

DATA SCIENCE

---

# BUILD AWS LAMBDA FUNCTION CALLING BATCH TRANSFORMATION ON ML MODEL

---

Example of creating batch transformation job using trained model on S3 PUT  
Event

Yiran Jing

July 16, 2019

# Contents

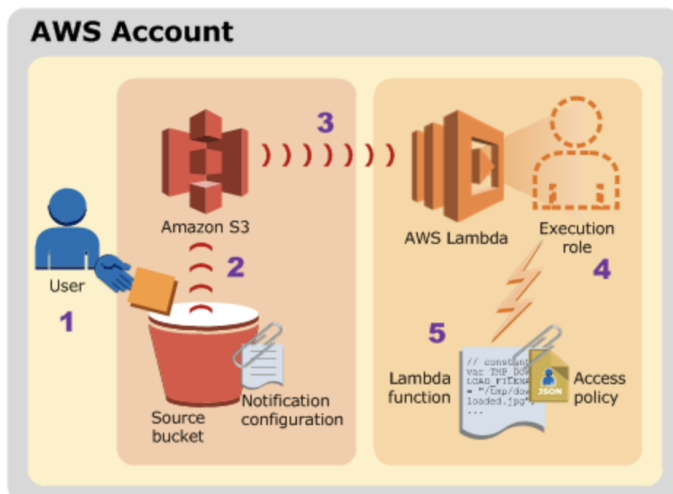
<b>1</b>	<b>Introduction to Lambda Function</b>	<b>1</b>
<b>2</b>	<b>Create IAM role that grants access to S3 bucket</b>	<b>2</b>
<b>3</b>	<b>Create an empty Lambda function</b>	<b>4</b>
<b>4</b>	<b>Build your Lambda Function</b>	<b>6</b>
4.1	S3 Event Triggers . . . . .	6
4.2	Configure test event . . . . .	7
4.3	Lambda Builders . . . . .	8
4.3.1	Lambda Handler with its Help function . . . . .	8
4.3.2	Lambda Handler Function . . . . .	8
4.3.3	Common error: Configuration is ambiguously defined. . . . .	9
<b>5</b>	<b>Test data</b>	<b>10</b>
5.1	Check CloudWatch . . . . .	10

# 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 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 policies ***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***.*

The screenshot shows the AWS IAM console interface. On the left is a navigation menu with options like Dashboard, Groups, Users, Roles (highlighted), Policies, Identity providers, Account settings, Credential report, Encryption keys, and AWS Organizations. The main content area is titled 'Summary' for the role 'Lambda\_Permission\_endpoint'. It displays the Role ARN, Role description, Instance Profile ARNs, Path, Creation time, and Maximum CLI/API session duration. Below the summary are tabs for Permissions, Trust relationships, Tags, Access Advisor, and Revoke sessions. The 'Permissions' tab is active, showing a list of 5 applied policies: AWSLambdaFullAccess, AmazonS3FullAccess, AmazonSageMakerFullAccess, Allow\_CloudWatch\_and\_s3, and Lambda\_InvokeEndpoint. Each policy entry includes a policy name, type, and a delete icon.

