BUILDING AWS LAMBDA FUNCTION ON MACHINE LEARNING MODEL

Example of invoking Lambda Function on S3 PUT Event with Existing XGBoost Endpoint

Yiran Jing

July 05, 2019

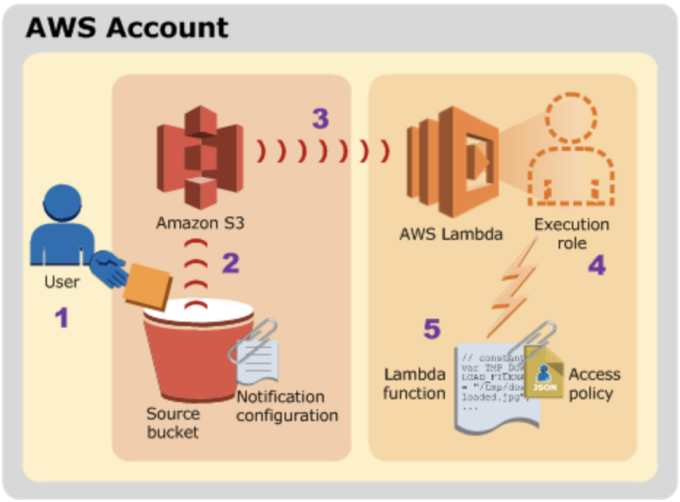
Contents

1 Introduction to Lambda Function

Basically **AWS Lambda** lets you focus on writing code and not dealing with annoying things like VPCs, EC2 instances, MySQL databases, etc. Just write some Python, give that code to Lambda, and it will execute that code in the Cloud. Even better, **you can trigger that code in a variety of ways**: every minute, once a day, when you put something into an S3 bucket, etc. In this case, I give an example of execute Lambda Function on S3 event trigger, that is, we can *execute lambda function automatically on our built ML models when we push new dataset to S3 bucket*. That is, after you write up your Lmabda Function, everyone can easily use it to run model on new dataset by S3 put trigger, and the user does not need to touch SamgeMaker or Lambda function again. See figure 5 below:

Example 1: Amazon S3 Pushes Events and Invokes a Lambda Function

Amazon S3 can publish events of different types, such as PUT, POST, COPY, and DELETE object events on a bucket. Using the bucket notification feature, you can configure an event source mapping that directs Amazon S3 to invoke a Lambda function when a specific type of event occurs, as shown in the following illustration.



The diagram illustrates the following sequence:

1. The user creates an object in a bucket.
2. Amazon S3 detects the object created event.
3. Amazon S3 invokes your Lambda function using the permissions provided by the execution role.
4. AWS Lambda executes the Lambda function, specifying the event as a parameter.

Figure 1: Example: Amazon S3 Pushes Events and Invokes a Lambda Function

2 Clean data within Lambda Function

2.1 Install AWS Lambda with Pandas and NumPy

AWS Lambda does not include Pandas/NumPy Python libraries by default. But we do need use Pandas and NumPy with Lambda functions for data cleaning and transformation. *click me to see how to do it*

2.2 Write python script to manipulate raw data

2.3 Combine python script to Lambda Function

3 Create IAM role that grants access to S3 bucket

Before you get started building your Lambda function, you must first have an IAM role which Lambda will use to work with S3 and to write logs to CloudWatch. You can use existing Role called ***Lambda_Permission_endpoint*** for any Lambda function with CloudWatch and S3 event trigger permission. The following is the details about how to create this role in AWS console.

This role should be set up with the appropriate S3 and CloudWatch policies. See figure 2, select Lambda and click *Next: Permission*.

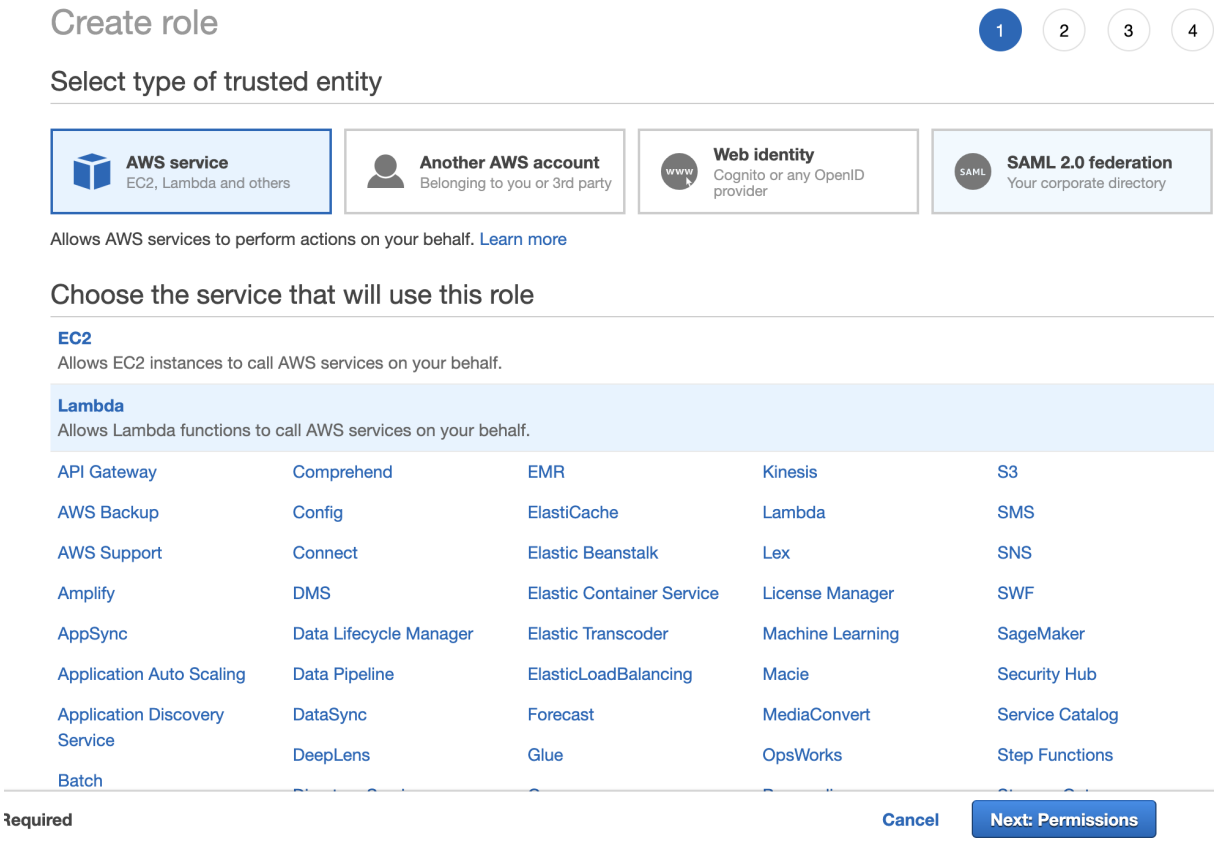


Figure 2: Create role steps

And then you need to select three pollicies ***AWSLambdaFullAccess, AmazonS3FullAccess and AmazonSageMakerFullAccess***, also you need give **CloudWatchPermission** by **adding inline policy with Json Format** after you create the role: See figure 3 and figure 4. *Click me to look the details of* ***Lambda_Permission_endpoint***.

