Using Lambda with AWS S3 Buckets

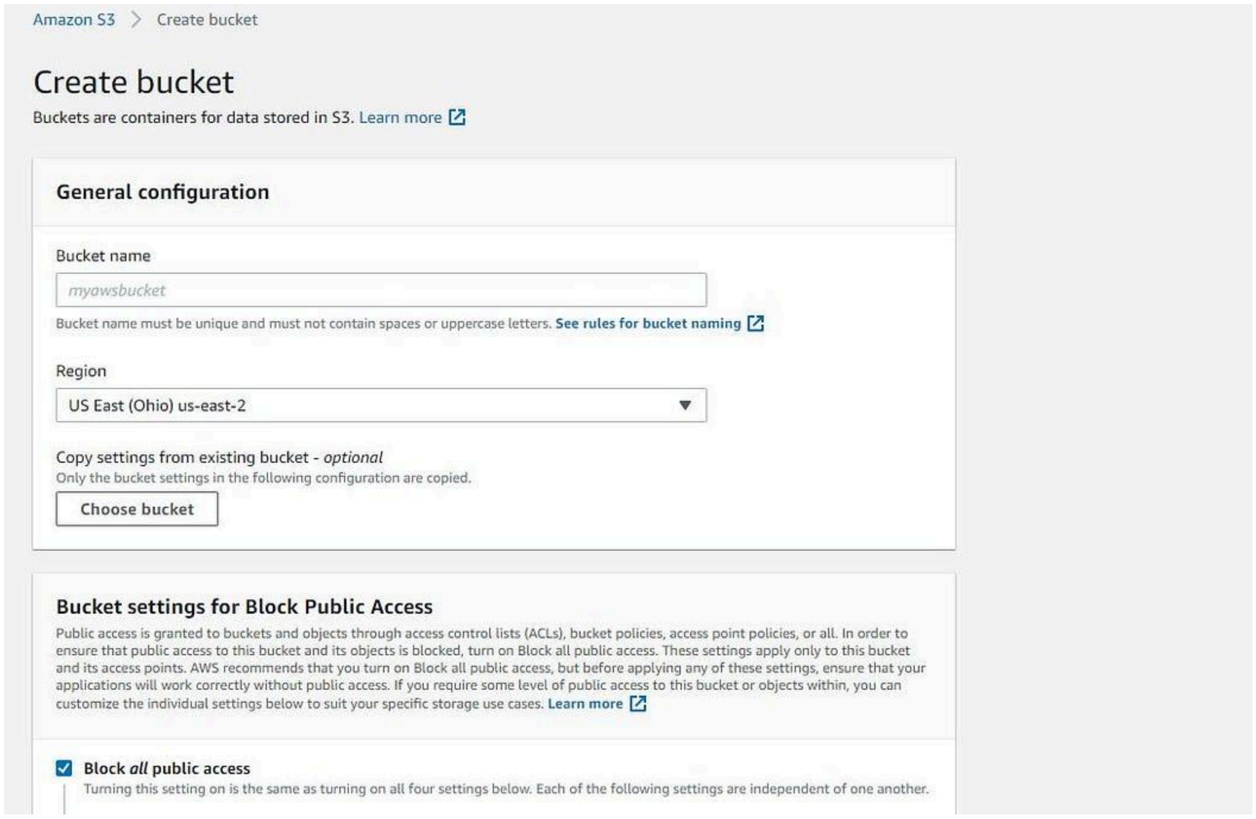
An **S3 bucket** is simply a storage space in AWS cloud for any kind of data (Eg., videos, code, AWS templates etc.). Every directory and file inside an S3 bucket can be uniquely identified using a key which is simply it's path relative to the root directory (which is the bucket itself). For example, "car.jpg" or "images/car.jpg".

