DATA SCIENCE

BUILDING AWS LAMBDA FUNCTION ON MACHINE LEARNING MODEL

Example of invoking Existing XGBoost Endpoint on S3 PUT Event

Yiran Jing

July 24, 2019

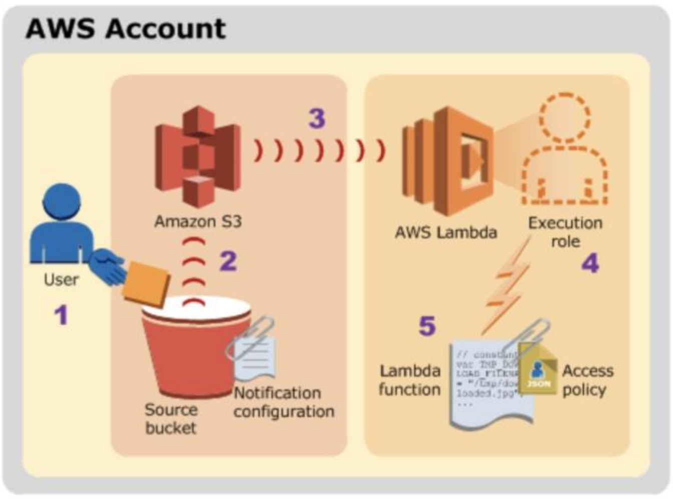
Contents

Introduction to Lambda Function

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 a execute Lambda Function on a S3 event trigger, for example, we could *execute a lambda function automatically on our built ML models when we push new dataset to S3 bucket*. After you write up your Lambda Function, everyone can easily use it to run a model on a new dataset by using a S3 put trigger, and this will automate the interaction between SageMaker and Lambda functions.

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:

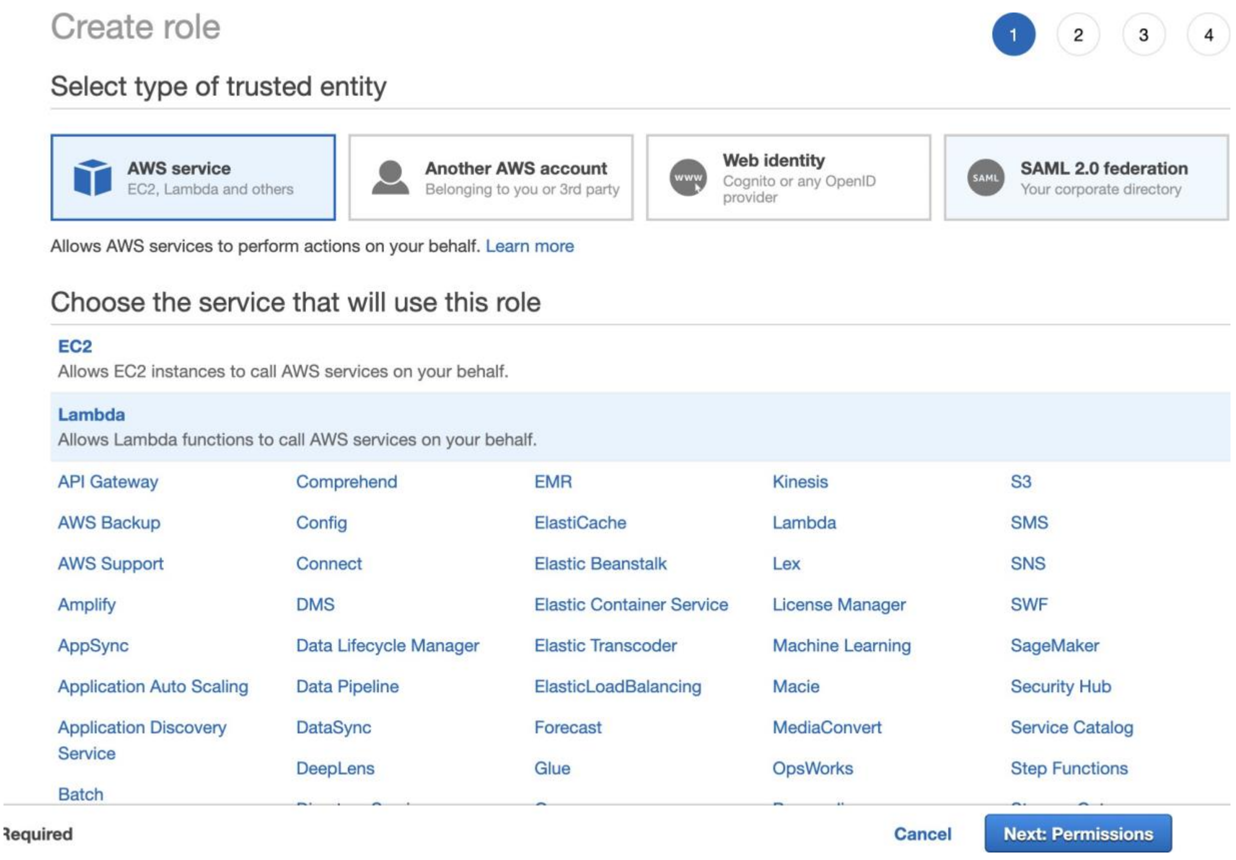
1. The user creates an object in a bucket.
2. Amazon S3 detects the object created event.
3. Amazon S3 invokes your Lambda function using the permissions provided by the execution role.
4. AWS Lambda executes the Lambda function, specifying the event as a parameter.

Step 1: 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 an 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.

1. Select Lambda and click *Next: Permission*.



Create role

Select type of trusted entity

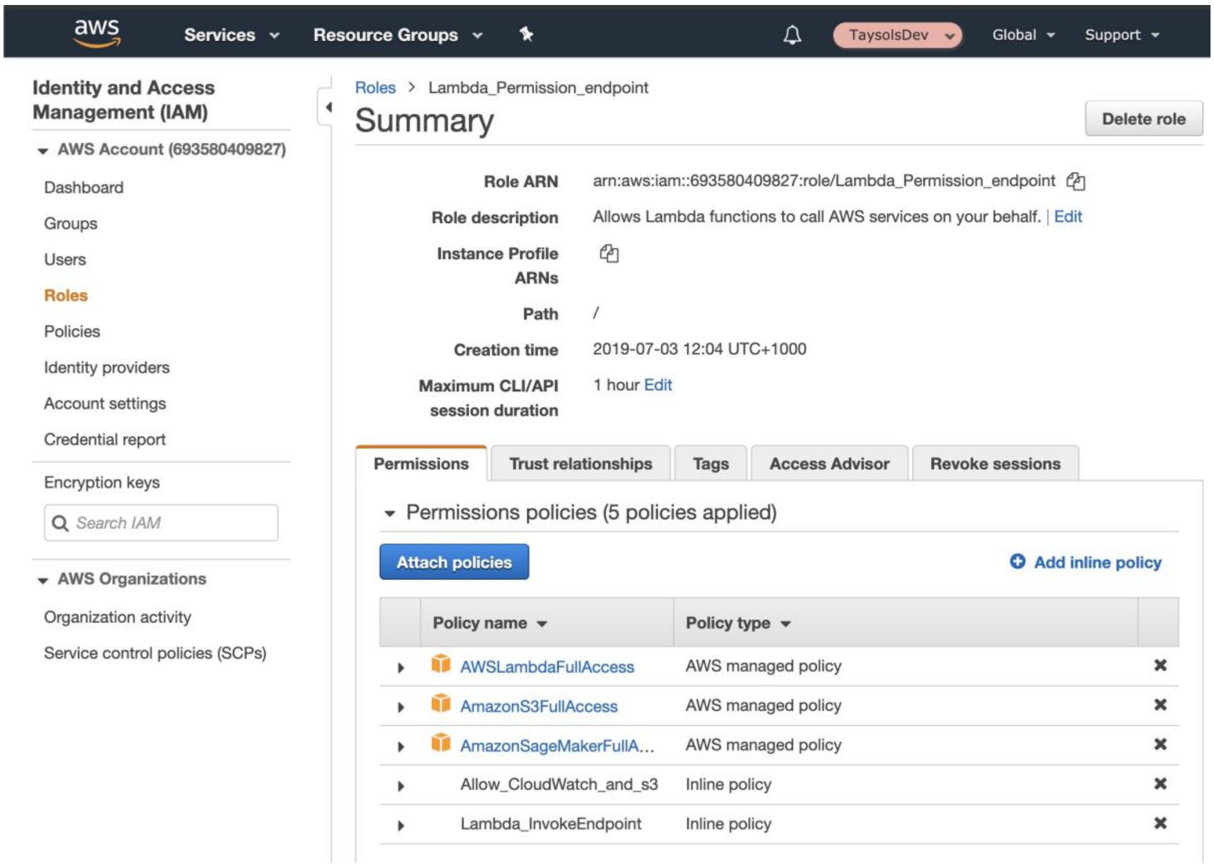
Allows AWS services to perform actions on your behalf. Learn more

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.