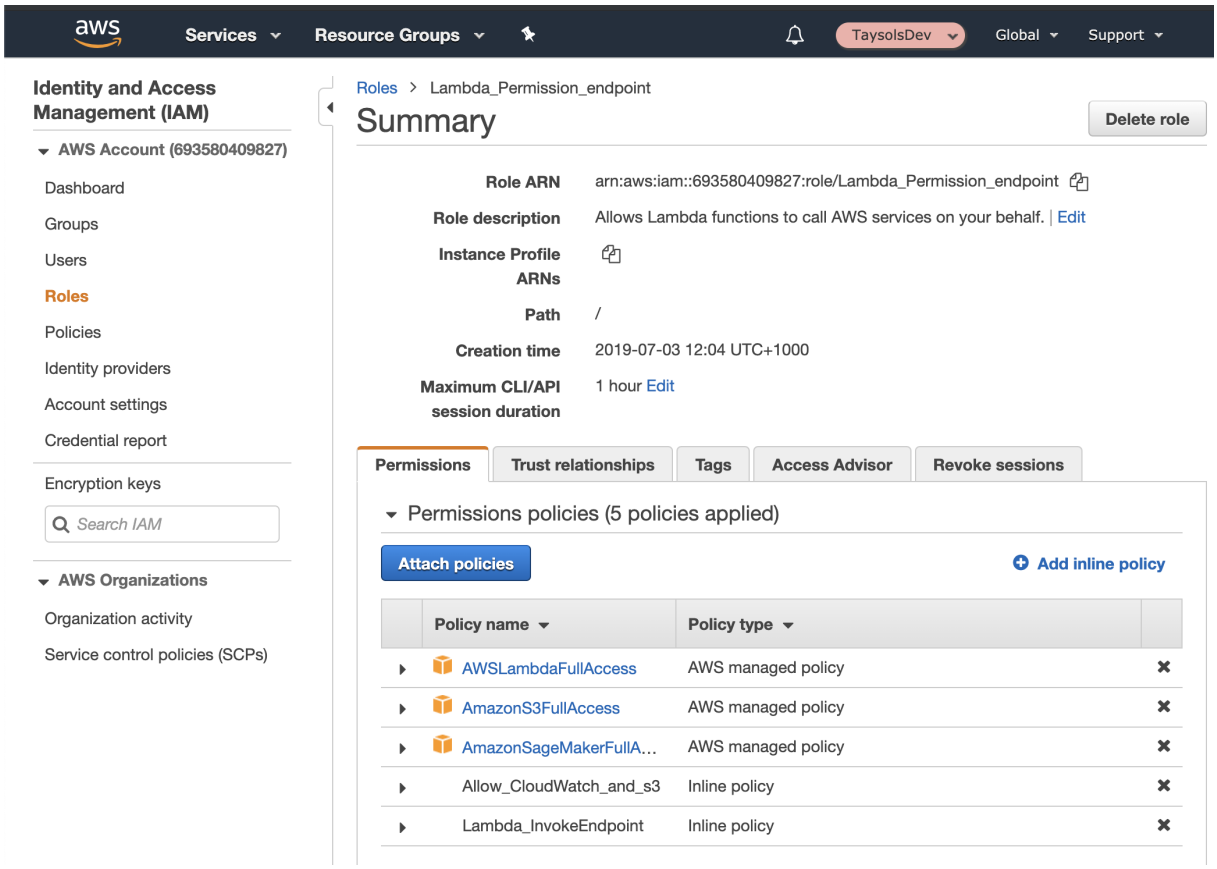


Figure 3: IAM role's policy: AWSLambdaFullAccess, AmazonS3FullAccess and AmazonSageMakerFullAccess

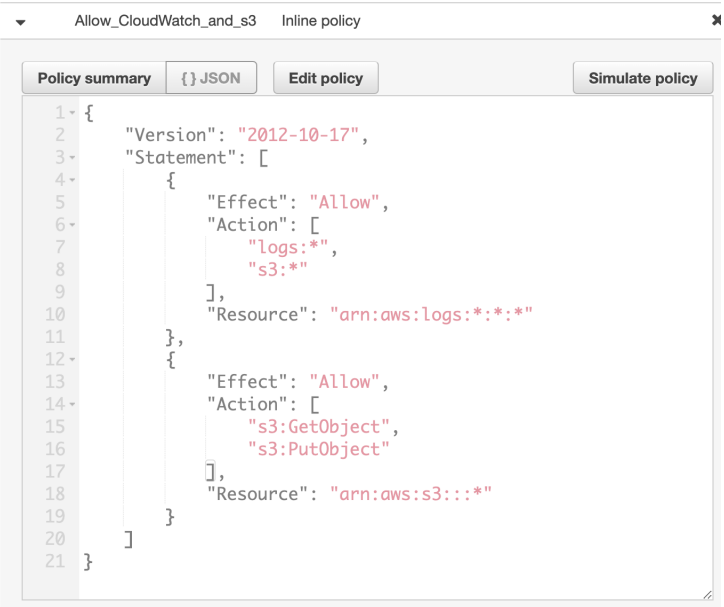


Figure 4: IAM role's policy: CloudWatchPermission

4 Create an empty Lambda function

After we have a SageMaker model endpoint, for further usage of modelling we need to do is to Create a Lambda function that calls the SageMaker Runtime Invoke_Endpoint See figure 5, click *create function*