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

The screenshot shows the 'Edit policy' view for the 'Allow\_CloudWatch\_and\_s3' inline policy. It displays the JSON policy document in a code editor. The policy allows the 'logs:\*' and 's3:\*' actions on the resource 'arn:aws:logs:\*:\*:\*'. It also allows the 's3:GetObject' and 's3:PutObject' actions on the resource 'arn:aws:s3:::\*'.

```

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

### 3 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*

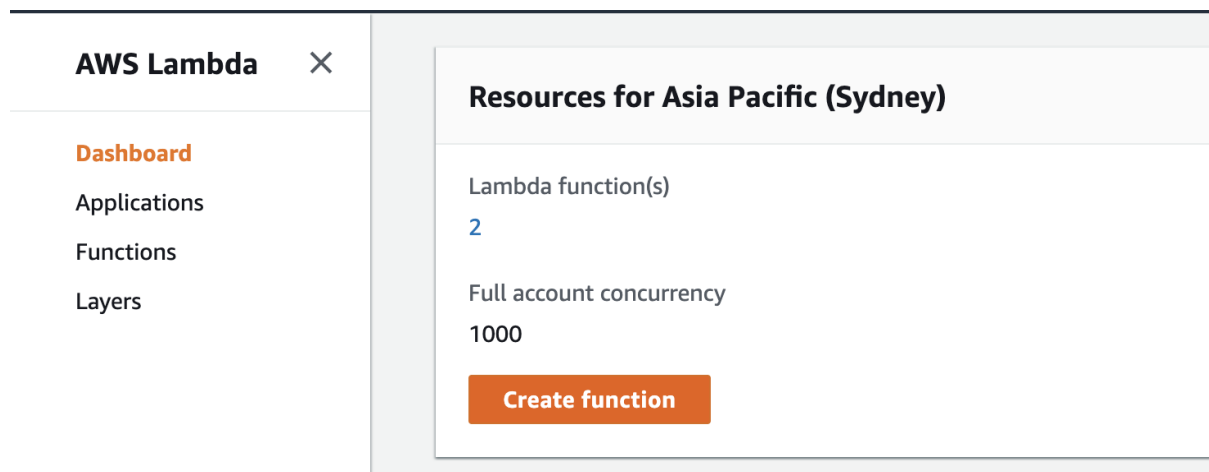


Figure 5: Create a new Lambda Function from Amazon User Interface

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

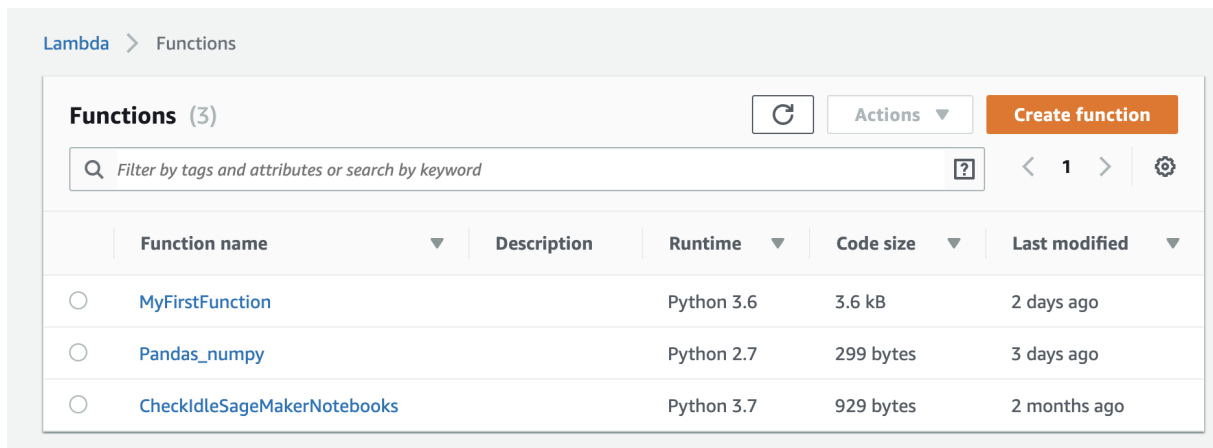
**Permissions** [Info](#)  
Lambda will create an execution role with permission to upload logs to Amazon CloudWatch Logs. You can configure and modify permissions further when you add triggers.  
