

TAYSOLS DATA SCIENCE

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

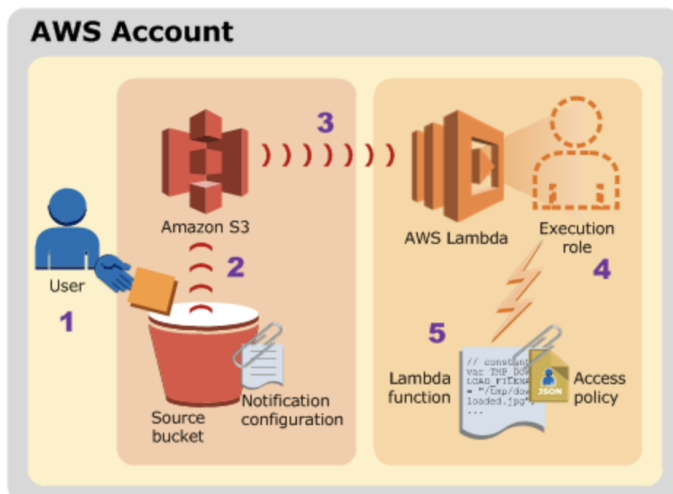
1	Introduction to Lambda Function	1
2	Clean data within Lambda Function	2
2.1	Install AWS Lambda with Pandas and NumPy	2
2.2	Write python script to manipulate raw data	2
2.3	Combine python script to Lambda Function	2
3	Create IAM role that grants access to S3 bucket	3
4	Create an empty Lambda function	5
5	Build your Lambda Function	7
5.1	S3 Event Triggers	7
5.2	Input environment variable	8
5.3	Configure test event	9
5.4	Lambda Builders	10
5.4.1	Lambda Handler with its Help function	10
5.4.2	Lambda Handler Function	11
5.4.3	Help functions for lambda handler	12
5.5	Common error and the way to fix	14
5.5.1	Configuration is ambiguously defined.	14
5.5.2	TypeError: expected string or bytes-like object	14
6	Test data: Check CloudWatch	15
7	Reference	16

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 Lambda Function, everyone can easily use it to run model on new dataset by S3 put trigger, and the user does not need to touch SageMaker 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*.

Create role

1 2 3 4

Select type of trusted entity

AWS service
EC2, Lambda and others

Another AWS account
Belonging to you or 3rd party

Web identity
Cognito or any OpenID provider

SAML 2.0 federation
Your corporate directory

Allows AWS services to perform actions on your behalf. [Learn more](#)

Choose the service that will use this role

EC2
Allows EC2 instances to call AWS services on your behalf.

Lambda
Allows Lambda functions to call AWS services on your behalf.

API Gateway	Comprehend	EMR	Kinesis	S3
AWS Backup	Config	ElastiCache	Lambda	SMS
AWS Support	Connect	Elastic Beanstalk	Lex	SNS
Amplify	DMS	Elastic Container Service	License Manager	SWF
AppSync	Data Lifecycle Manager	Elastic Transcoder	Machine Learning	SageMaker
Application Auto Scaling	Data Pipeline	ElasticLoadBalancing	Macie	Security Hub
Application Discovery Service	DataSync	Forecast	MediaConvert	Service Catalog
Batch	DeepLens	Glue	OpsWorks	Step Functions

required Cancel Next: Permissions

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***.*

Identity and Access Management (IAM)

- AWS Account (693580409827)
 - Dashboard
 - Groups
 - Users
 - Roles**
 - Policies
 - Identity providers
 - Account settings
 - Credential report
 - Encryption keys
- AWS Organizations
 - Organization activity
 - Service control policies (SCPs)

Roles > Lambda_Permission_endpoint

Summary Delete role

Role ARN arn:aws:iam::693580409827:role/Lambda_Permission_endpoint

Role description Allows Lambda functions to call AWS services on your behalf. [Edit](#)

Instance Profile ARNs [Copy](#)

Path /

Creation time 2019-07-03 12:04 UTC+1000

Maximum CLI/API session duration 1 hour [Edit](#)

Permissions Trust relationships Tags Access Advisor Revoke sessions

▼ Permissions policies (5 policies applied)

[Attach policies](#) [Add inline policy](#)

Policy name	Policy type	
AWSLambdaFullAccess	AWS managed policy	✕
AmazonS3FullAccess	AWS managed policy	✕
AmazonSageMakerFullA...	AWS managed policy	✕
Allow_CloudWatch_and_s3	Inline policy	✕
Lambda_InvokeEndpoint	Inline policy	✕