2. Then select the three policies:
   - **AWSLambdaFullAccess**
   - **AmazonS3FullAccess and**
   - **AmazonSageMakerFullAccess**



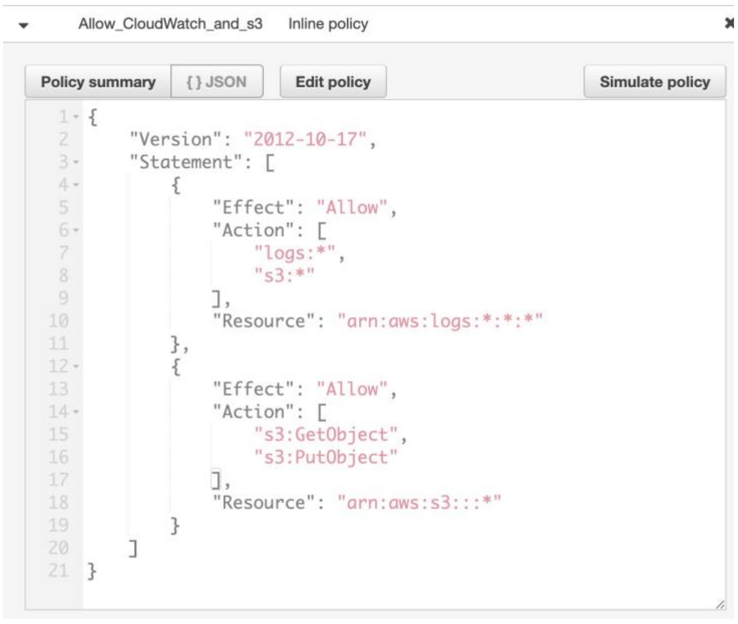
Summary

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	
Path	/
Creation time	2019-07-03 12:04 UTC+1000
Maximum CLI/API session duration	1 hour Edit

Permissions policies (5 policies applied)

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

3. You also need **CloudWatchPermission:**
   1. After you create the role (see the screenshot above), click *Add inline policy*.
   2. Click *{}JSON*, (see the screen shot below), and then copy paste the following JSON code to the Policy area. After that, click *Add*.

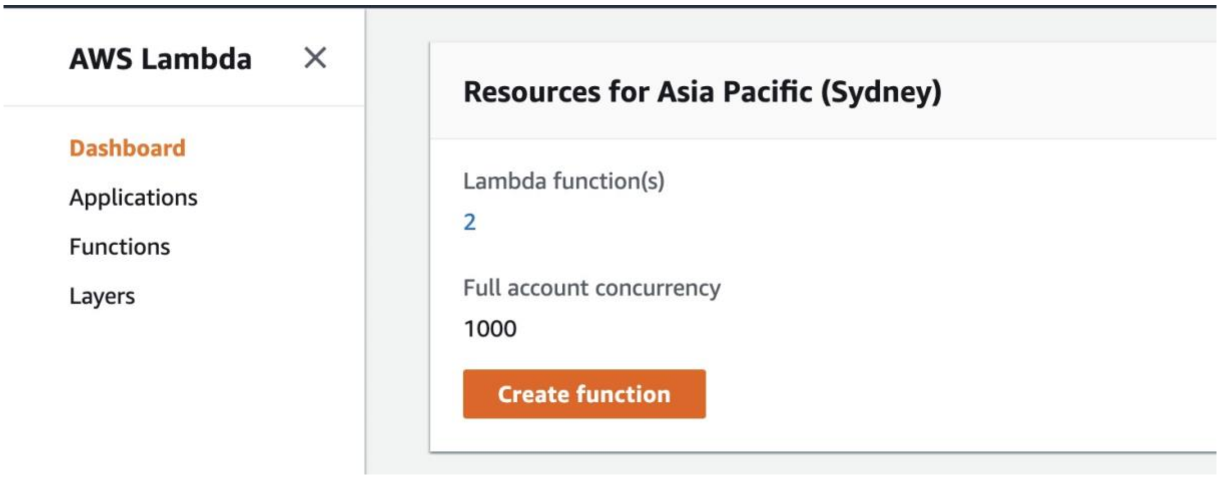


```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "logs:*",
                "s3:*"
            ],
            "Resource": "arn:aws:logs:*:*:*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:PutObject"
            ],
            "Resource": "arn:aws:s3:::*"
        }
    ]
}
```

Step 2: 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.

1. Go to *AWS Lambda dashboard*, Click *create function*

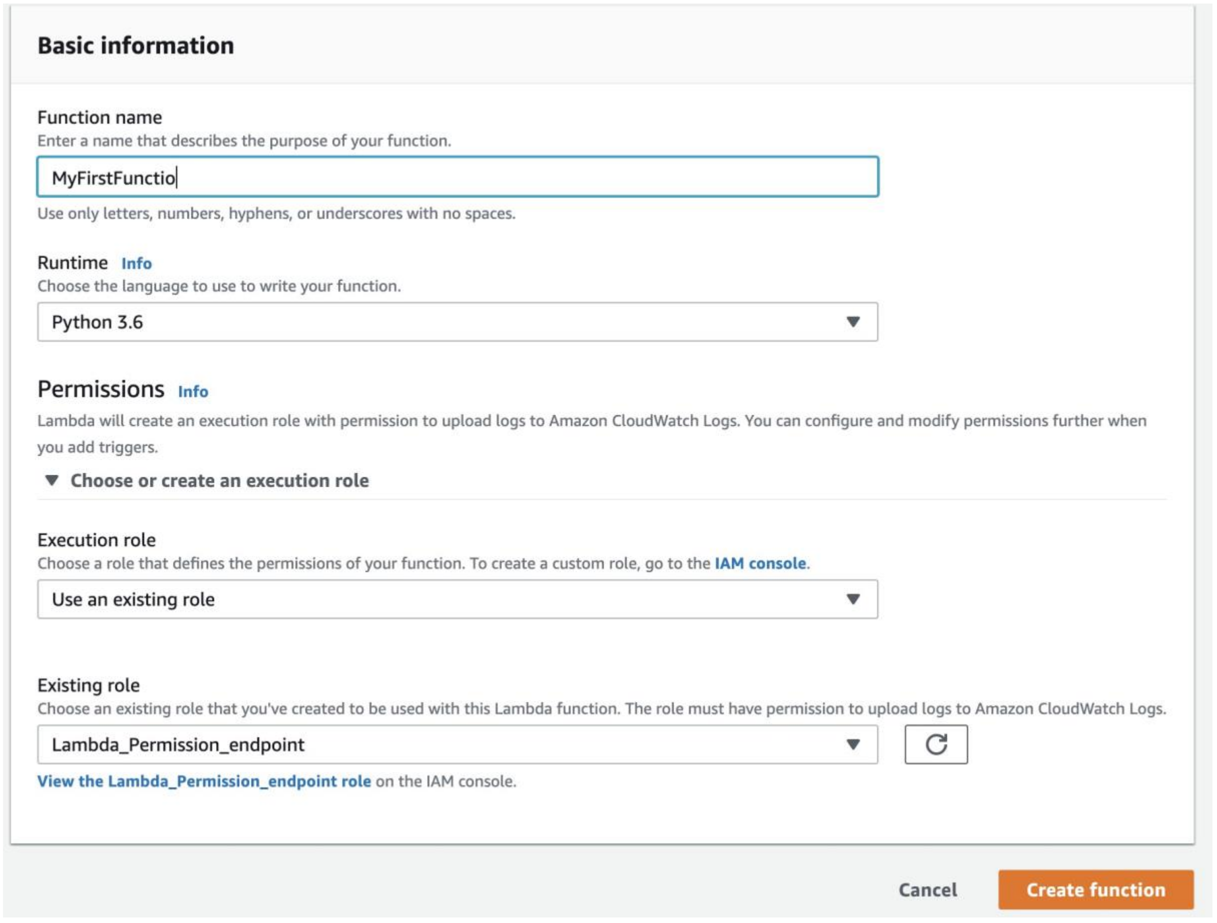


Resources for Asia Pacific (Sydney)