Say you are receiving XML data from three different gas meters straight into an AWS S3 bucket. You want to sort the XML files into three separate folders based on which gas meter the data comes from. The only way to know the data source is to look inside the XML files, which look like this:

```
<data>
  <data-source>gas_meter3</data-source>
  <data-content>bla bla bla</data-content>
</data>
```

How would you automate this process? This is where AWS lambda could prove handy. Let's look at how to do this.

1 - Creating an S3 bucket


Amazon S3 > Create bucket

Create bucket

Buckets are containers for data stored in S3. Learn more

General configuration

Bucket name

myawsbucket

Bucket name must be unique and must not contain spaces or uppercase letters. See rules for bucket naming

Region

US East (Ohio) us-east-2

Copy settings from existing bucket - *optional*

Only the bucket settings in the following configuration are copied.

Choose bucket

Bucket settings for Block Public Access

Public access is granted to buckets and objects through access control lists (ACLs), bucket policies, access point policies, or all. In order to ensure that public access to this bucket and its objects is blocked, turn on Block all public access. These settings apply only to this bucket and its access points. AWS recommends that you turn on Block all public access, but before applying any of these settings, ensure that your applications will work correctly without public access. If you require some level of public access to this bucket or objects within, you can customize the individual settings below to suit your specific storage use cases. Learn more

✓ Block *all* public access

Turning this setting on is the same as turning on all four settings below. Each of the following settings are independent of one another.

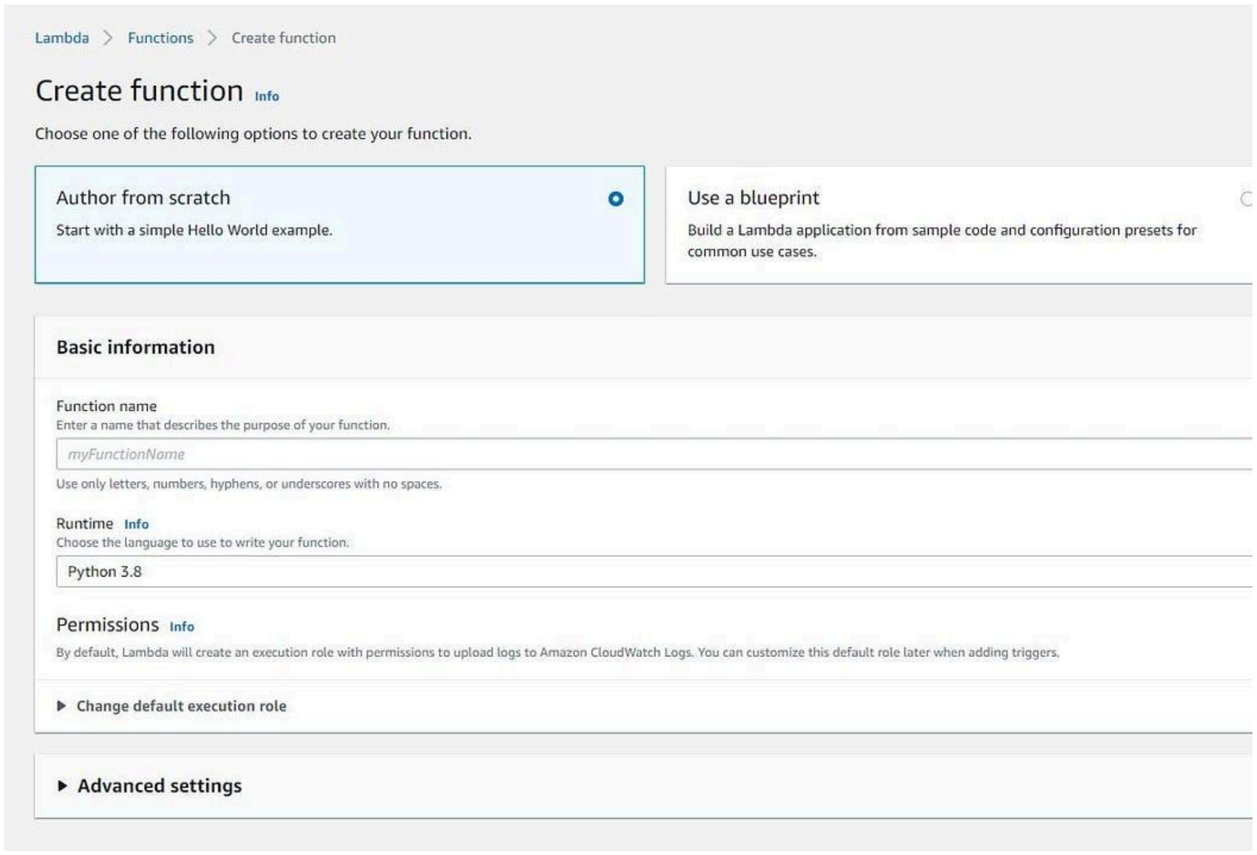
Let's start by building an empty S3 bucket. All you have to do is to go to the S3 page from your AWS console and click on the "Create bucket" button. Make sure you leave the "Block all public access" checkbox ticked and click on "Create bucket".