Figure 3: IAM role's policy: AWSLambdaFullAccess, AmazonS3FullAccess and Amazon-SageMakerFullAccess

Allow_CloudWatch_and_s3 Inline policy

Policy summary { } JSON Edit policy Simulate policy

```

1 {
2   "Version": "2012-10-17",
3   "Statement": [
4     {
5       "Effect": "Allow",
6       "Action": [
7         "logs:*",
8         "s3:*"
9       ],
10      "Resource": "arn:aws:logs:*:*:*"
11    },
12    {
13      "Effect": "Allow",
14      "Action": [
15        "s3:GetObject",
16        "s3:PutObject"
17      ],
18      "Resource": "arn:aws:s3::*:*"
19    }
20  ]
21 }

```

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 `InvokeEndpoint` See figure 5, click *create function*

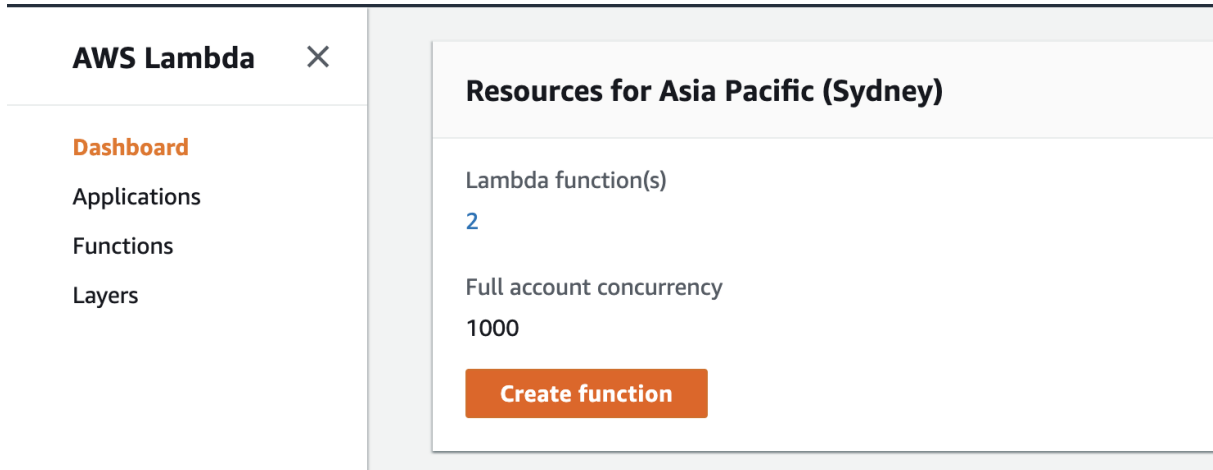
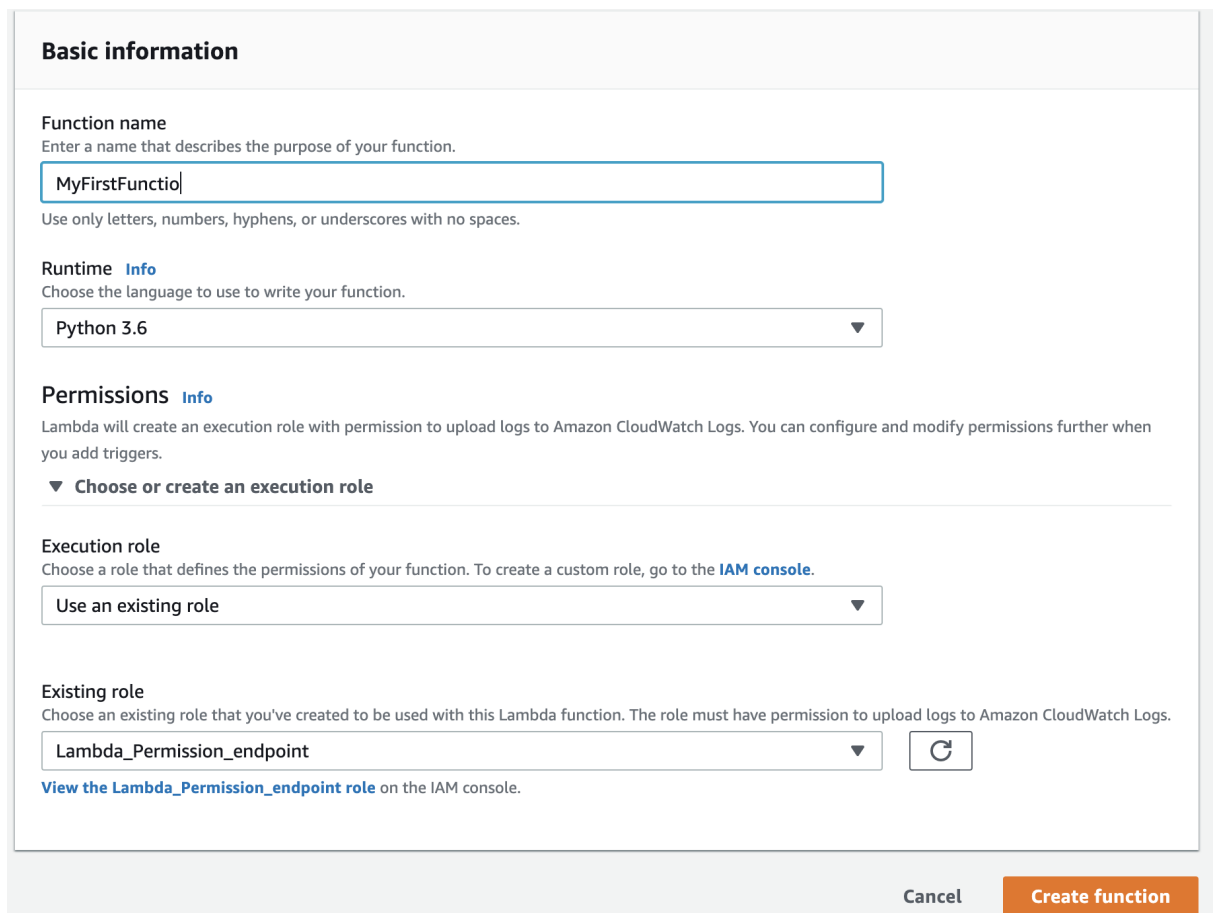