Lambda function(s)
2

Full account concurrency
1000

2. Give the name and language for your lambda function. Select *Python 3.6* and under execution role select *use an existing role*. Then under existing role select the role which you created earlier. In this example, the IAM role was *Lambda Permission endpoint* which was the role created in the preceding step. Your screen should looks like the screenshot below.



Basic information

Function name
Enter a name that describes the purpose of your function.
MyFirstFunctio
Use only letters, numbers, hyphens, or underscores with no spaces.

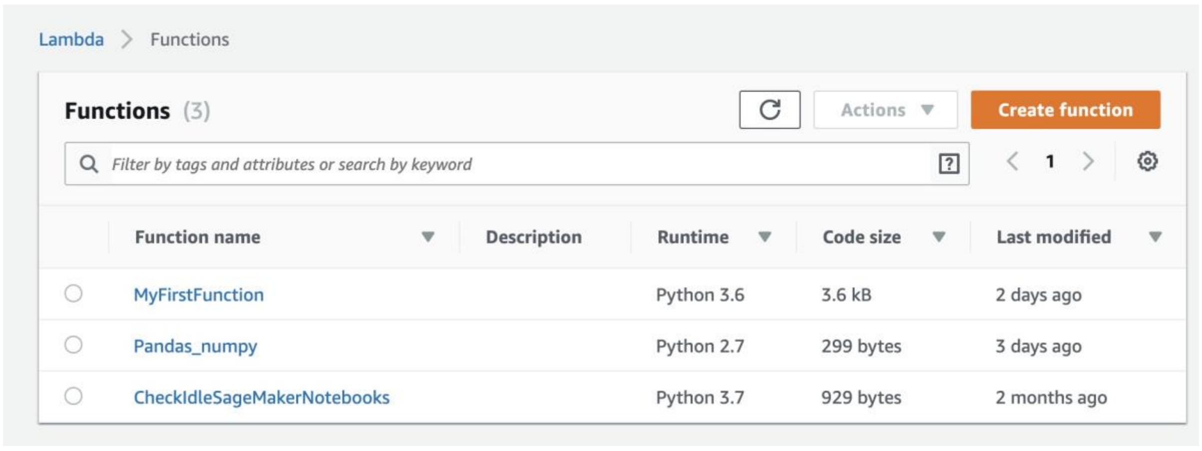
Runtime
Choose the language to use to write your function.
Python 3.6

Permissions
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.

Execution role
Choose a role that defines the permissions of your function. To create a custom role, go to the IAM console.
Use an existing role

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.
Lambda_Permission_endpoint
View the Lambda_Permission_endpoint role on the IAM console.

3. Then, click *Function* located in the Lambda Dashboard, check if the lambda function has been created. You should find the function name of your new lambda function. To further modify your lambda function, click the name of your lambda function for step 3.



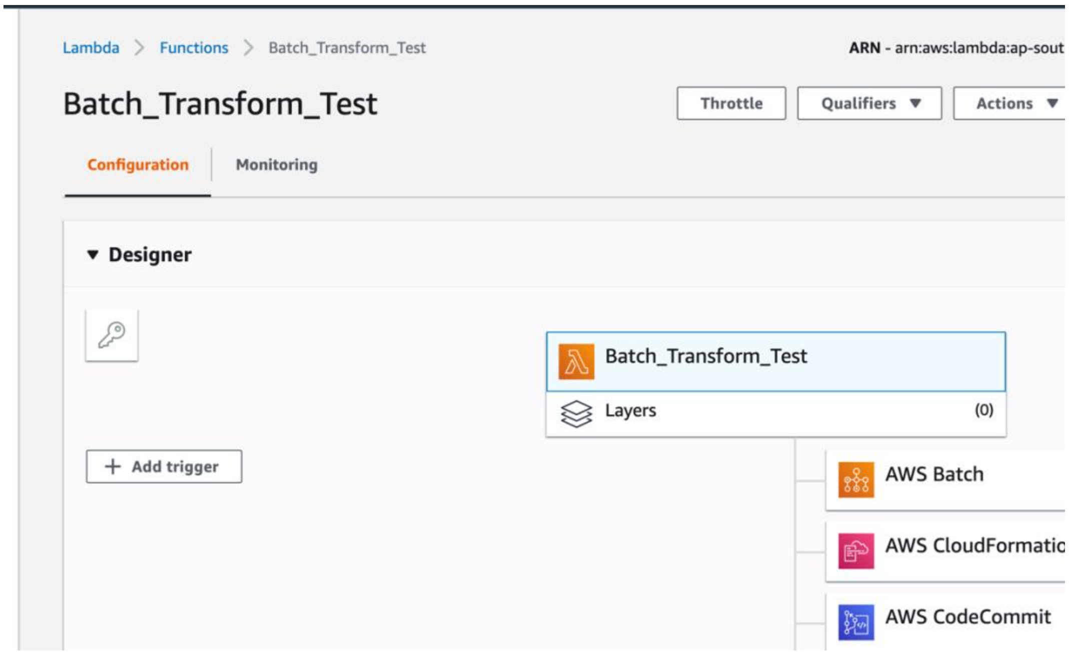
	Function name	Description	Runtime	Code size	Last modified
○	MyFirstFunction		Python 3.6	3.6 kB	2 days ago
○	Pandas_numpy		Python 2.7	299 bytes	3 days ago
○	CheckIdleSageMakerNotebooks		Python 3.7	929 bytes	2 months ago