Now, add a directory called "unsorted" where all the XML files will be stored initially. Create a .xml file named "testdata.xml" with the following content:

```
<data>
  <data-source>gas_meter3</data-source>
  <data-content>bla bla bla</data-content>
</data>
```

2 - Creating a Lambda function

From the Services tab on the AWS console, click on “Lambda”. From the left pane on the Lambda page, select “Functions” and then “Create Functions”.



Create function Info

Choose one of the following options to create your function.

[✓ Author from scratch] Start with a simple Hello World example.

Use a blueprint — Build a Lambda application from sample code and configuration presets for common use cases.

Basic information

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

Runtime Info
Choose the language to use to write your function.
Python 3.8

Permissions Info
By default, Lambda will create an execution role with permissions to upload logs to Amazon CloudWatch Logs. You can customize this default role later when adding triggers.

Select “Author from scratch” and give the function a suitable name. Since I’ll be using Python3, I chose “Python3.8” as the runtime language. There are other versions of Python2 and Python3 available as well. Select a runtime language and click on the “Create function” button. From the list of Lambda functions on the “Functions” page, select the function you just created and you will be taken to the function’s page.

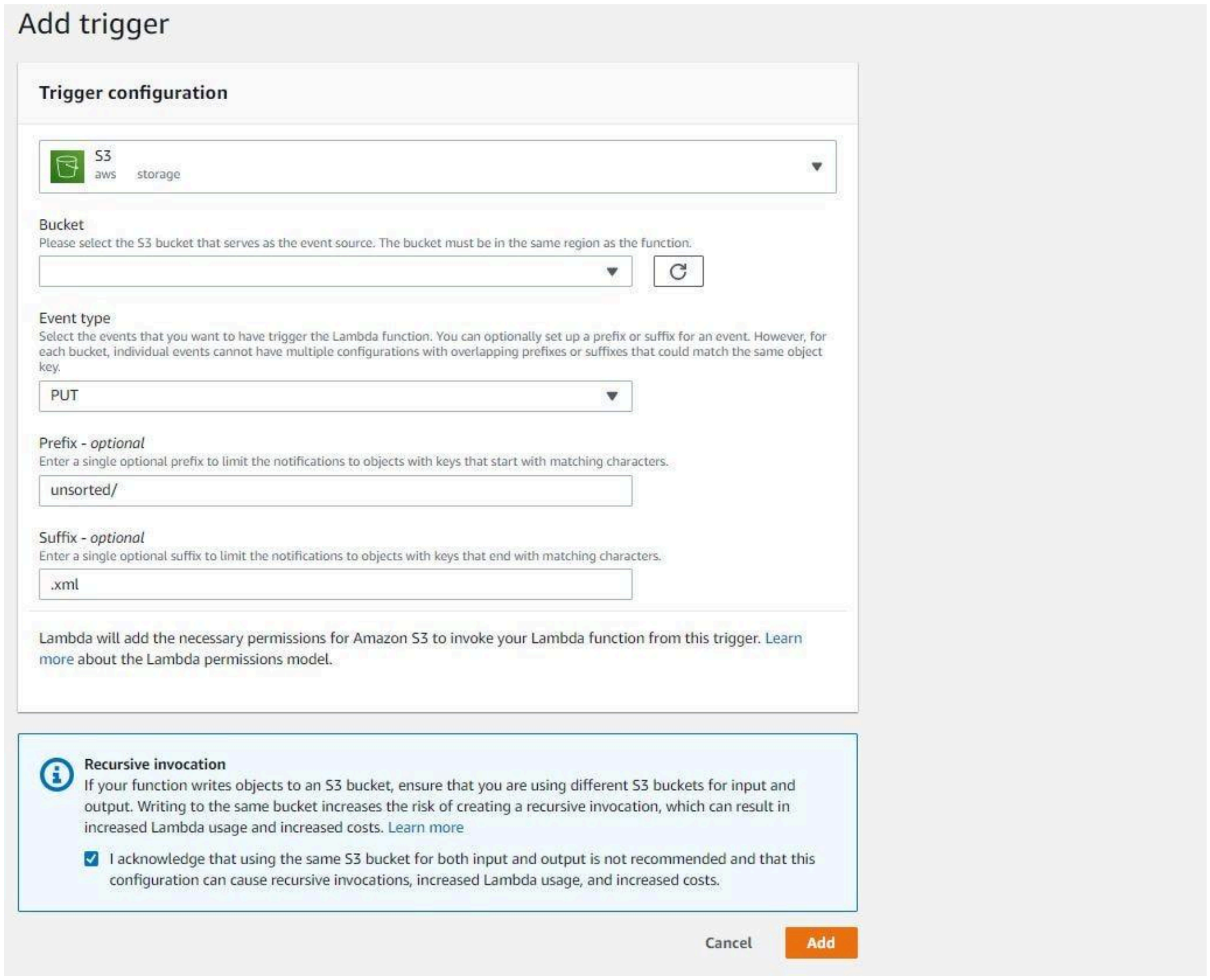
Lambda automatically creates an IAM role for you to use with the Lambda function. The IAM role can be found under the “Permissions” tab on the function’s page. **You need to ensure that the function’s IAM role has permission to access and/or manage the AWS services you connect to from inside your function.**

Make sure you add “S3” permissions to the IAM role’s list of permissions, accessible via the IAM console.

3 - Adding a trigger for our Lambda function

We want the Lambda function to be invoked every time an XML file is uploaded to the “unsorted” folder. To do this, we will use an S3 bucket PUT event as a trigger for our function.

Under the “Designer” section on our Lambda function’s page, click on the “Add trigger” button.


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.

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 - *optional*
Enter a single optional prefix to limit the notifications to objects with keys that start with matching characters.
unsorted/

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

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

Recursive invocation
If your function writes objects to an S3 bucket, ensure that you are using different S3 buckets for input and output. Writing to the same bucket increases the risk of creating a recursive invocation, which can result in increased Lambda usage and increased costs. Learn more

✓ I acknowledge that using the same S3 bucket for both input and output is not recommended and that this configuration can cause recursive invocations, increased Lambda usage, and increased costs.