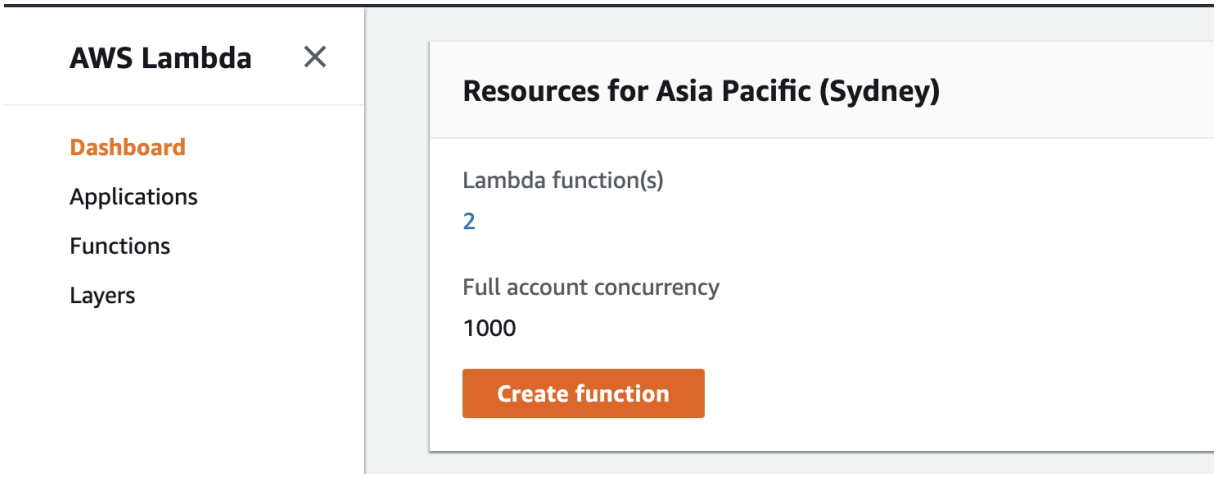


Figure 5: Create a new Lambda Function from Amazon User Interface

After that, give the name and language for your lambda funcation, see figure 6. Please select **Python 3.6** and **Use an existing role**, then select the role which you created before. In this example, the IAM role I created in the last step is *Lambda_Permission_endpoint*.

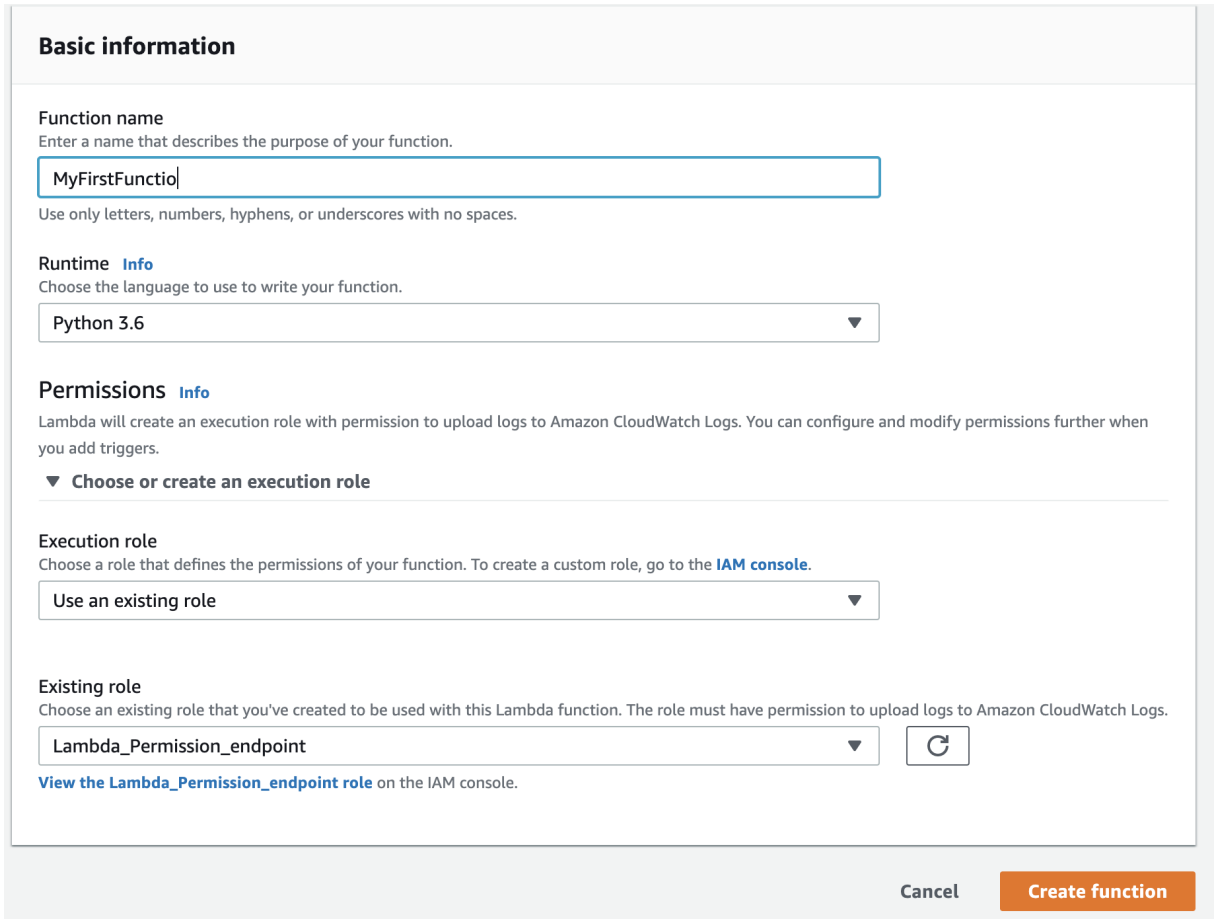


Figure 6: Give the name and choice the language for your lambda function

Then, you can check the lambda function you have create a new lambda function through AWS Lambda interface, see figure 7.

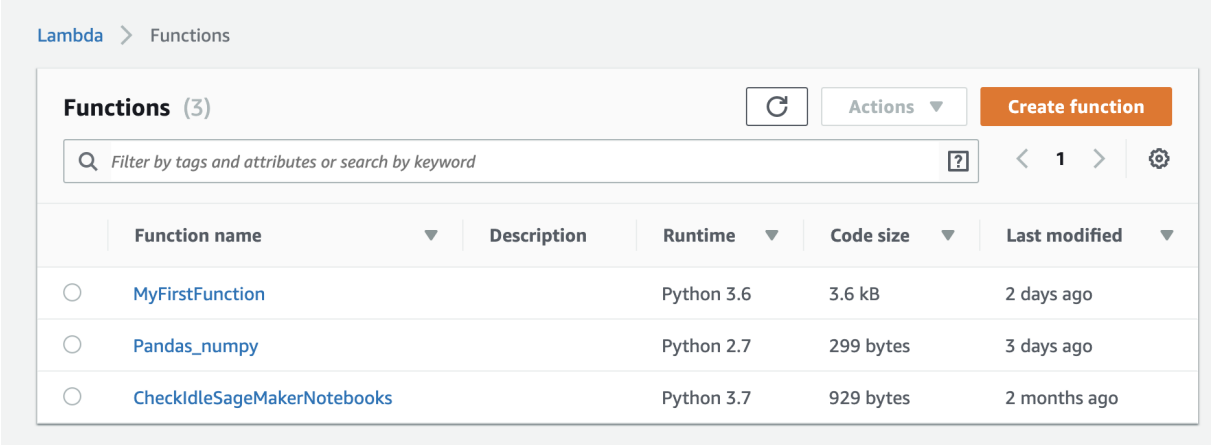


Figure 7: Check you have create a new lambda function

5 Build your Lambda Function

For the following steps, Please remember that every time you **must click *save*** before click *test* to running the new code on your *test event*.