Step 3: Build your Lambda Function

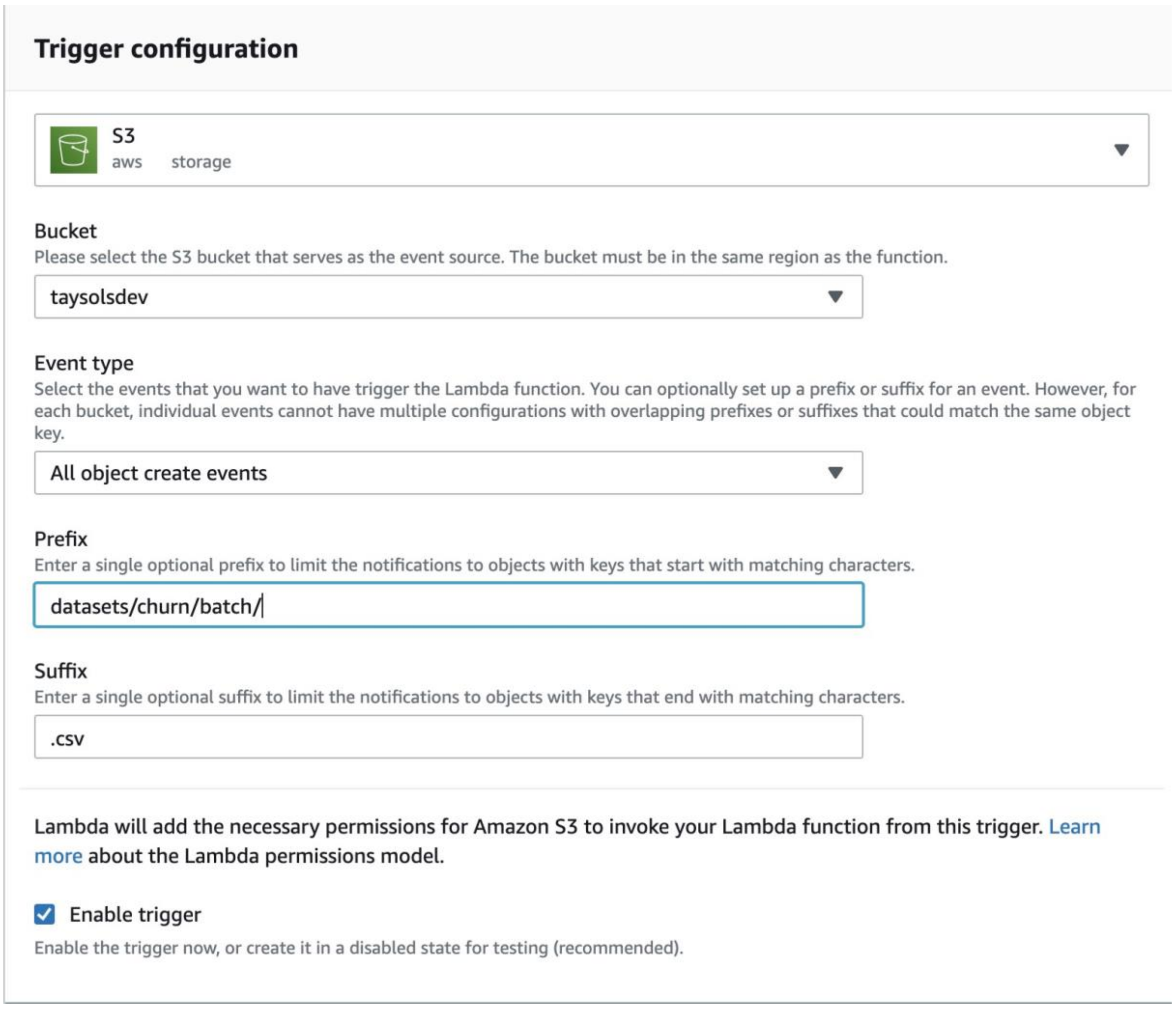
This example uses a lambda function called *MyFirstFunction*. For the following steps remember for the test to run on the most current code changes you must click save before clicking test.

Step 3.1: Create S3 Event Triggers

After you create a new empty lambda function, the next step is to add an 'S3 put' as an event trigger. This will mean that when an object is added to that folder the lambda function is triggered.



1. After you click the name of your new lambda function, your screen should look like the screenshot above. Then, Click + *Add trigger*
2. Firstly, Select *S3* as storage and appropriate bucket within S3 in this case 'taysolsdev'. Under event type select '*All object create event*' as the trigger event.
3. If a specific folder is allocated to trigger the event then add the folder path under Prefix. In our example, the *Prefix* is the path of the folder containing input dataset. Then add suffix and in our case this is *.csv*. your screen should look like the screenshot below


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.
All object create events

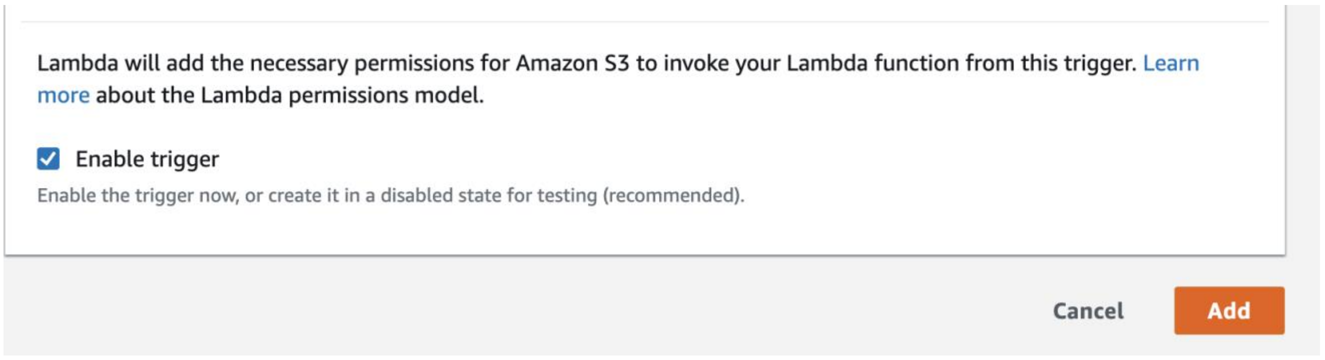
Prefix
Enter a single optional prefix to limit the notifications to objects with keys that start with matching characters.
datasets/churn/batch/

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.

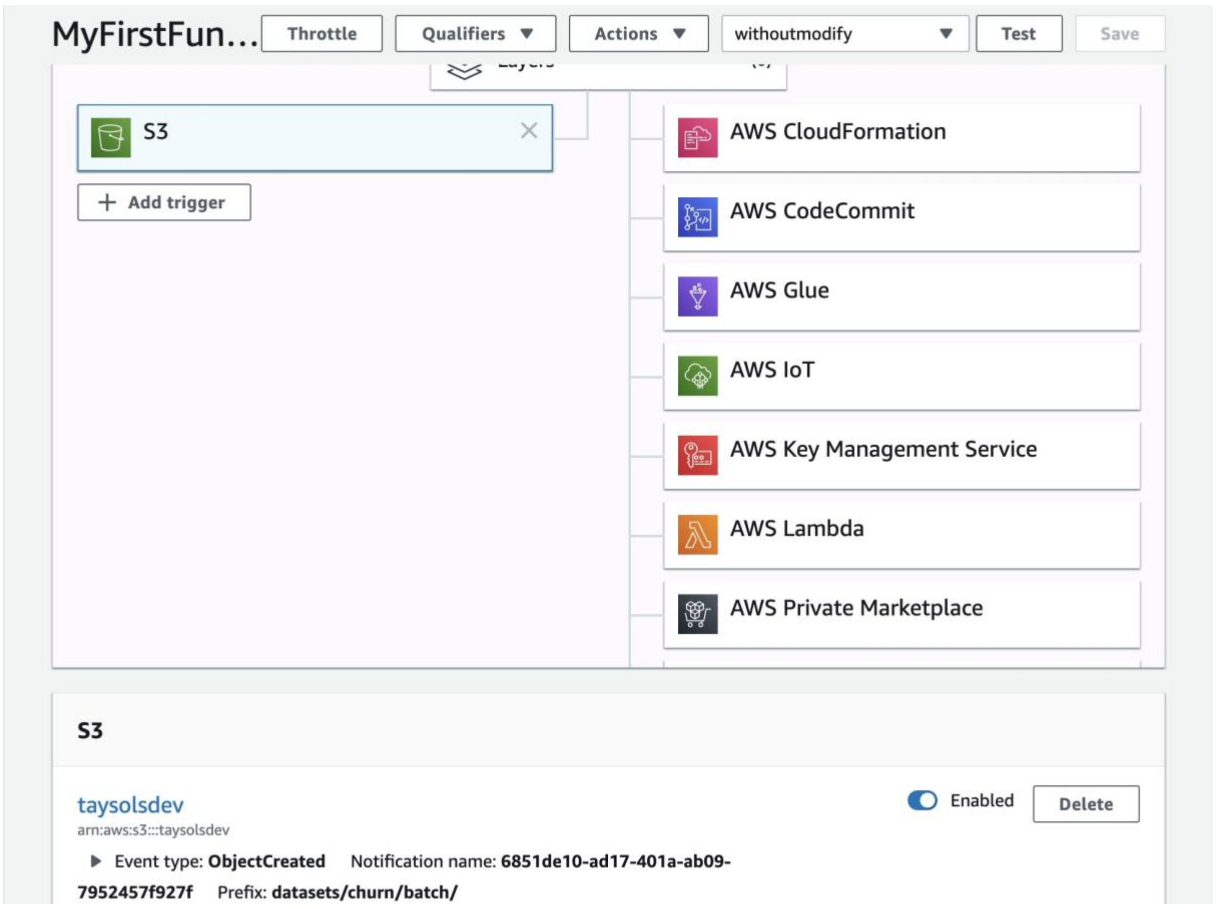
✓ Enable trigger
Enable the trigger now, or create it in a disabled state for testing (recommended).