Figure 5: Create a new Lambda Function from Amazon User Interface

After that, give the name and language for your lambda function, 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*.



Basic information

Function name
Enter a name that describes the purpose of your function.

Use only letters, numbers, hyphens, or underscores with no spaces.

Runtime [Info](#)
Choose the language to use to write your function.

Permissions [Info](#)
Lambda will create an execution role with permission to upload logs to Amazon CloudWatch Logs. You can configure and modify permissions further when you add triggers.
▼ **Choose or create an execution role**

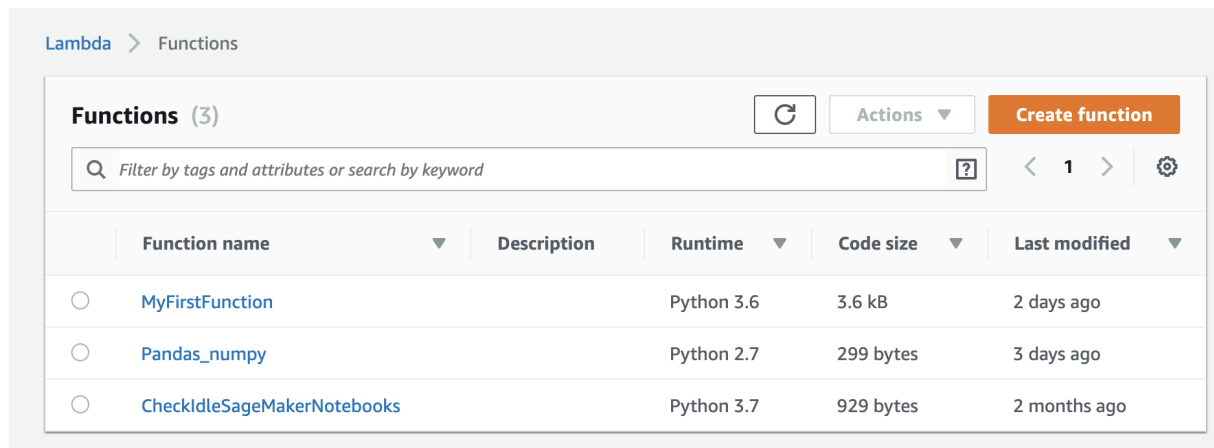
Execution role
Choose a role that defines the permissions of your function. To create a custom role, go to the [IAM console](#).

Existing role
Choose an existing role that you've created to be used with this Lambda function. The role must have permission to upload logs to Amazon CloudWatch Logs.

[View the Lambda_Permission_endpoint role](#) on the IAM console.

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.