▼ **Choose or create an execution role**

**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

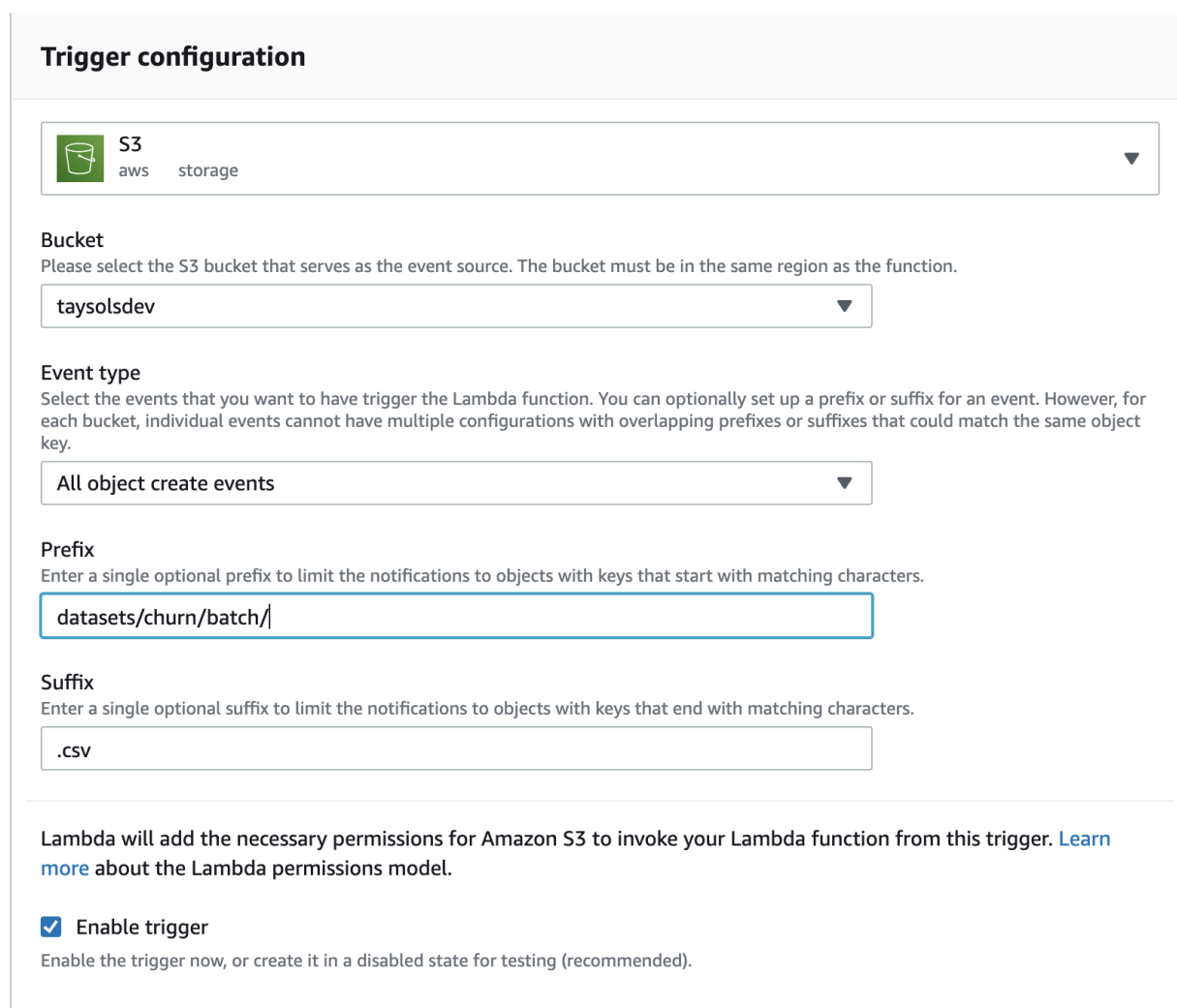
## 4 Build your Lambda Function

This example uses lambda function called **Batch\_Transform\_Test**.

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*.


### 4.1 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 All Object create event** 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 folder containing input dataset**. Suffix is **.csv** since our dataset is csv.



The screenshot shows the 'Trigger configuration' section of the AWS Lambda console. It is configured for an S3 event trigger. The 'Bucket' is set to 'taysolsdev'. The 'Event type' is 'All object create events'. The 'Prefix' is 'datasets/churn/batch/'. The 'Suffix' is '.csv'. There is a checkbox for 'Enable trigger' which is checked. A link 'Learn more about the Lambda permissions model.' is provided.