4. Click *add*


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).

After you successfully add a S3 trigger, your screen should look like the screenshot below. Check the lambda function to ensure the event type and other details are correct. Here you can see the event type is ‘ObjectCreated’ and the Prefix is correct.



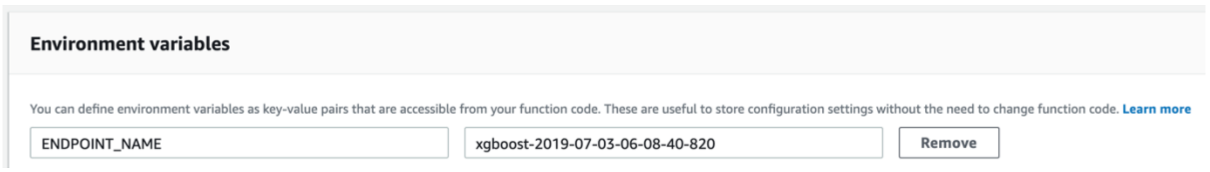
S3

taysolsdev
arn:aws:s3:::taysolsdev

▶ Event type: **ObjectCreated** Notification name: **6851de10-ad17-401a-ab09-7952457f927f** Prefix: **datasets/churn/batch/**

Step 3.2: Create Environment variables

Environment variable is a dynamic-named value that can affect the way running processes will behave on a computer. To build lambda function for batch job, we need three environment variables: KEY, BUCKET, and MODELNAME. The reason we use environmental variable is that these three variables can vary case by case, and it is much easier for user to modify the content of environmental variable without understanding the code of lambda function. See the screenshot below.



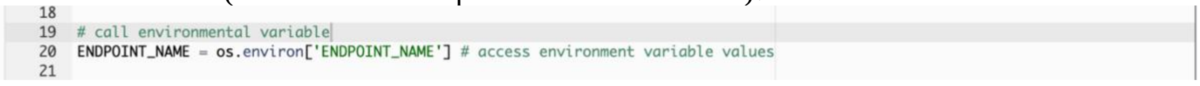
Environment variables

You can define environment variables as key-value pairs that are accessible from your function code. These are useful to store configuration settings without the need to change function code. Learn more

ENDPOINT_NAME	xgboost-2019-07-03-06-08-40-820	Remove

ENDPOINT_NAME: The endpoint name of trained model after we deploy model in sagemaker. You can copy it from Amazon SageMaker-Inference-Models

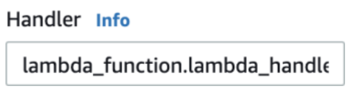
And then, you can call these environment variables through lambda function. See the screenshot below (will be further explained in next section):



```
# call environmental variable
ENDPOINT_NAME = os.environ['ENDPOINT_NAME'] # access environment variable values
```

Step 3.3: Modify Lambda handler

In this example, our *main function* is ***lambda_handler*** within ***lambda_function.py***. See the screenshot below. You can modify the main function in this section.



Handler Info

lambda_function.lambda_handle

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 create 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.

Below is the code used in the Lambda function to initiate a batch transform job.

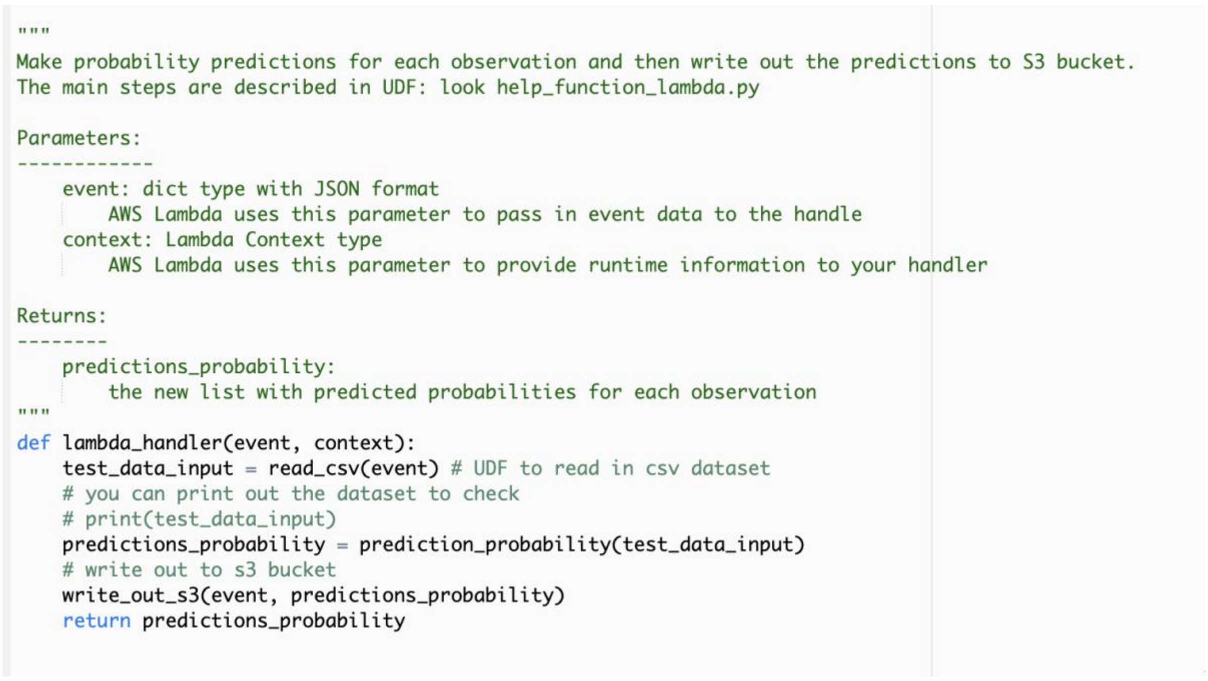
1. Firstly, we add libraries and relevant script needed to run the code:

```
import json
import boto3
import csv
import os
import io
import logging
import pickle
from botocore.exceptions import ClientError
from pprint import pprint
from time import strftime, gmtime
from json import dumps, loads, JSONEncoder, JSONDecoder
from six.moves import urllib

## inport UDF for lambda function
from help_function_lambda import read_csv
from help_function_lambda import prediction_probability
from help_function_lambda import predicted_label
from help_function_lambda import write_out_s3
```

2. Call environment variable and create sagemaker runtime object

```
ENDPOINT_NAME = os.environ['ENDPOINT_NAME'] # access environment variable values
runtime= boto3.client('runtime.sagemaker') # A low-level client representing Amazon SageMaker Runtime
s3 = boto3.resource('s3')
bucket = s3.Bucket('taysolsdev')
prefix = 'datasets/churn'
```

