository you created: **ps-labs-app.**
7. Once selected, the image you just pushed with the tag **latest** should be displayed. Select this image by clicking the radio button to its left.
8. Click the button **Select image.** This should close the modal.
9. Back to the Lambda function form page, click **Create function**.

Creating the function should take a minute or so. Once finished, you should see the success alert at the top with the message saying **Successfully created the function hello-function**.

1. Click the **Test** tab.
2. Enter an **Event name** of test-hello.
3. For **Event JSON**, overwrite the existing json with {}.
4. Click **Save**.
5. Click **Test**.
6. Under **Execution result**, expand **Details**.

You will see the message **Hello from PS Labs**. Congratulations! You’ve successfully deployed a Lambda function using a container image and reached the end of this lab.

Lab 3 Schedule AWS Lambda Functions Trigger Using Amazon EventBridge

# Create and Deploy a Lambda Function

You are a Senior developer at Globoticket, a Globomantics Company. You have been asked to update the DynamoDB table every minute with the latest event and ticket details. You have decided to use EventBridge rules to trigger a Lambda function at a regular interval in order to update the DynamoDB table.

As a first step, you will create and deploy a Lambda function.

1. Log int o the AWS console using the provided credentials.
2. In the AWS console, type Lambda in the search box at the top and hit enter.
3. Click **Create function**.
4. In the Create function page:
   * Choose **Author from scratch**.
   * Function name: update-function
   * Runtime: **Python 3.9**
   * Under Permissions, expand **Change default execution role** and choose **Create a new role from AWS policy templates**.
   * Role name: lambda-ddb-role
   * Policy templates: **Simple microservice permissions**

Click **Create function**.

You will be redirected to the update-function page.

1. In the **Code source**'s **lambda\_function.py**, replace the existing code with the code below:

import uuid

import random

import boto3

dynamodb = boto3.resource('dynamodb')

table = dynamodb.Table('event\_details')

location = ['SFO', 'London', 'HongKong', 'NewYork', 'Seattle']

def lambda\_handler(event, context):

event\_id = str(uuid.uuid4())

event\_location = random.choice(location)

number\_of\_tickets = random.randint(50,1000)

cost = random.randint(10,1000)