**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).

Figure 8: Select S3 put as event trigger

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



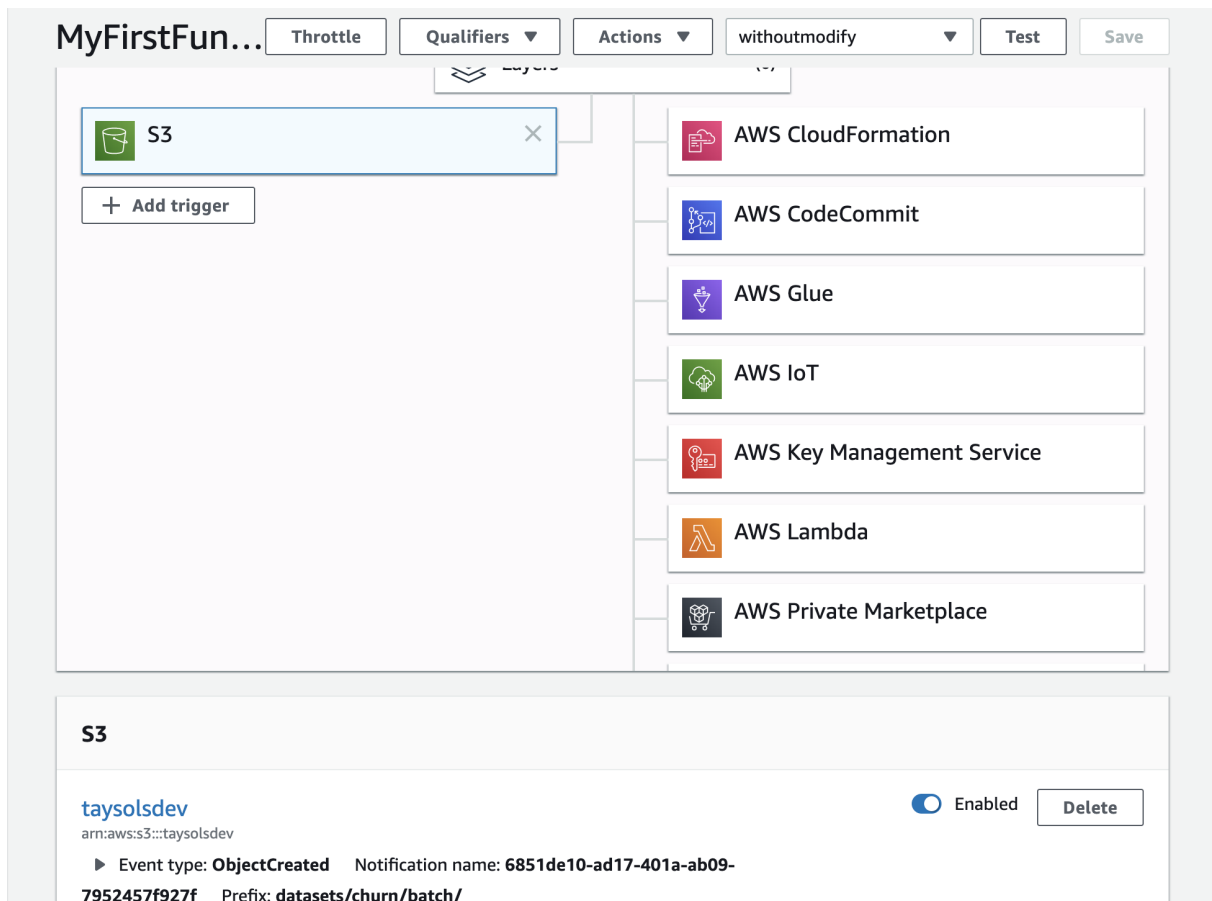


Figure 9: your lambda function

## 4.2 Configure test event

The event is *dict* type with *JSON* format. The **test event** is used for debug your Lambda function. See figure 10, Select **Create new test event** and then choose **Amazon S3 put**.

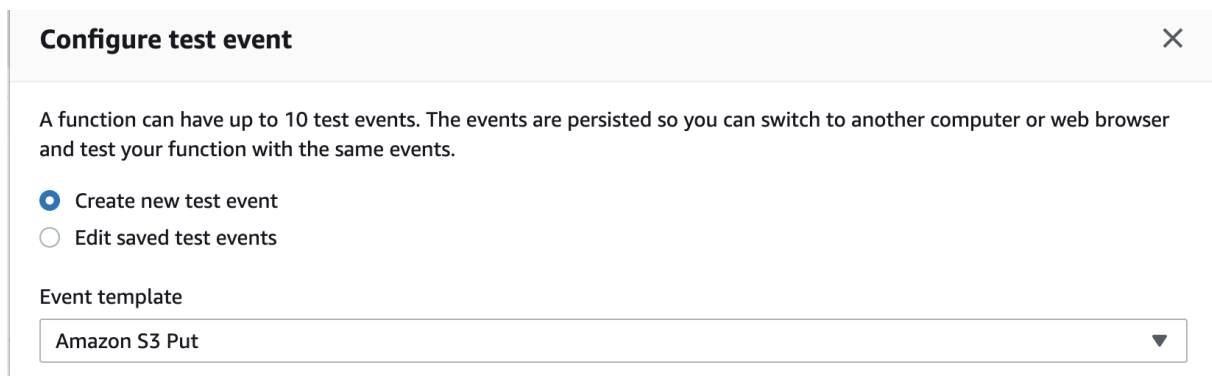


Figure 10: Add endpoint of model to your Lambda Function

you need to modify *bucket name* and the *key*(the path of the folder your input dataset). For example, the test event used in our case see figure 11.

Saved test event

testevent
▼

↺