5.1 S3 Event Triggers

After you create a new empty Lambda function, the next step is add **S3 put** as event trigger. Click **add triggers** See figure 8. select S3 **PUT** as trigger event, then Enter prefix, in case if you have any folders inside the S3 and want to triggered only uploading to that folder. In our example, the **Prefix is the path of the file containing input dataset**. Suffix is **.csv** since our dataset is csv.

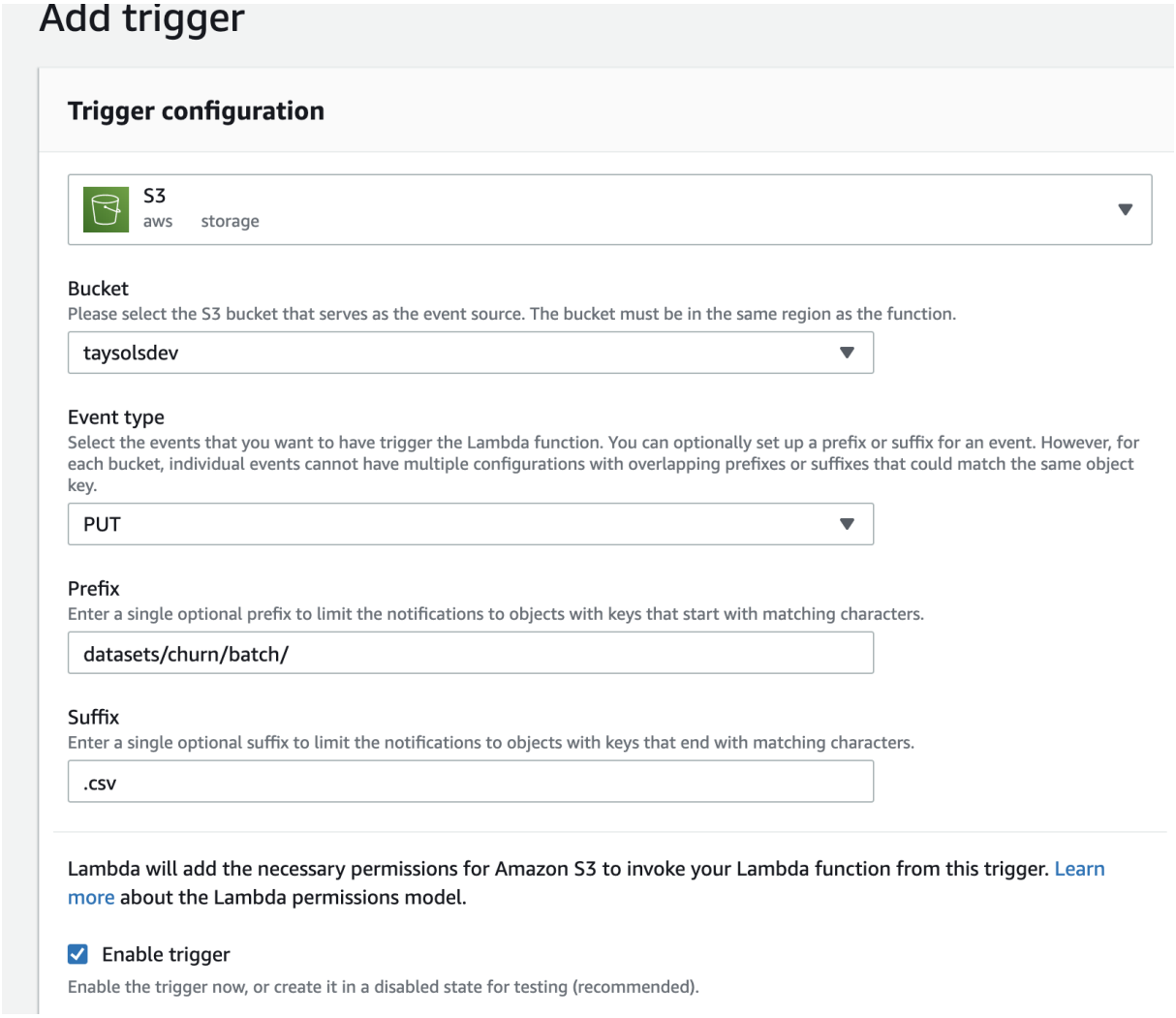


Figure 8: Select S3 put as event trigger

After that, check your lambda function, it should look like figure 9.:

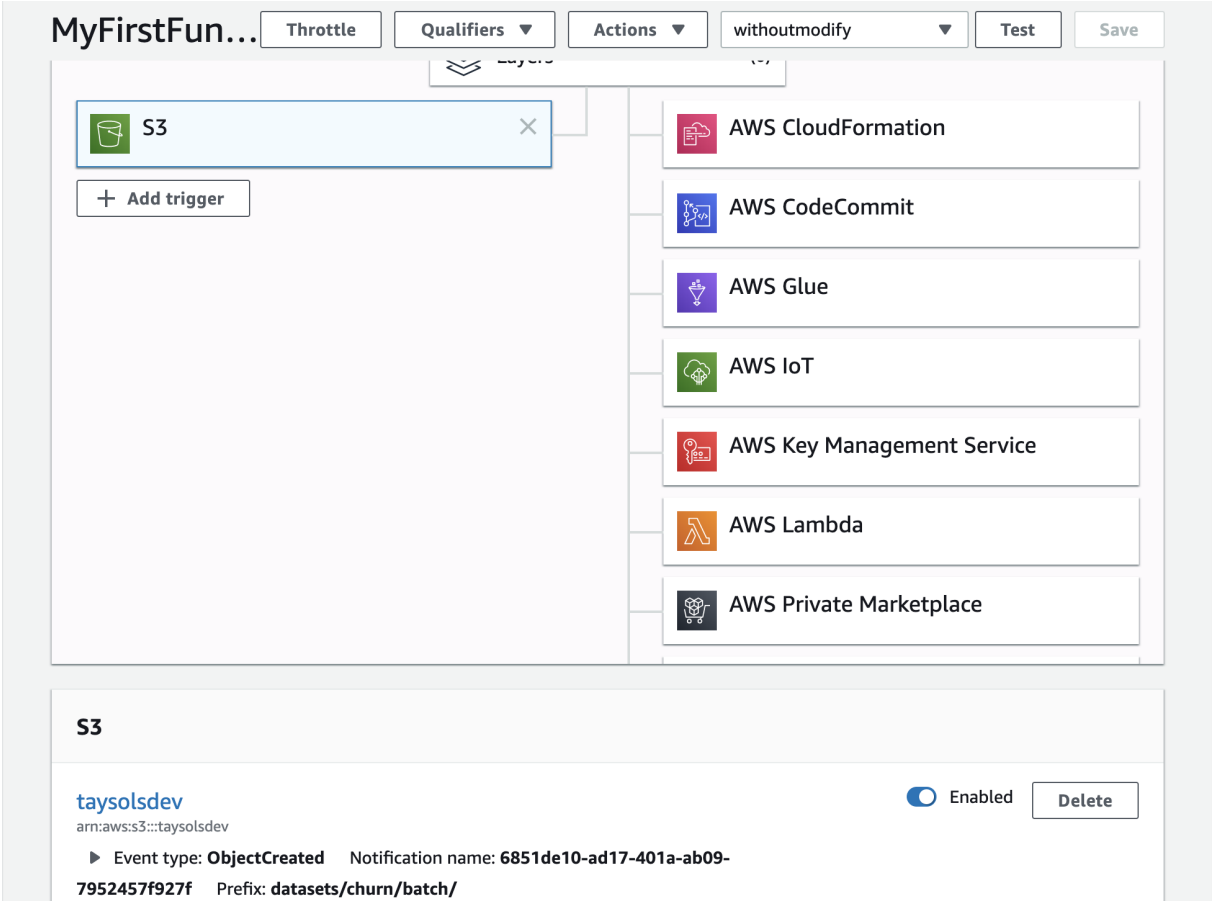


Figure 9: your lambda function

5.2 Input environment variable

ENDPOINT_NAME is an environment variable that holds the name of the SageMaker model endpoint you just deployed using the sample notebook as shown in the following screenshot figure 10:

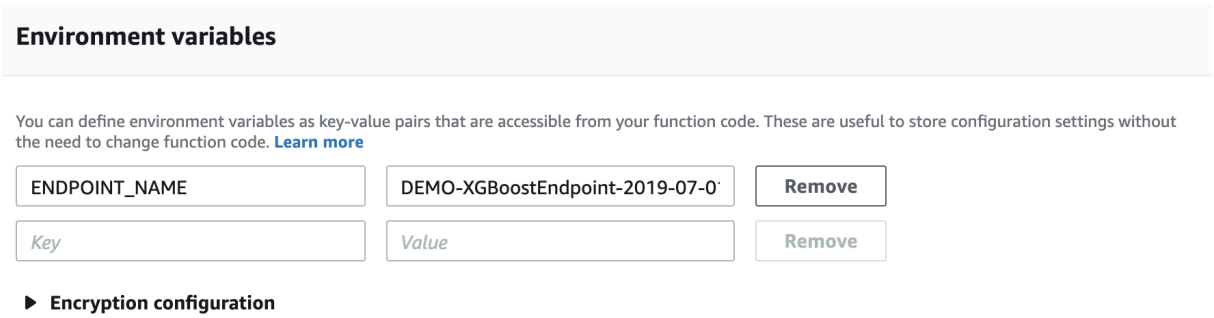


Figure 10: Add endpoint of model to your Lambda Function

You can also modify or delete S3 event trigger in S3 bucket. See figure 11 and figure 12:

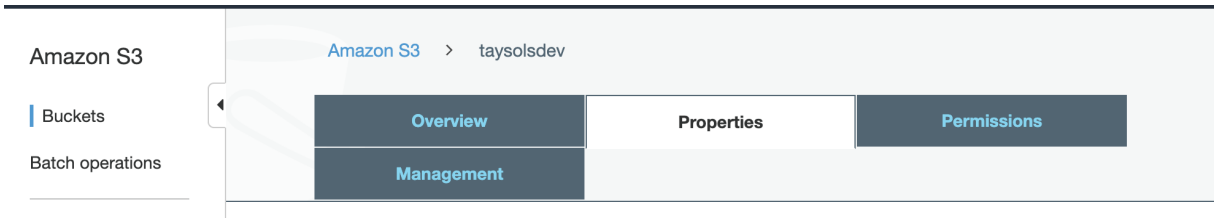


Figure 11

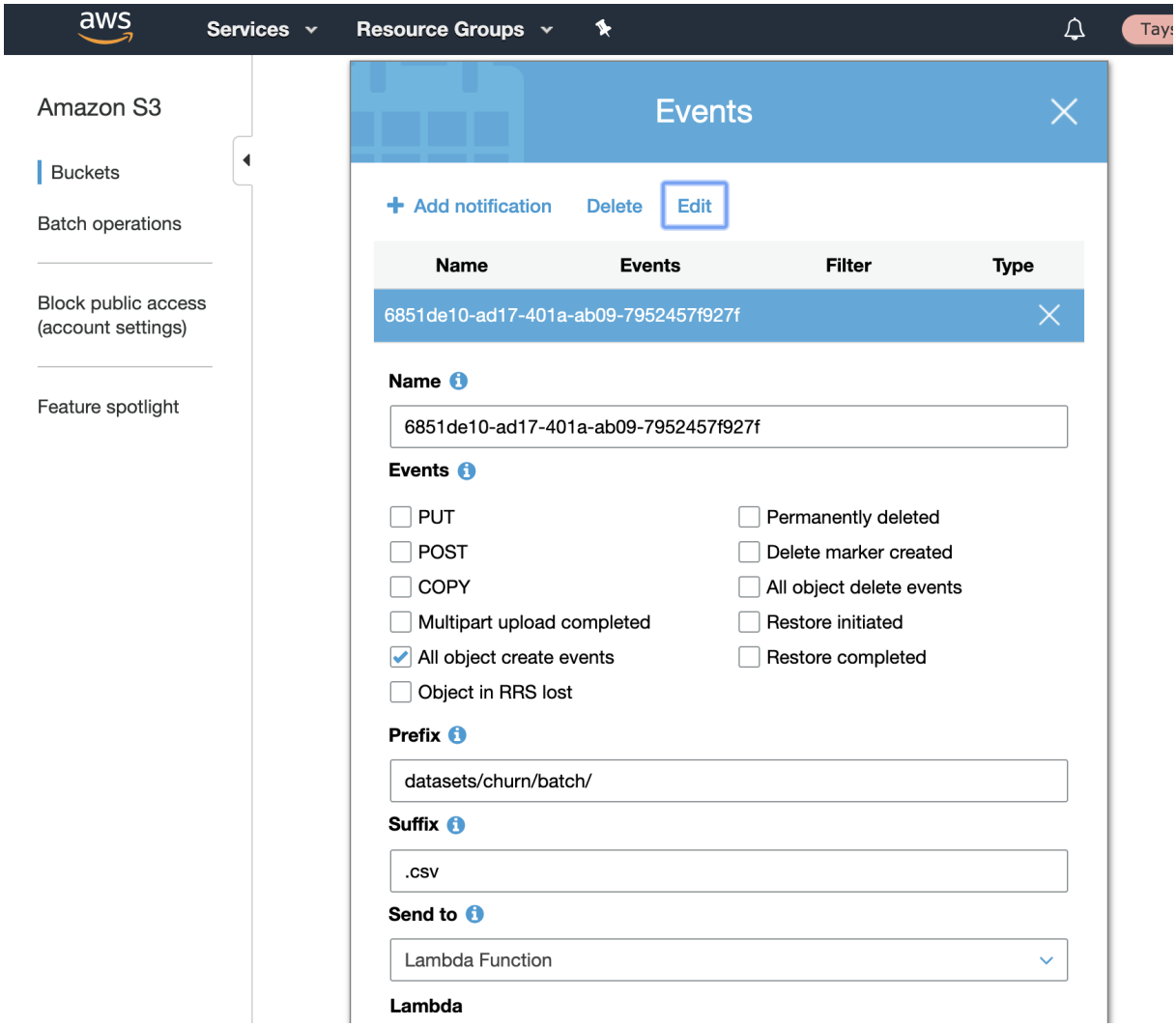


Figure 12

5.3 Configure test event

The event is *dict* type with *JSON* format. The **test event** is used for debug your Lambda function. See figure 13, Select **Create new test event** and then choose **Amazon S3 put**.

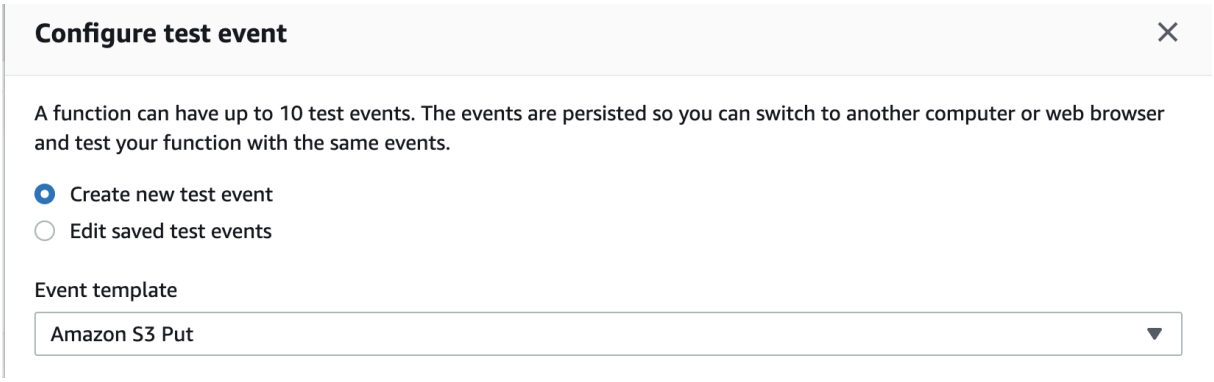


Figure 13: Add endpoint of model to your Lambda Function

you need to modify *bucket name* and the *key*(the path of your input dataset). For example, the test event used in our case see figure 14.

```
    "bucket": {
      "name": "taysolsdev",
      "ownerIdentity": {
        "principalId": "EXAMPLE"
      },
      "arn": "arn:aws:s3:::example-bucket"
    },
    "object": {
      "key": "datasets/churn/batch/test_data_Batch.csv",
      "size": 1024,
      "eTag": "0123456789abcdef0123456789abcdef",
      "sequencer": "0A1B2C3D4E5F678901"
    }
  }
```

Figure 14: Modify the content of test event template

5.4 Lambda Builders

Lambda Builders is a separate project that **contains scripts to build Lambda functions**,given a source location. Build Actions could be implemented in any programming language. Preferably in the language that they are building, I use Python as the DEMO example in this note. See figure 17.

5.4.1 Lambda Handler with its Help function

In this example, our main function is ***lambda_handler*** within ***lambda_function.py***, see figure 15

Figure 15: Lambda Handler information

To make the main function easy to be understood and modify, I create script ***help_function_lambda.py*** for all necessary help functions used in ***lambda_handler***

function. These scripts are located in the same folder ***MyFirstFunction*** *(The name of folder is same as the name of the lambda function we just created).*

5.4.2 Lambda Handler Function

At the time you create a Lambda function, you specify a handler, which is a function in your code, that AWS Lambda can invoke when the service executes your code.I show the example that how to creating a handler function in Python.
In the syntax, note the following:

1. **event** AWS Lambda uses this parameter to pass in event data to the handler. This parameter is usually of the Python ***dict*** type with ***JSON*** format.

2. **context** AWS Lambda uses this parameter to provide runtime information to your handler. This parameter is of the *Lambda Context* type.

3. **runtime.invoke_endpoint** After you deploy a model into production using Amazon SageMaker hosting services, your client applications use this API to get inferences from the model hosted at the specified endpoint. Parameter ***EndpointName***: The name of endpoint of your per-trained model. You can find the endpoint name of your model in Amazon SageMaker interface. Parameter ***Body*** (*bytes or seekable file-like object*): Provides input data, Amazon SageMaker passes all of the data in the body to the model. ***.get()*** returns a *StreamingBody*. This is a series of bytes, not a string, thus we do need *.decode('utf-8')*. Return Type of *invoke_endpoint*: See figure 16