3. Write details of lambda handler function

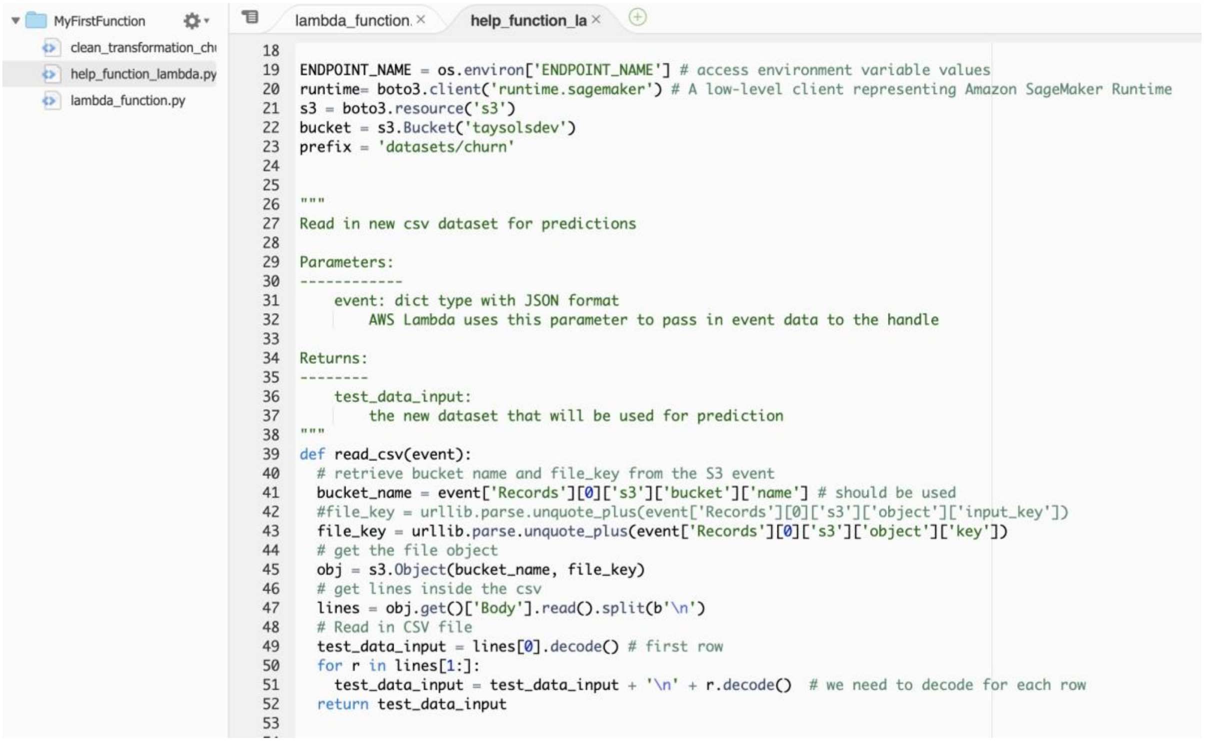


```python
"""
Make probability predictions for each observation and then write out the predictions to S3 bucket.
The main steps are described in UDF: look help_function_lambda.py

Parameters:
------------
    event: dict type with JSON format
        AWS Lambda uses this parameter to pass in event data to the handle
    context: Lambda Context type
        AWS Lambda uses this parameter to provide runtime information to your handler

Returns:
--------
    predictions_probability:
        the new list with predicted probabilities for each observation
"""
def lambda_handler(event, context):
    test_data_input = read_csv(event) # UDF to read in csv dataset
    # you can print out the dataset to check
    # print(test_data_input)
    predictions_probability = prediction_probability(test_data_input)
    # write out to s3 bucket
    write_out_s3(event, predictions_probability)
    return predictions_probability
```

4. Write Help function for lambda handler

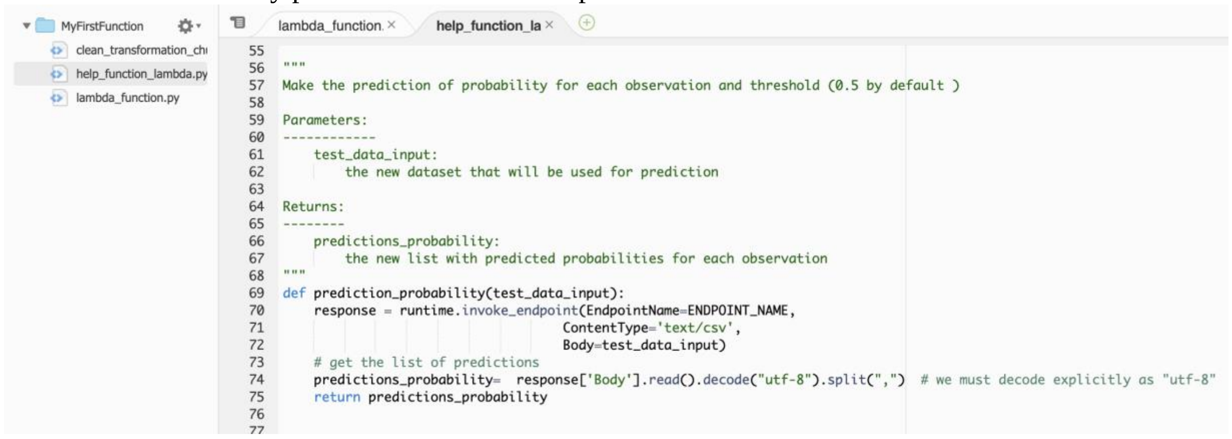
To make function easy to be understood and modify, I block functions as followings. See *help_function_lambda.py*

1. Read CSV



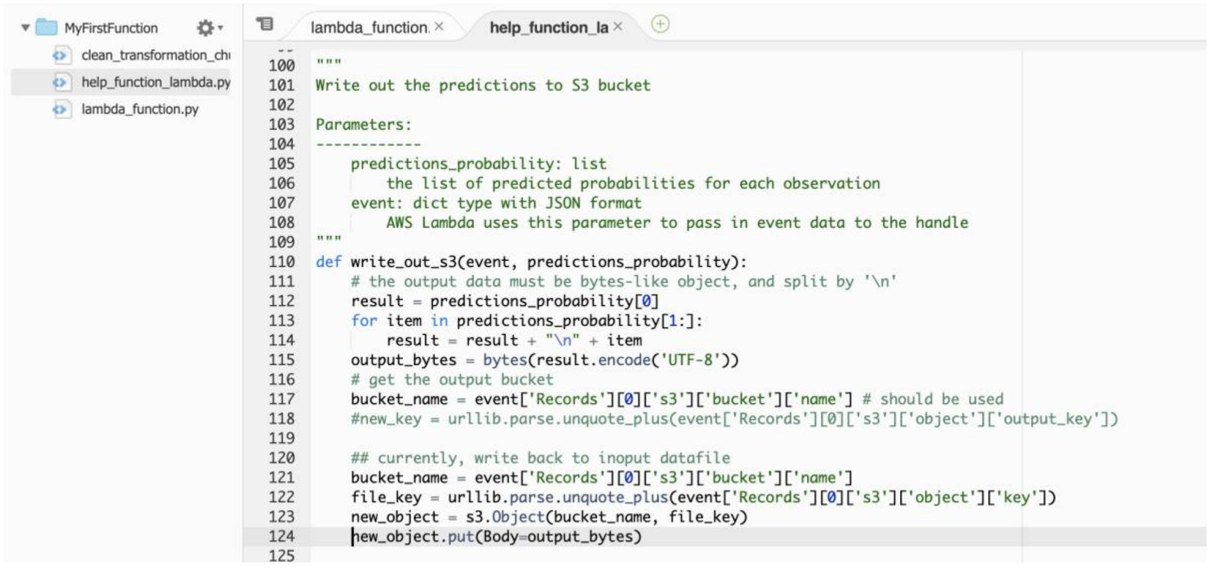
```python
ENDPOINT_NAME = os.environ['ENDPOINT_NAME'] # access environment variable values
runtime= boto3.client('runtime.sagemaker') # A low-level client representing Amazon SageMaker Runtime
s3 = boto3.resource('s3')
bucket = s3.Bucket('taysolsdev')
prefix = 'datasets/churn'


"""
Read in new csv dataset for predictions

Parameters:
------------
    event: dict type with JSON format
        AWS Lambda uses this parameter to pass in event data to the handle

Returns:
--------
    test_data_input:
        the new dataset that will be used for prediction
"""
def read_csv(event):
  # retrieve bucket name and file_key from the S3 event
  bucket_name = event['Records'][0]['s3']['bucket']['name'] # should be used
  #file_key = urllib.parse.unquote_plus(event['Records'][0]['s3']['object']['input_key'])
  file_key = urllib.parse.unquote_plus(event['Records'][0]['s3']['object']['key'])
  # get the file object
  obj = s3.Object(bucket_name, file_key)
  # get lines inside the csv
  lines = obj.get()['Body'].read().split(b'\n')
  # Read in CSV file
  test_data_input = lines[0].decode() # first row
  for r in lines[1:]:
    test_data_input = test_data_input + '\n' + r.decode()  # we need to decode for each row
  return test_data_input