```

1 {
2   "Records": [
3     {
4       "eventVersion": "2.0",
5       "eventSource": "aws:s3",
6       "awsRegion": "ap-southeast-2",
7       "eventTime": "1970-01-01T00:00:00.000Z",
8       "eventName": "ObjectCreated:Put",
9       "userIdentity": {
10        "principalId": "693580409827"
11      },
12      "s3": {
13        "s3SchemaVersion": "1.0",
14        "bucket": {
15          "name": "taysolsdev",
16          "arn": "arn:aws:s3:::taysolsdev"
17        },
18        "object": {
19          "key": "datasets/churn/batch/"
20        }
21      }
22    }
23  ]
24 }

```

Delete

Cancel

Save

Figure 11: Modify the content of test event template

## 4.3 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 13.

### 4.3.1 Lambda Handler with its Help function

In this example, our main function is *lambda\_handler* within *lambda\_function.py*, see figure 12

Handler
[Info](#)

lambda\_function.lambda\_handle

Figure 12: Lambda Handler information

### 4.3.2 Lambda Handler Function

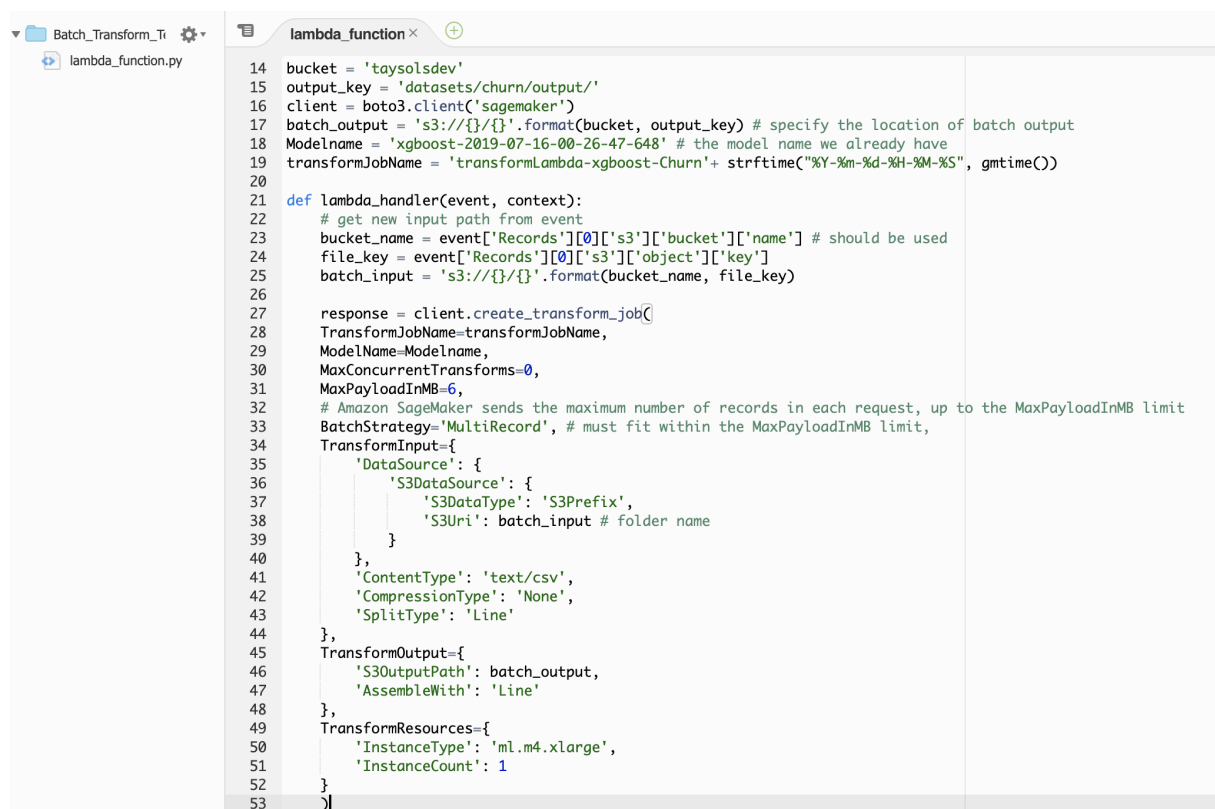
At the time you create a Lambda function, you specify a handler, which is a function in your code, that AWS Lambda can invoke when the service executes your code. I show the

example that how to creating a handler function in Python.

In other use case, you can only modify the following variables: **bucket**, **output\_key**, **batch\_input**, **Modelname**, **transformJobName**. See figure 13

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. **SplitType** - The method to use to split the transform job's data files into smaller batches. Splitting is necessary when the total size of each object is too large to fit in a single request.