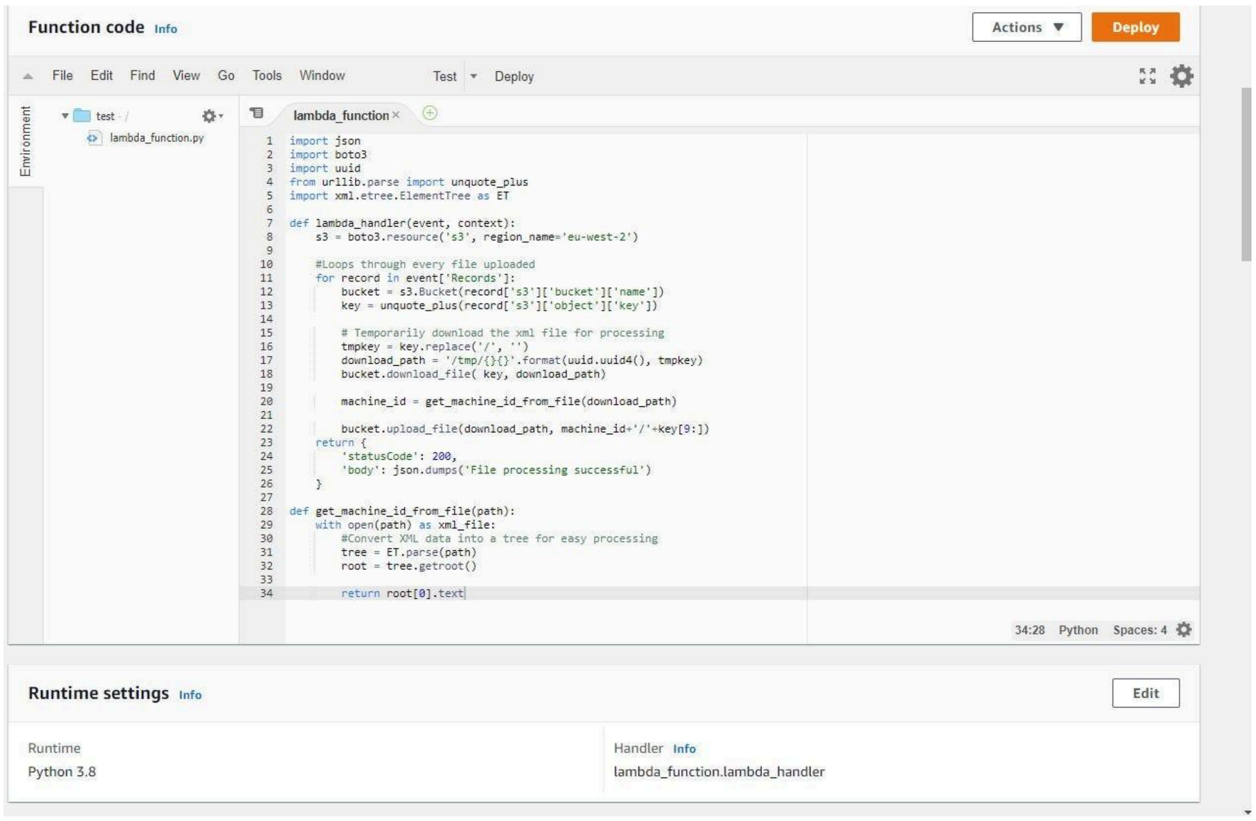
Select the “S3” trigger and the bucket you just created. Select “PUT” event type. Set the prefix and suffix as “unsorted/” and “.xml” respectively. Finally, click on “Add”.

4 - Adding code to our Lambda function

There are 3 ways you can add code to your Lambda function:

1. Through the code editor available on the console.
2. By uploading a .zip file containing all your code and dependencies.
3. By uploading code from an S3 bucket.

We will use the first method for this tutorial. On your function page, go down to the “Function code” section to find the code editor.



```python
import json
import boto3
import uuid
from urllib.parse import unquote_plus
import xml.etree.ElementTree as ET

def lambda_handler(event, context):
    s3 = boto3.resource('s3', region_name='eu-west-2')

    #Loops through every file uploaded
    for record in event['Records']:
        bucket = s3.Bucket(record['s3']['bucket']['name'])
        key = unquote_plus(record['s3']['object']['key'])

        # Temporarily download the xml file for processing
        tmpkey = key.replace('/', '')
        download_path = '/tmp/{}{}'.format(uuid.uuid4(), tmpkey)
        bucket.download_file( key, download_path)

        machine_id = get_machine_id_from_file(download_path)

        bucket.upload_file(download_path, machine_id+'/'+key[9:])
    return {
        'statusCode': 200,
        'body': json.dumps('File processing successful')
    }

def get_machine_id_from_file(path):
    with open(path) as xml_file:
        #Convert XML data into a tree for easy processing
        tree = ET.parse(path)
        root = tree.getroot()

        return root[0].text
```