```

2. Probability prediction based on endpoint



```
"""
Make the prediction of probability for each observation and threshold (0.5 by default )

Parameters:
------------
    test_data_input:
        the new dataset that will be used for prediction

Returns:
--------
    predictions_probability:
        the new list with predicted probabilities for each observation
"""
def prediction_probability(test_data_input):
    response = runtime.invoke_endpoint(EndpointName=ENDPOINT_NAME,
                                       ContentType='text/csv',
                                       Body=test_data_input)
    # get the list of predictions
    predictions_probability=  response['Body'].read().decode("utf-8").split(",")  # we must decode explicitly as "utf-8"
    return predictions_probability
```

3. Label prediction based on threshold
4. Write out output to S3 bucket



```
"""
Write out the predictions to S3 bucket

Parameters:
------------
    predictions_probability: list
        the list of predicted probabilities for each observation
    event: dict type with JSON format
        AWS Lambda uses this parameter to pass in event data to the handle
"""
def write_out_s3(event, predictions_probability):
    # the output data must be bytes-like object, and split by '\n'
    result = predictions_probability[0]
    for item in predictions_probability[1:]:
        result = result + "\n" + item
    output_bytes = bytes(result.encode('UTF-8'))
    # get the output bucket
    bucket_name = event['Records'][0]['s3']['bucket']['name'] # should be used
    #new_key = urllib.parse.unquote_plus(event['Records'][0]['s3']['object']['output_key'])

    ## currently, write back to inoput datafile
    bucket_name = event['Records'][0]['s3']['bucket']['name']
    file_key = urllib.parse.unquote_plus(event['Records'][0]['s3']['object']['key'])
    new_object = s3.Object(bucket_name, file_key)
    new_object.put(Body=output_bytes)
```

Common error and the way to fix

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*

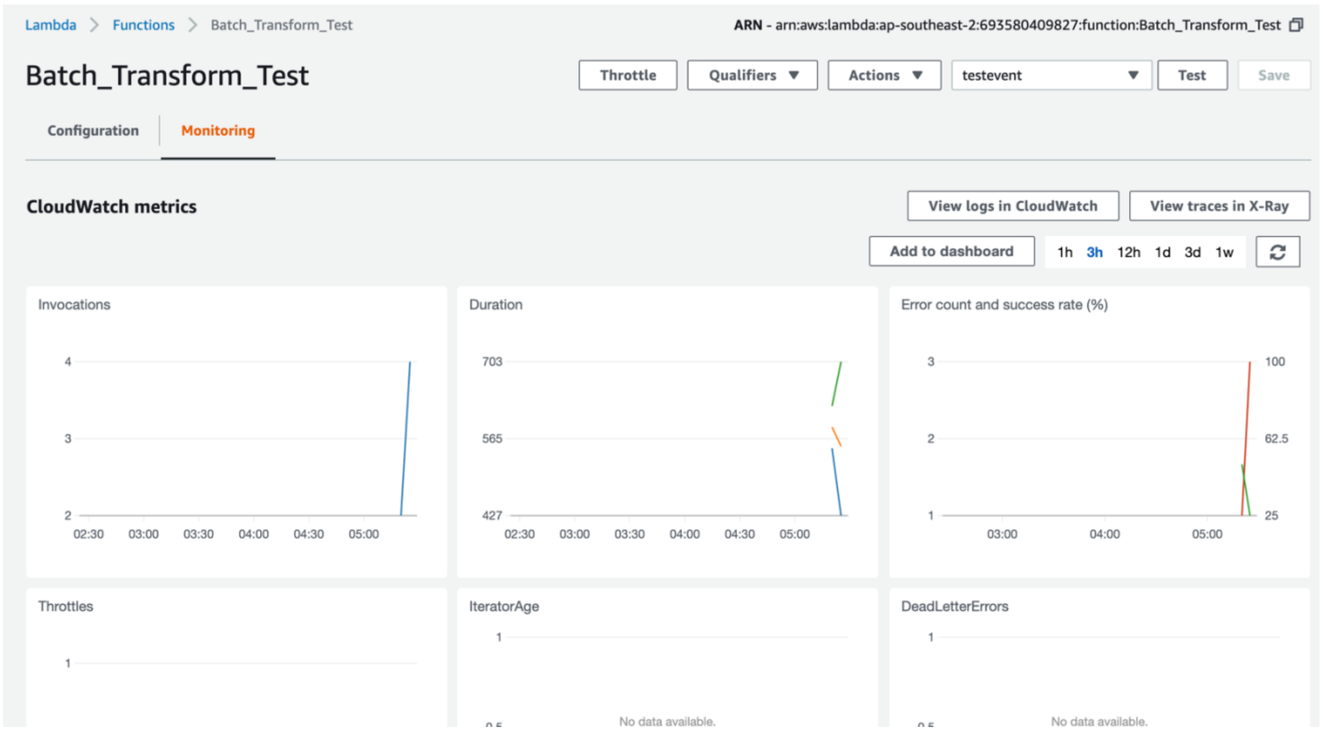
2. TypeError: expected string or bytes-like object

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 $bytes(json.dumps(predictions_probability, indent = 2).encode(^\prime UTF - 8^\prime)))$

Step 4: Test S3 Trigger Event: Check CloudWatch

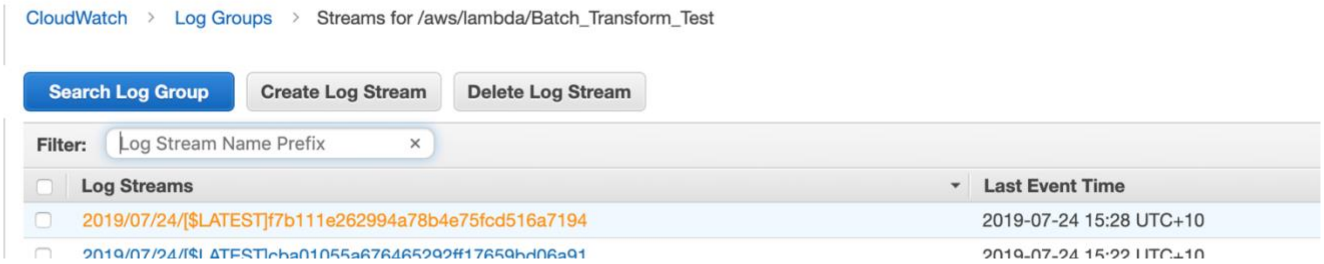
Testing lambda function using test event is different with the real trigger test. In other words, in this step, we need to upload new dataset to S3 to ensure that the lambda function is triggered automatically when we put new data to S3. The location of new data folder given when we create S3 event trigger. In this example, the folder location is *datasets/churn/batch/*

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.

To open Log Insights, click *View logs in CloudWatch,* then your screen should look like the screenshot below