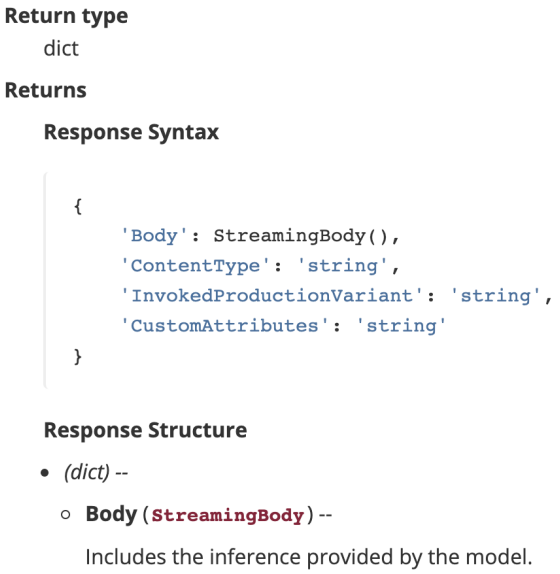


Figure 16: Invoke Endpoint Return Content

See figure 17

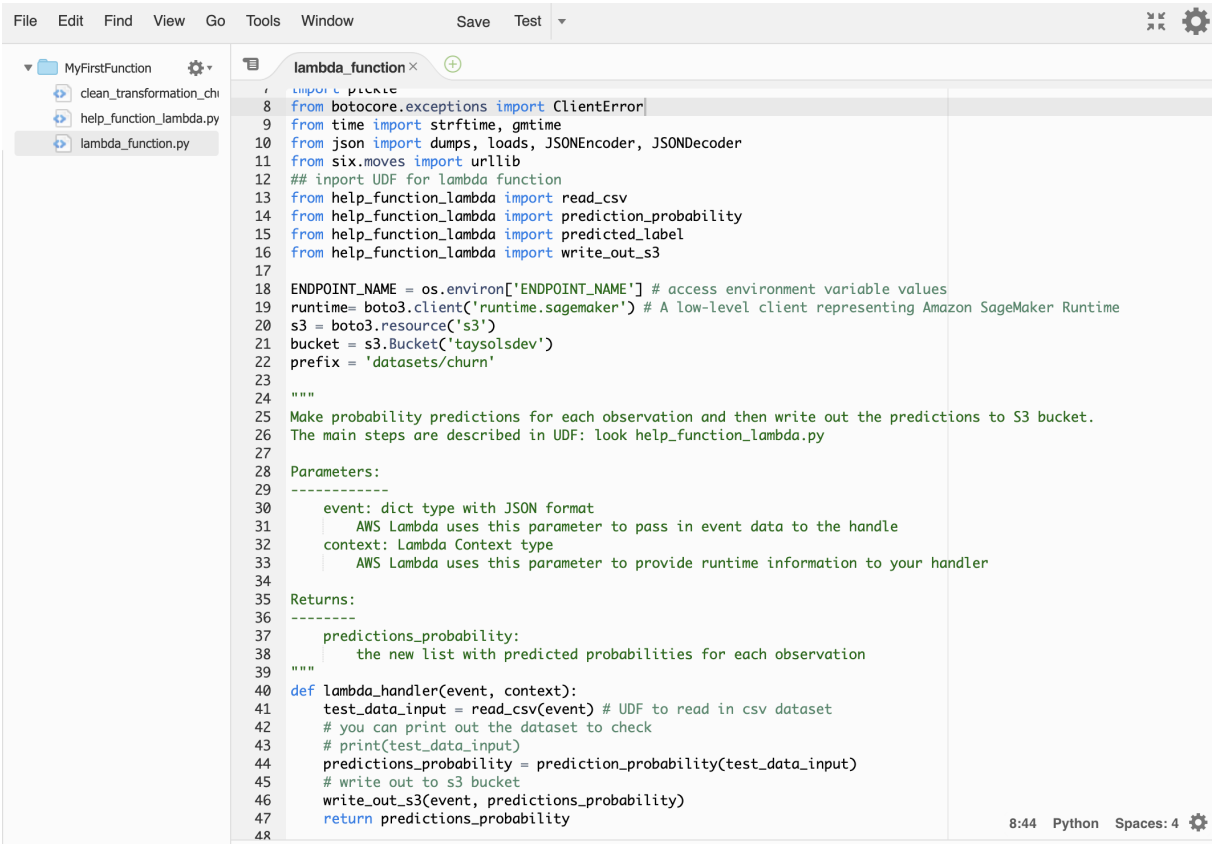


Figure 17: Lambda Handler function

5.4.3 Help functions for lambda handler

To make function easy to be understood and modify, I block functions as followings see figure 18, 19, 20, and 21:

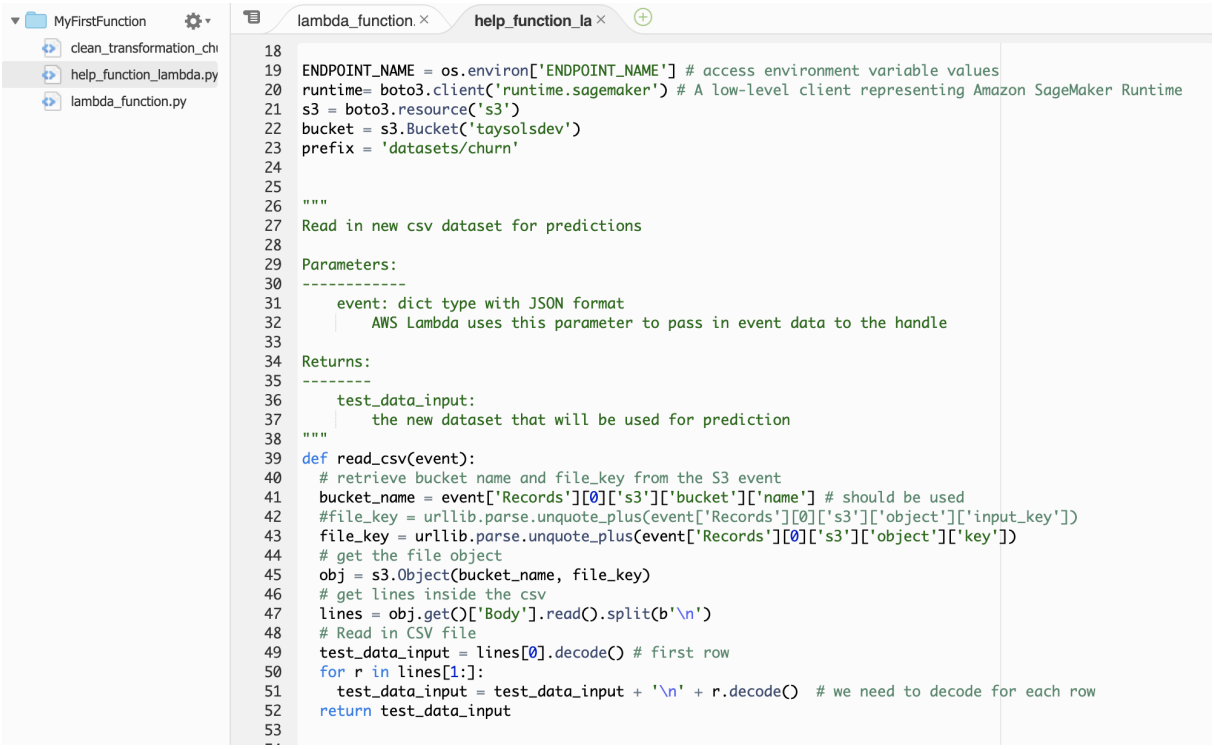


Figure 18: Read CSV

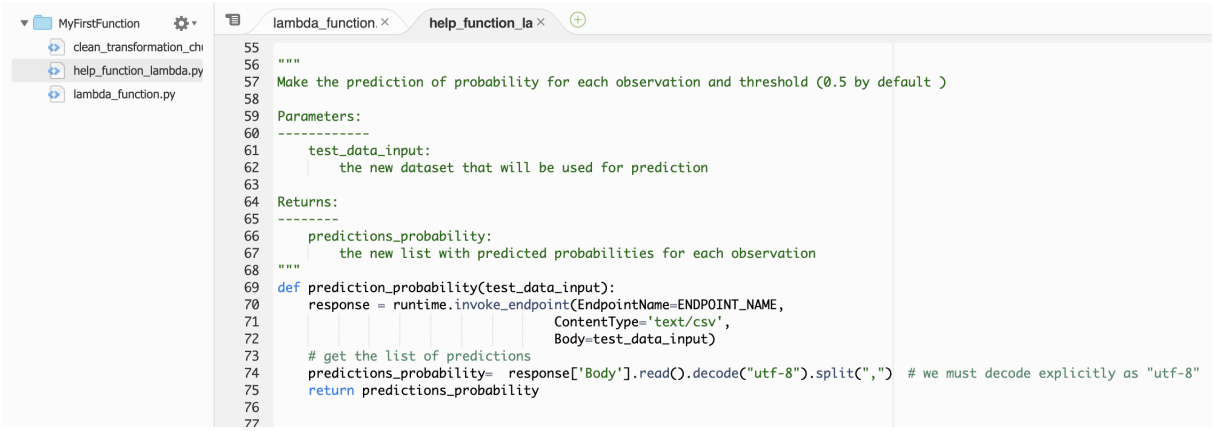


Figure 19: Probability prediction based on endpoint

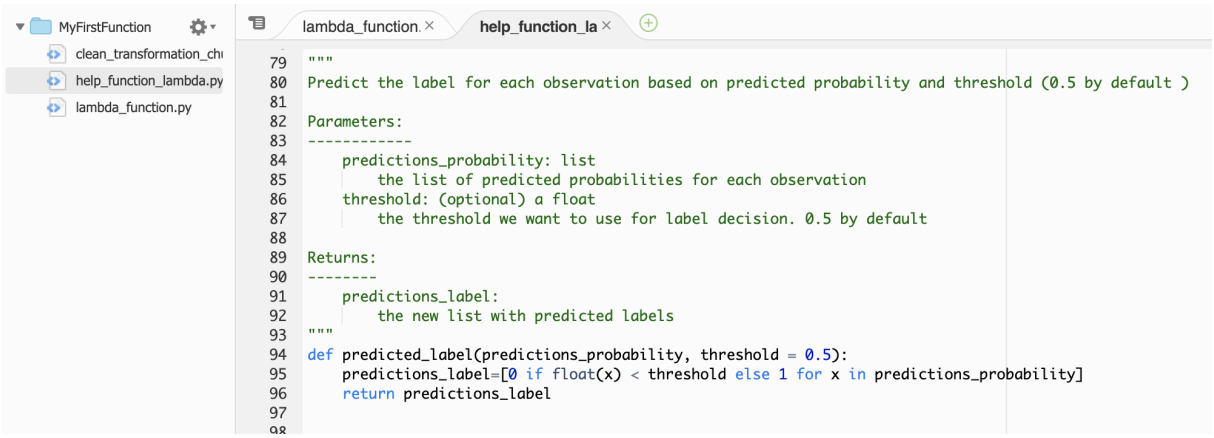


Figure 20: Label prediction based on threshold

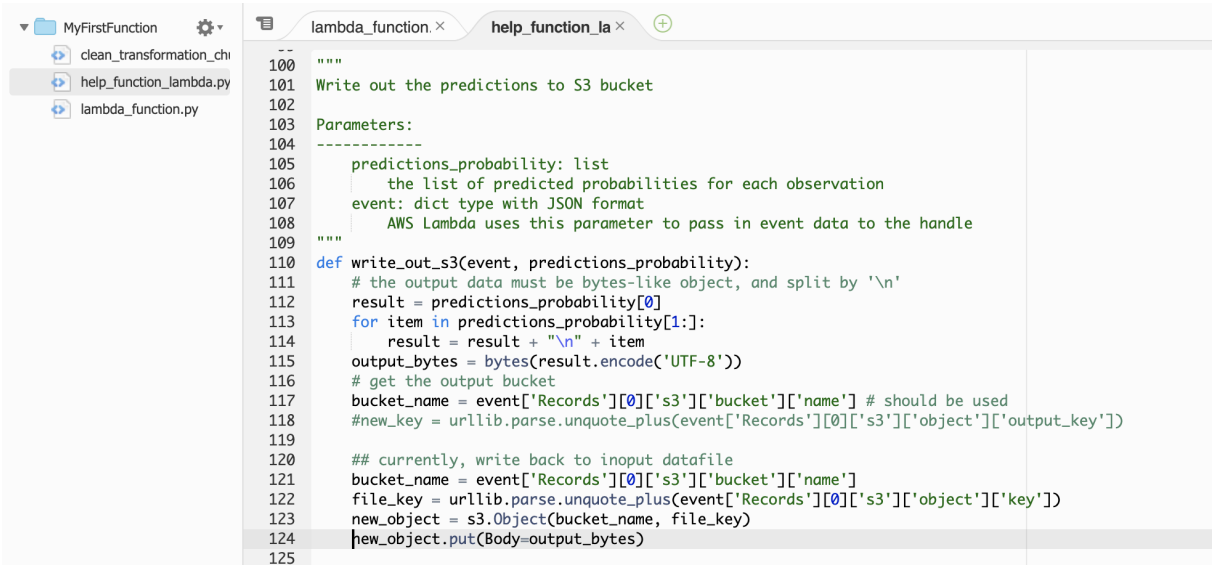


Figure 21: Write out output to S3 bucket

5.5 Common error and the way to fix

5.5.1 Configuration is ambiguously defined.

When you fail to add s3 trigger as *Lambda Error for event source : Configuration is ambiguously defined*, the reason could be that some other lambda function previously using the same trigger was deleted. This does not automatically clear the event notification from the S3 side. You have to ***navigate to the S3 console and manually delete the stale event notifications***. *Clink me to read the detail about this error*

5.5.2 TypeError: expected string or bytes-like object

It is the type error you might meet when try to save a Python list to an S3 bucket. In this case, we have to **convert list to bytes**. Thus, we need $bytes(json.dumps(predictions_probability, indent = 2).encode('UTF-8')))$

6 Test data: Check CloudWatch

By default, Lambda will write function activity to CloudWatch. This is why the role that was created earlier had to get access to CloudWatch. When a new file is uploaded to the S3 bucket that has the subscribed event, this should automatically kick off the Lambda function. To confirm this, head over to CloudWatch or click on the Monitoring tab inside of the function itself.
It is important to know how to look ***CloudWatch Logs Insights*** *to check if the event (for example, inpout data to S3 in our case) trigger the Lambda function successfully, and if fail, you can read the error information here to debug.*

7 Reference

1. *https://aws.amazon.com/cn/blogs/machine-learning/call-an-amazon-sagemaker-mo*
2. *https://n2ws.com/blog/aws-automation/lambda-function-s3-event-triggers*