The screenshot displays the AWS Lambda console's 'Functions' page. At the top, there's a breadcrumb 'Lambda > Functions'. Below this, the 'Functions (3)' header is followed by a refresh icon, an 'Actions' dropdown, and a prominent orange 'Create function' button. A search bar with the placeholder 'Filter by tags and attributes or search by keyword' is also present. The main content is a table listing three functions:

	Function name	Description	Runtime	Code size	Last modified
<input type="radio"/>	MyFirstFunction		Python 3.6	3.6 kB	2 days ago
<input type="radio"/>	Pandas_numpy		Python 2.7	299 bytes	3 days ago
<input type="radio"/>	CheckIdleSageMakerNotebooks		Python 3.7	929 bytes	2 months ago

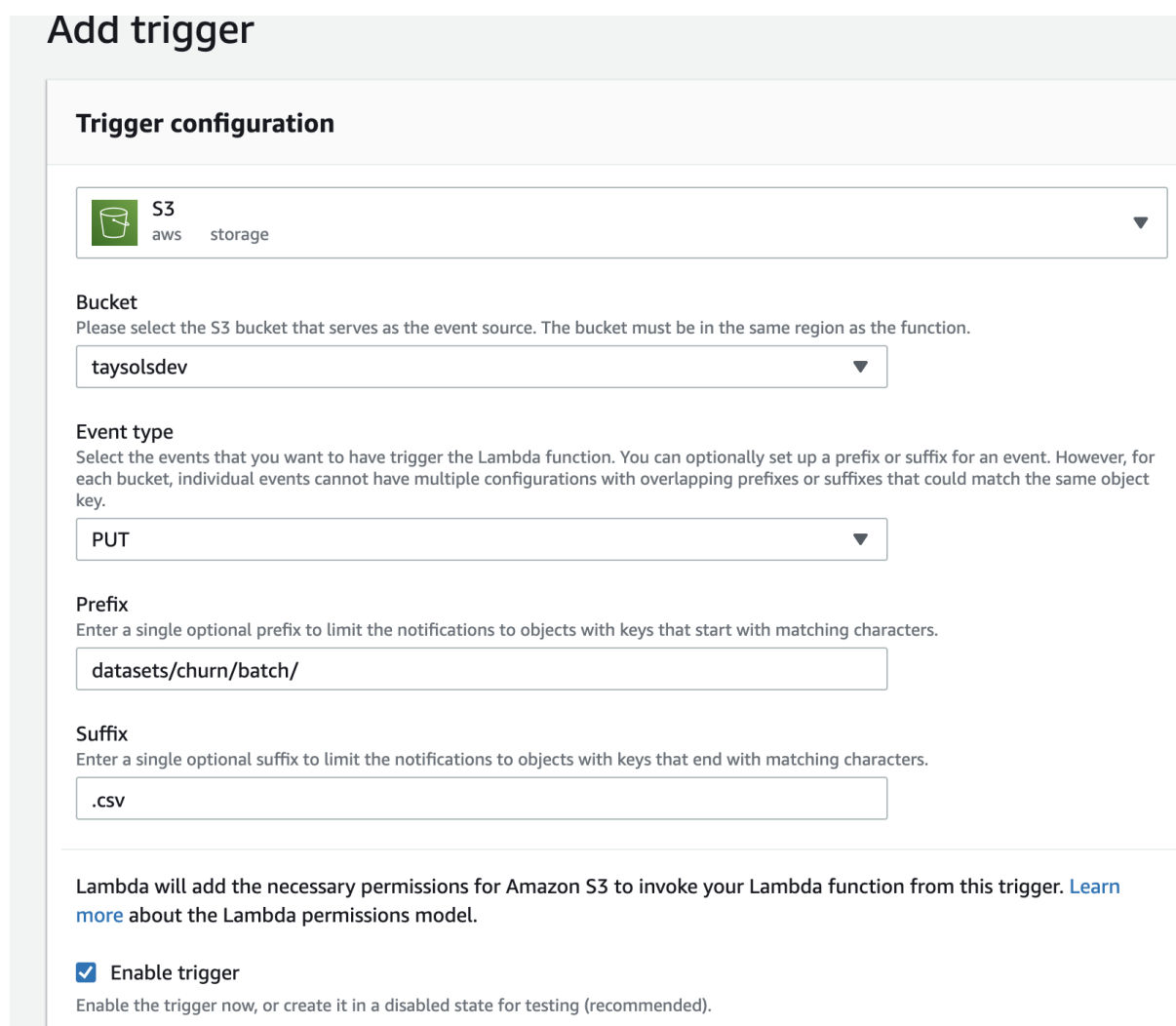
Figure 7: Check you have create a new lambda function

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.