```
14 bucket = 'taysolsdev'
15 output_key = 'datasets/churn/output/'
16 client = boto3.client('sagemaker')
17 batch_output = 's3://{}/{}'.format(bucket, output_key) # specify the location of batch output
18 Modelname = 'xgboost-2019-07-16-00-26-47-648' # the model name we already have
19 transformJobName = 'transformLambda-xgboost-Churn'+ strftime("%Y-%m-%d-%H-%M-%S", gmtime())
20
21 def lambda_handler(event, context):
22     # get new input path from event
23     bucket_name = event['Records'][0]['s3']['bucket']['name'] # should be used
24     file_key = event['Records'][0]['s3']['object']['key']
25     batch_input = 's3://{}/{}'.format(bucket_name, file_key)
26
27     response = client.create_transform_job(
28         TransformJobName=transformJobName,
29         ModelName=Modelname,
30         MaxConcurrentTransforms=0,
31         MaxPayloadInMB=6,
32         # Amazon SageMaker sends the maximum number of records in each request, up to the MaxPayloadInMB limit
33         BatchStrategy='MultiRecord', # must fit within the MaxPayloadInMB limit,
34         TransformInput={
35             'DataSource': {
36                 'S3DataSource': {
37                     'S3DataType': 'S3Prefix',
38                     'S3Uri': batch_input # folder name
39                 }
40             },
41             'ContentType': 'text/csv',
42             'CompressionType': 'None',
43             'SplitType': 'Line'
44         },
45         TransformOutput={
46             'S3OutputPath': batch_output,
47             'AssembleWith': 'Line'
48         },
49         TransformResources={
50             'InstanceType': 'ml.m4.xlarge',
51             'InstanceCount': 1
52         }
53     )
```

Figure 13: Lambda Handler function

#### 4.3.3 Common error: 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 Test data

### 5.1 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.