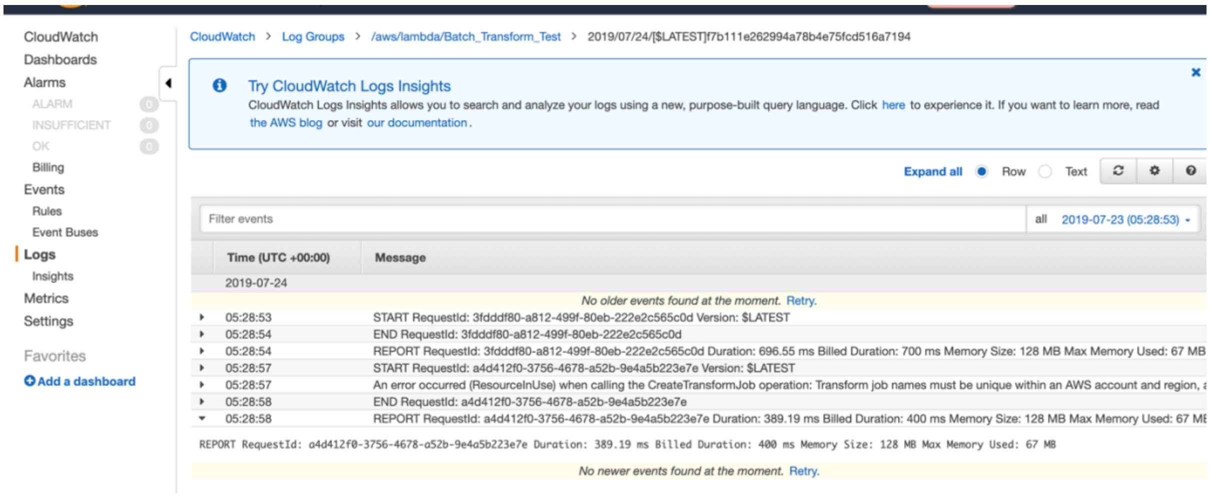

CloudWatch > Log Groups > Streams for /aws/lambda/Batch_Transform_Test

Search Log Group | Create Log Stream | Delete Log Stream

Filter: Log Stream Name Prefix

Log Streams	Last Event Time
2019/07/24/[$LATEST]f7b111e262994a78b4e75fcd516a7194	2019-07-24 15:28 UTC+10
2019/07/24/[$LATEST]cba01055a676465292ff17659bd06a91	2019-07-24 15:22 UTC+10

Click the first row, and then you can read details of running information of your last event trigger.



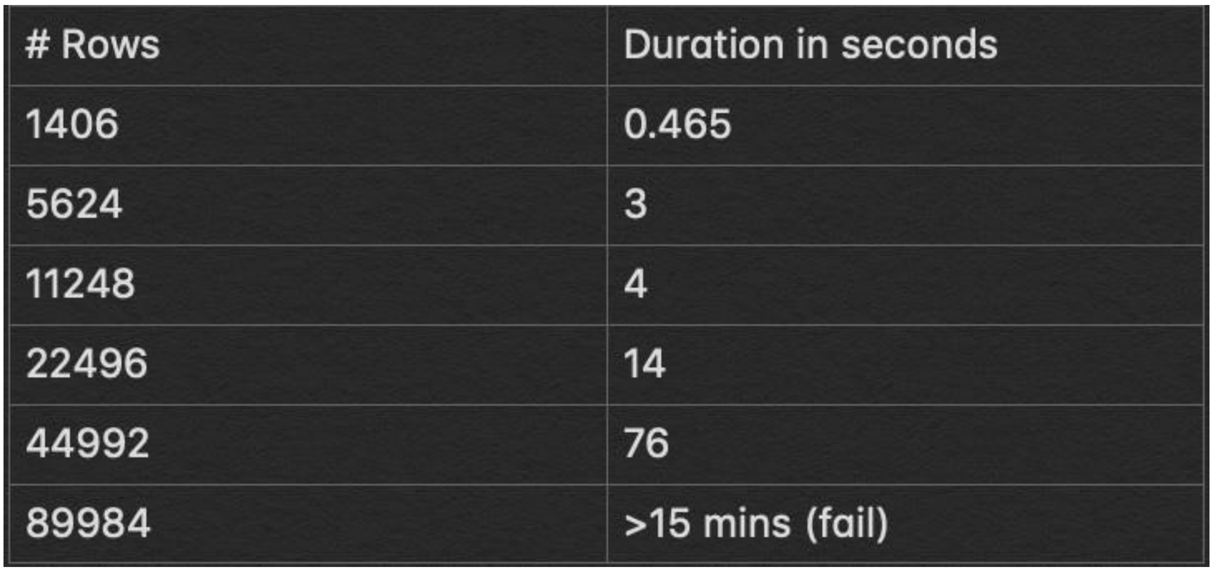
CloudWatch > Log Groups > /aws/lambda/Batch_Transform_Test > 2019/07/24/[$LATEST]f7b111e262994a78b4e75fcd516a7194

Try CloudWatch Logs Insights
CloudWatch Logs Insights allows you to search and analyze your logs using a new, purpose-built query language. Click here to experience it. If you want to learn more, read the AWS blog or visit our documentation.

```
Time (UTC +00:00)  Message
2019-07-24
                   No older events found at the moment. Retry.
05:28:53           START RequestId: 3fdddf80-a812-499f-80eb-222e2c565c0d Version: $LATEST
05:28:54           END RequestId: 3fdddf80-a812-499f-80eb-222e2c565c0d
05:28:54           REPORT RequestId: 3fdddf80-a812-499f-80eb-222e2c565c0d Duration: 696.55 ms Billed Duration: 700 ms Memory Size: 128 MB Max Memory Used: 67 MB
05:28:57           START RequestId: a4d412f0-3756-4678-a52b-9e4a5b223e7e Version: $LATEST
05:28:57           An error occurred (ResourceInUse) when calling the CreateTransformJob operation: Transform job names must be unique within an AWS account and region, a
05:28:58           END RequestId: a4d412f0-3756-4678-a52b-9e4a5b223e7e
05:28:58           REPORT RequestId: a4d412f0-3756-4678-a52b-9e4a5b223e7e Duration: 389.19 ms Billed Duration: 400 ms Memory Size: 128 MB Max Memory Used: 67 ME
REPORT RequestId: a4d412f0-3756-4678-a52b-9e4a5b223e7e Duration: 389.19 ms Billed Duration: 400 ms Memory Size: 128 MB Max Memory Used: 67 MB
                   No newer events found at the moment. Retry.
```

Actual Running time

From the screenshot below, there are expensive computing cost if the



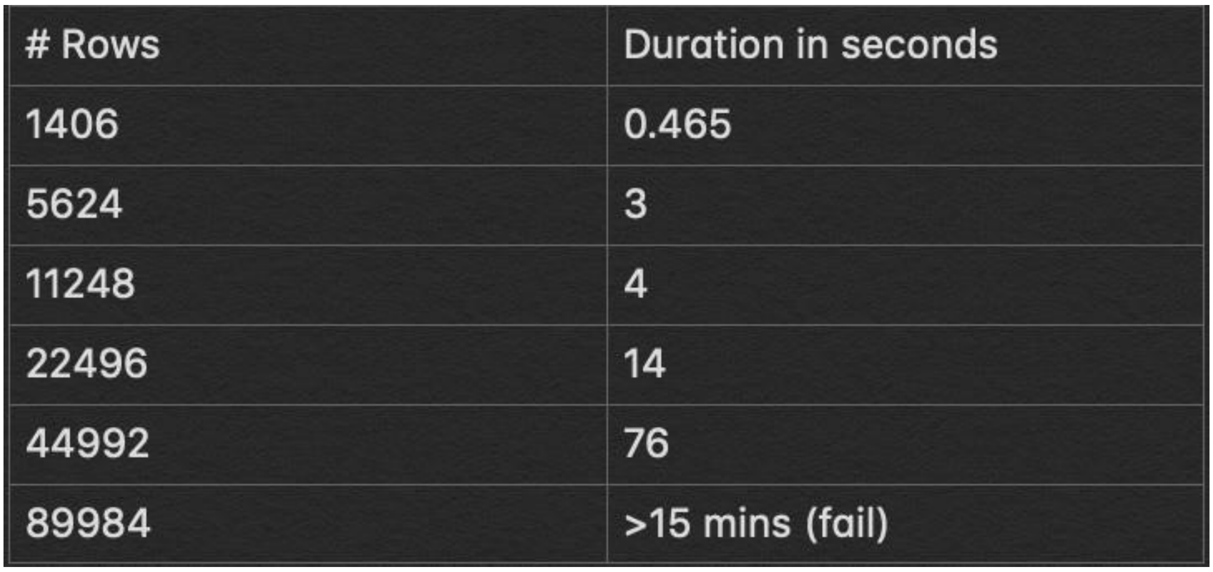
# Rows	Duration in seconds
1406	0.465
5624	3
11248	4
22496	14
44992	76
89984	>15 mins (fail)

number of rows exceed 50 000.

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*



# Rows	Duration in seconds
1406	0.465
5624	3
11248	4
22496	14
44992	76
89984	>15 mins (fail)