Add trigger

Trigger configuration

 **S3**
aws storage ▼

Bucket
Please select the S3 bucket that serves as the event source. The bucket must be in the same region as the function.
taysolsdev ▼

Event type
Select the events that you want to have trigger the Lambda function. You can optionally set up a prefix or suffix for an event. However, for each bucket, individual events cannot have multiple configurations with overlapping prefixes or suffixes that could match the same object key.
PUT ▼

Prefix
Enter a single optional prefix to limit the notifications to objects with keys that start with matching characters.
datasets/churn/batch/

Suffix
Enter a single optional suffix to limit the notifications to objects with keys that end with matching characters.
.csv

Lambda will add the necessary permissions for Amazon S3 to invoke your Lambda function from this trigger. [Learn more](#) about the Lambda permissions model.

☒ **Enable trigger**
Enable the trigger now, or create it in a disabled state for testing (recommended).

Figure 8: Select S3 put as event trigger

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

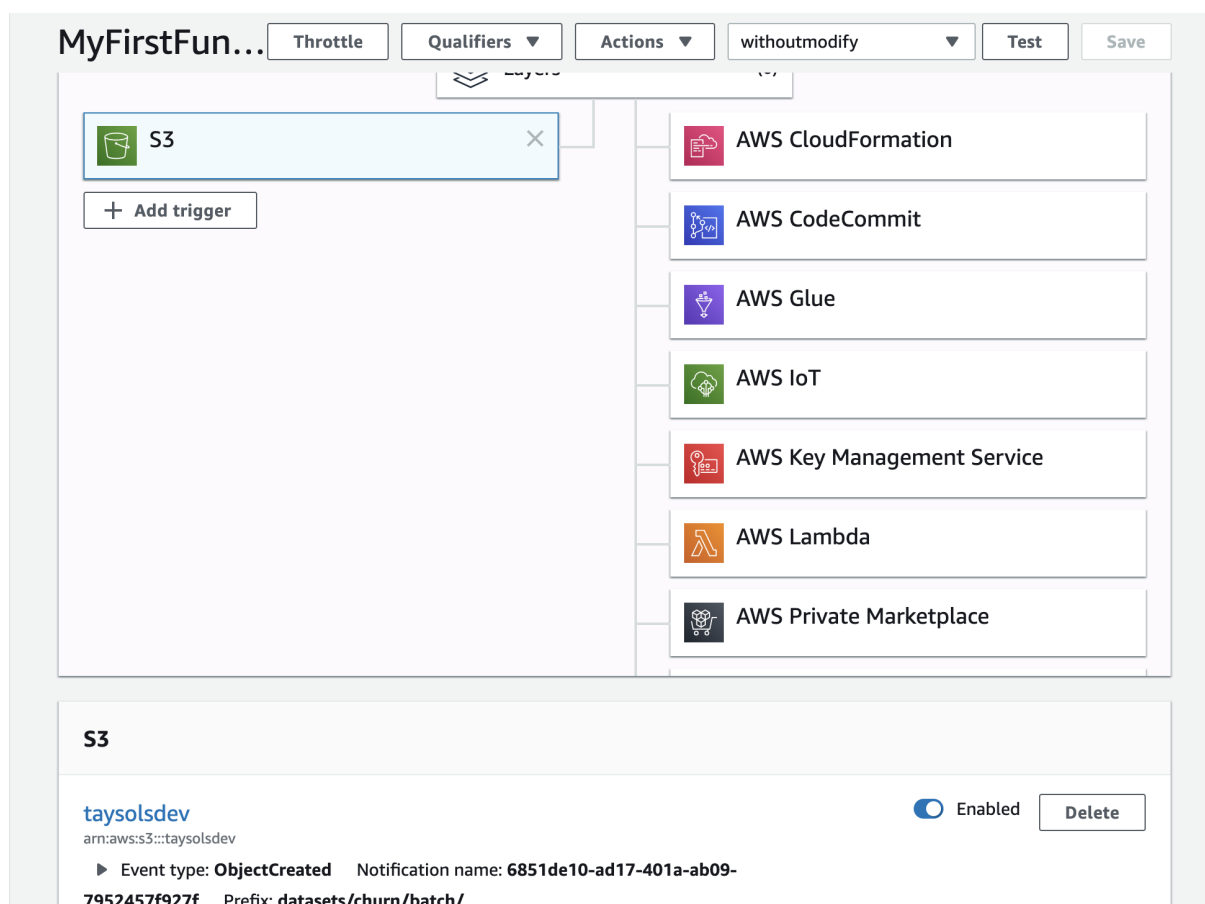


Figure 9: your lambda function

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:

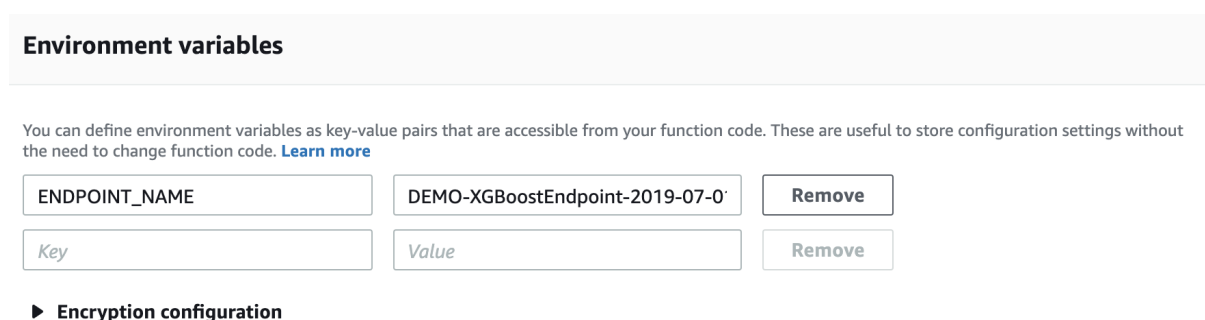


Figure 10: Add endpoint of model to your Lambda Function

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

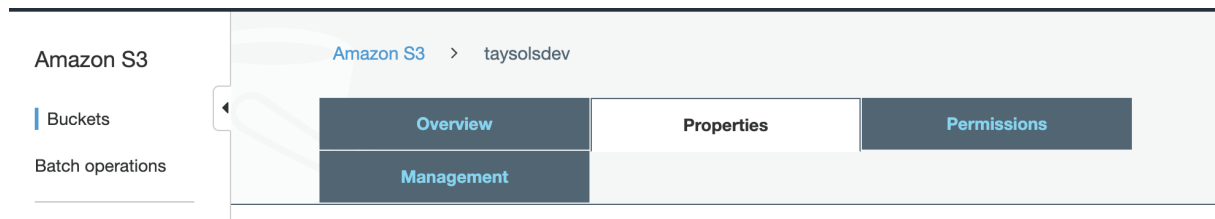


Figure 11

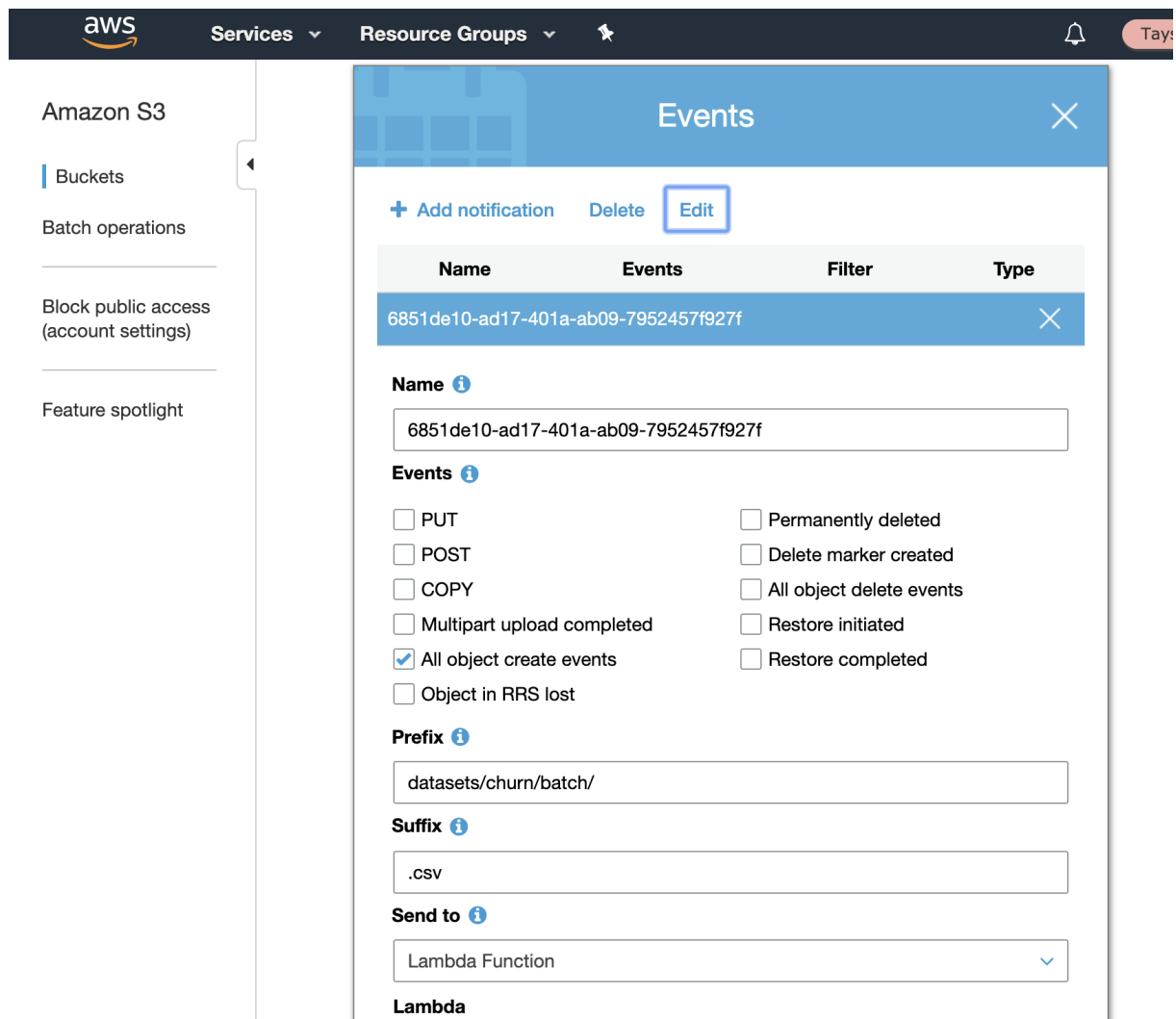


Figure 12

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**.

Configure test event ✕

A function can have up to 10 test events. The events are persisted so you can switch to another computer or web browser and test your function with the same events.

☒ Create new test event
☐ Edit saved test events

Event template

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

Handler [Info](#)

lambda_function.lambda_handler

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

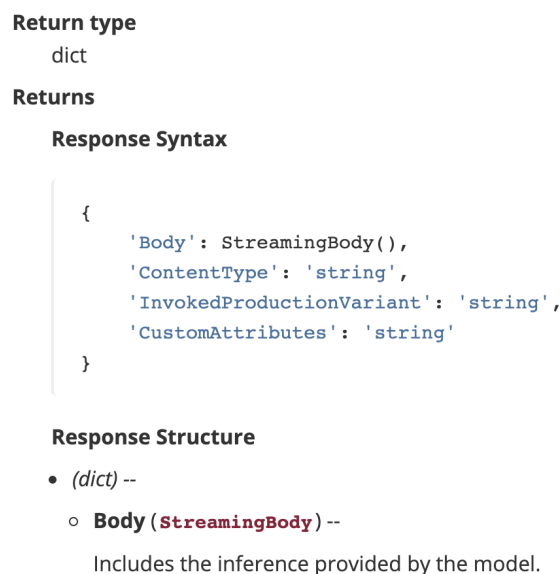


Figure 16: Invoke Endpoint Return Content

See figure 17

```

1  import pickle
2  from boto3.exceptions import ClientError
3  from time import strftime, gmtime
4  from json import dumps, loads, JSONEncoder, JSONDecoder
5  from six.moves import urllib
6  ## import UDF for lambda function
7  from help_function_lambda import read_csv
8  from help_function_lambda import prediction_probability
9  from help_function_lambda import predicted_label
10 from help_function_lambda import write_out_s3
11
12 ENDPOINT_NAME = os.environ['ENDPOINT_NAME'] # access environment variable values
13 runtime= boto3.client('runtime.sagemaker') # A low-level client representing Amazon SageMaker Runtime
14 s3 = boto3.resource('s3')
15 bucket = s3.Bucket('taysolsdev')
16 prefix = 'datasets/churn'
17
18 """
19 Make probability predictions for each observation and then write out the predictions to S3 bucket.
20 The main steps are described in UDF: look help_function_lambda.py
21
22 Parameters:
23 -----
24 event: dict type with JSON format
25     AWS Lambda uses this parameter to pass in event data to the handler
26 context: Lambda Context type
27     AWS Lambda uses this parameter to provide runtime information to your handler
28
29 Returns:
30 -----
31 predictions_probability:
32     the new list with predicted probabilities for each observation
33 """
34
35 def lambda_handler(event, context):
36     test_data_input = read_csv(event) # UDF to read in csv dataset
37     # you can print out the dataset to check
38     # print(test_data_input)
39     predictions_probability = prediction_probability(test_data_input)
40     # write out to s3 bucket
41     write_out_s3(event, predictions_probability)
42     return predictions_probability
43