response = table.put\_item(

Item={

'event\_id': event\_id,

'event\_location': event\_location,

'number\_of\_tickets': number\_of\_tickets,

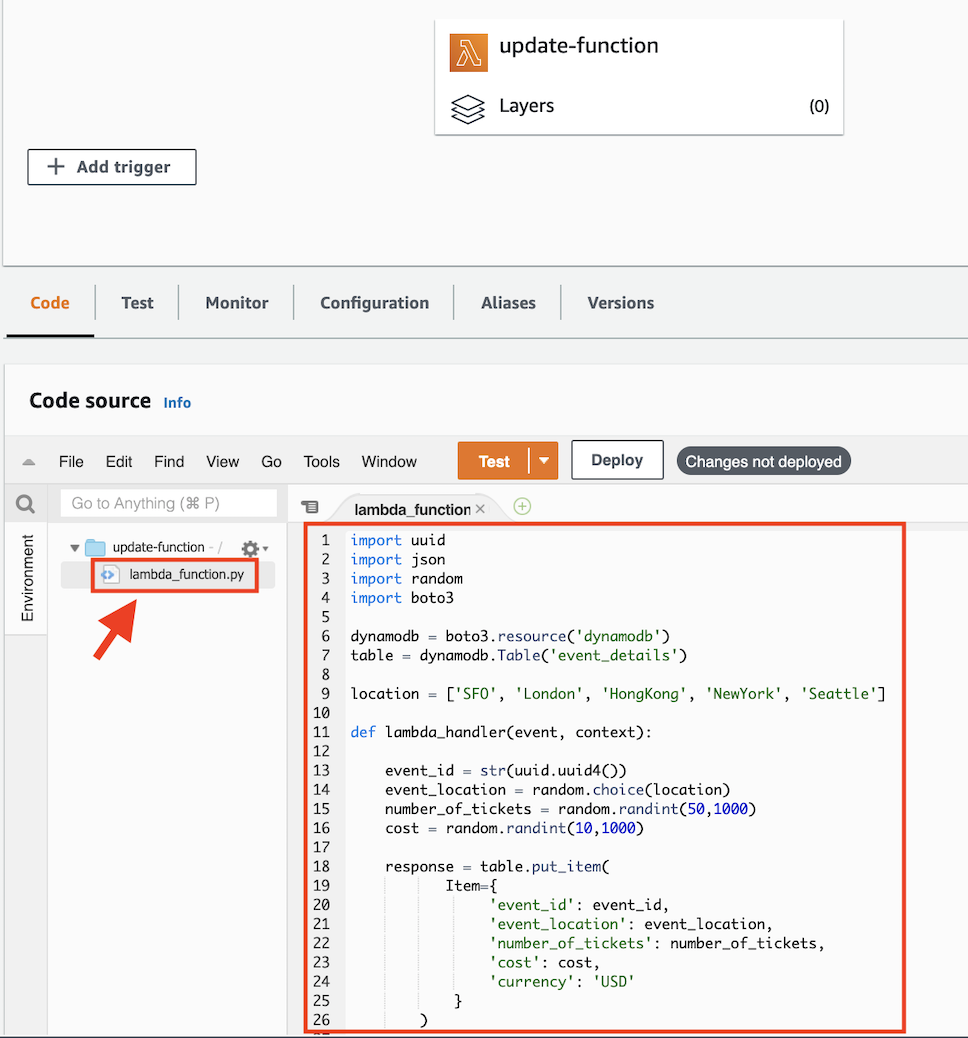
'cost': cost,

'currency': 'USD'

}

)

return response

﻿The Lambda function code creates a new event id, event location, number of tickets available for that event. All these details are then updated in the DynamoDB table event\_details using the put\_item API on line 18.

Note: The event\_details table was been pre-created for you when you started the lab.

1. Click **Deploy**.

A **Changes deployed** message will appear above the code.

Congratulations! You have successfully created and deployed a Lambda function that updates the DynamoDB table with the latest event and ticket details. In the next challenge, you will create an EventBridge rule to trigger the Lambda function on an automated schedule.

# Create an EventBridge Rule and Setup Lambda Function as Target

Now that you have a Lambda function deployed, you have been asked to create an EventBridge Rule to trigger the Lambda function in a scheduled manner to update the DynamoDB table with the last event and ticket details.

1. In the AWS console, type EventBridge in the search box at the top and hit enter.
2. In the left navigation pane, click **Rules** under **Events** section.

**Note:** Ignore the AccessDeniedExecption errors for events:DescribeEventBus

1. Click **Create rule**.
2. In the Create rule page, enter a **Name** of scheduled-lambda-trigger
3. Choose **Schedule** as the **Rule type,** then click **Next.**
4. On the **Define schedule** page, select the radio button for the regular rate schedule, then enter a value of 1 and change the drop-down to **Minutes.** Click **Next**.
5. In the **Target types section** choose **AWS service**.
6. In the **Target types** section, choose **Lambda function** from the drop-down.
7. Choose **update-function** from the **Function** drop-down.
8. Click **Next** on this and the following screen, then click **Create rule**.

This will trigger the Lambda function every minute and update the DynamoDB table event\_details with the latest event and ticket details.

**Note:** Ignore the AccessDeniedExecption errors for events:DescribeEventBus

1. In the AWS console, type DynamoDB in the search box at the top and hit enter.
2. Click **Tables** in the left navigation pane.
3. Click **event\_details** in the center.
4. Click **Explore table items**.

Wait for a few minutes or refresh the browser page, and you will see one or more records show up.

Congratulations! You have successfully created an EventBridge rule to trigger Lambda function every minute and update the DynamoDB table with the latest event and ticket details.

Lab 6:-

Deploy a Lambda Using AWS CodeDeploy and the Serverless Application Model