Runtime
Python 3.8

Handler Info
lambda_function.lambda_handler

Copy and paste the following code into the code editor:

```
import json
import boto3
import uuid
from urllib.parse import unquote_plus
import xml.etree.ElementTree as ET
def lambda_handler(event, context):
    s3 = boto3.resource('s3', region_name='')#Replace with your
region name
    #Loops through every file uploaded
    for record in event['Records']:
        bucket_name = record['s3']['bucket']['name']
        bucket = s3.Bucket(bucket_name)
        key = unquote_plus(record['s3']['object']['key'])
        # Temporarily download the xml file for processing
        tmpkey = key.replace('/', '')
        download_path = '/tmp/{}{}'.format(uuid.uuid4(),
tmpkey)
        bucket.download_file( key, download_path)
        machine_id = get_machine_id_from_file(download_path)
        bucket.upload_file(download_path,
machine_id+'/'+key[9:])

        s3.Object(bucket_name,key).delete()
def get_machine_id_from_file(path):
    tree = ET.parse(path)
    root = tree.getroot()
    return root[0].text
```

Don’t forget to replace the region name.

Make sure the handler value is “<filename>.lambda_handler” . The handler value specifies which function contains the main code that Lambda executes.

Whenever Lambda runs your function, it passes a context object and an event object to it. This object can be used to get information about the function itself and its invocation, eg., function name, memory limit, log group id etc. The context object can be very useful for logging, monitoring and data analytics usages.

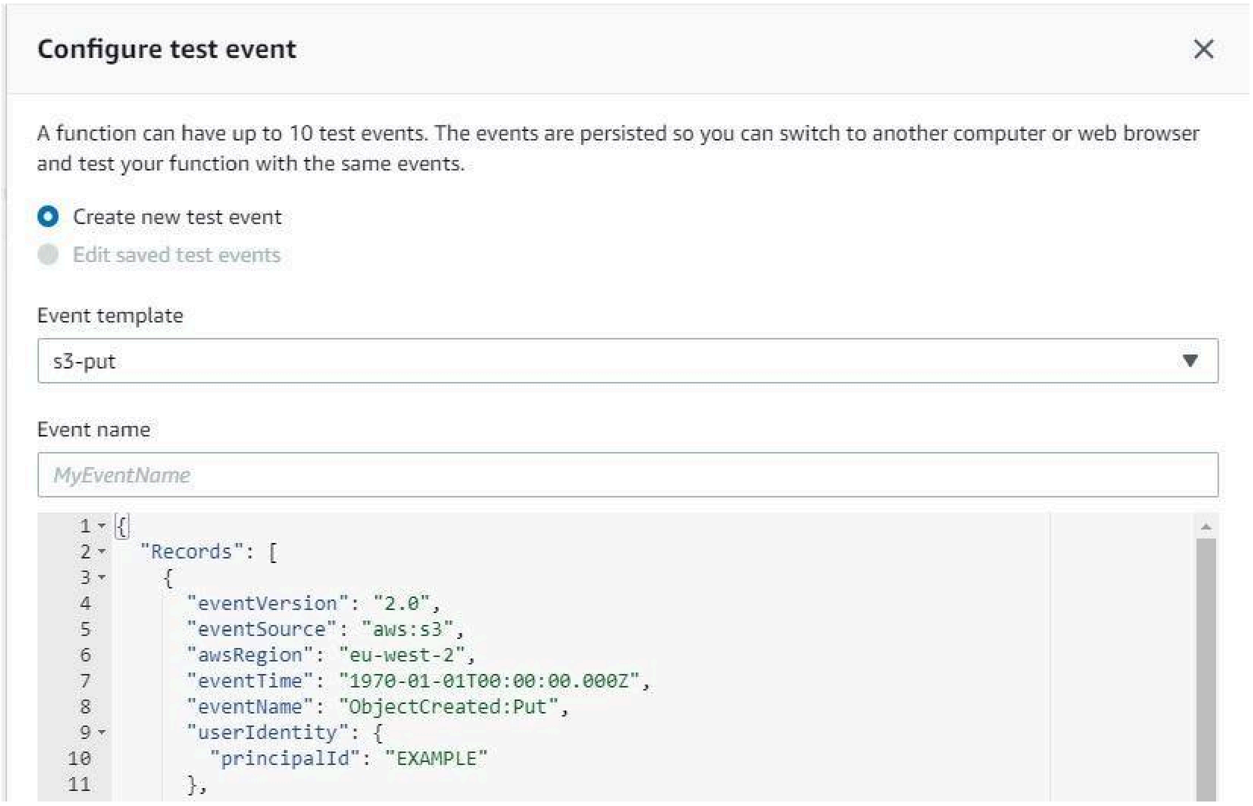
As mentioned earlier the event object is used by Lambda to provide specific information to the Lamda function from the AWS service that invoked the function. The information, which originally comes in JSON format, is converted to an object before being passed into the function. In the case of Python, this object is typically a dictionary. In the code above, you can see that the event object has been used to get the name of the S3 bucket and the key of the object inside the S3 bucket that triggered our function.