```

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:

```

18
19 ENDPOINT_NAME = os.environ['ENDPOINT_NAME'] # access environment variable values
20 runtime= boto3.client('runtime.sagemaker') # A low-level client representing Amazon SageMaker Runtime
21 s3 = boto3.resource('s3')
22 bucket = s3.Bucket('taysolsdev')
23 prefix = 'datasets/churn'
24
25
26 """
27 Read in new csv dataset for predictions
28
29 Parameters:
30 -----
31     event: dict type with JSON format
32         AWS Lambda uses this parameter to pass in event data to the handle
33
34 Returns:
35 -----
36     test_data_input:
37         the new dataset that will be used for prediction
38 """
39 def read_csv(event):
40     # retrieve bucket name and file_key from the S3 event
41     bucket_name = event['Records'][0]['s3']['bucket']['name'] # should be used
42     #file_key = urllib.parse.unquote_plus(event['Records'][0]['s3']['object']['input_key'])
43     file_key = urllib.parse.unquote_plus(event['Records'][0]['s3']['object']['key'])
44     # get the file object
45     obj = s3.Object(bucket_name, file_key)
46     # get lines inside the csv
47     lines = obj.get()['Body'].read().split(b'\n')
48     # Read in CSV file
49     test_data_input = lines[0].decode() # first row
50     for r in lines[1:]:
51         test_data_input = test_data_input + '\n' + r.decode() # we need to decode for each row
52     return test_data_input
53

```

Figure 18: Read CSV

```

55
56 """
57 Make the prediction of probability for each observation and threshold (0.5 by default )
58
59 Parameters:
60 -----
61     test_data_input:
62         the new dataset that will be used for prediction
63
64 Returns:
65 -----
66     predictions_probability:
67         the new list with predicted probabilities for each observation
68 """
69 def prediction_probability(test_data_input):
70     response = runtime.invoke_endpoint(EndpointName=ENDPOINT_NAME,
71                                     ContentType='text/csv',
72                                     Body=test_data_input)
73     # get the list of predictions
74     predictions_probability= response['Body'].read().decode("utf-8").split(",") # we must decode explicitly as "utf-8"
75     return predictions_probability
76
77

```

Figure 19: Probability prediction based on endpoint

```

79
80 Predict the label for each observation based on predicted probability and threshold (0.5 by default )
81
82 Parameters:
83 -----
84     predictions_probability: list
85         the list of predicted probabilities for each observation
86     threshold: (optional) a float
87         the threshold we want to use for label decision. 0.5 by default
88
89 Returns:
90 -----
91     predictions_label:
92         the new list with predicted labels
93 """
94 def predicted_label(predictions_probability, threshold = 0.5):
95     predictions_label=[0 if float(x) < threshold else 1 for x in predictions_probability]
96     return predictions_label
97
98

```

Figure 20: Label prediction based on threshold

```

100 """
101 Write out the predictions to S3 bucket
102
103 Parameters:
104 -----
105     predictions_probability: list
106         the list of predicted probabilities for each observation
107     event: dict type with JSON format
108         AWS Lambda uses this parameter to pass in event data to the handle
109 """
110 def write_out_s3(event, predictions_probability):
111     # the output data must be bytes-like object, and split by '\n'
112     result = predictions_probability[0]
113     for item in predictions_probability[1:]:
114         result = result + "\n" + item
115     output_bytes = bytes(result.encode('UTF-8'))
116     # get the output bucket
117     bucket_name = event['Records'][0]['s3']['bucket']['name'] # should be used
118     #new_key = urllib.parse.unquote_plus(event['Records'][0]['s3']['object']['output_key'])
119
120     ## currently, write back to inoput datafile
121     bucket_name = event['Records'][0]['s3']['bucket']['name']
122     file_key = urllib.parse.unquote_plus(event['Records'][0]['s3']['object']['key'])
123     new_object = s3.Object(bucket_name, file_key)
124     new_object.put(Body=output_bytes)
125

```

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***. *Click 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, input 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-model/*](https://aws.amazon.com/cn/blogs/machine-learning/call-an-amazon-sagemaker-model/)
2. [*https://n2ws.com/blog/aws-automation/lambda-function-s3-event-triggers*](https://n2ws.com/blog/aws-automation/lambda-function-s3-event-triggers)