Now press the “Deploy” button and our function should be ready to run.

5 - Testing our Lambda function

AWS has made it pretty easy to test Lambda functions via the Lambda console. No matter what trigger your Lambda function uses, you can simulate the invocation of your Lambda function using the **Test** feature on the Lambda console. All this takes is defining what **event** object will be passed into the function. To help you do this, Lambda provides JSON templates specific to each type of trigger.

To test the Lambda function you just created, you need to configure a *test event* for your function. To do this, click on the “Select a test event” dropdown right above the Lambda code editor and click on “Configure test event”.



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: s3-put

Event name: MyEventName

```
{
  "Records": [
    {
      "eventVersion": "2.0",
      "eventSource": "aws:s3",
      "awsRegion": "eu-west-2",
      "eventTime": "1970-01-01T00:00:00.000Z",
      "eventName": "ObjectCreated:Put",
      "userIdentity": {
        "principalId": "EXAMPLE"
      },
```

From the pop-up menu, make sure the “Create new test event” radio button is selected and select the “Amazon S3 Put” event template. You should be provided with JSON data similar to that in the code snippet above. All we are concerned with is the data that is used in our Lambda function, which is the bucket name and the object key. Edit those two values appropriate to the S3 bucket and the XML file you created earlier. Finally, give the test event a name and click on “Create”.

```
{
  "Records": [
    {
      "eventVersion": "2.0",
      "eventSource": "aws:s3",
      "awsRegion": "us-west-2",
      "eventTime": "1970-01-01T00:00:00.000Z",
      "eventName": "ObjectCreated:Put",
      "userIdentity": {
        "principalId": "EXAMPLE"
      },
      "requestParameters": {
        "sourceIPAddress": "126.0.0.1"
      },
      "responseElements": {
        "x-amz-request-id": "EXAMPLE123456789",
        "x-amz-id-2":
"EXAMPLE123/5678abcdefghijklambdaisawesome/mnopqrstuvwxyzABCDEF
GH"
      },
      "s3": {
        "s3SchemaVersion": "1.0",
        "configurationId": "testConfigRule",
        "bucket": {
          "name": "example-bucket",
          "ownerIdentity": {
            "principalId": "EXAMPLE"
          },
          "arn": "arn:aws:s3:::example-bucket"
        },
        "object": {
          "key": "test/key",
          "size": 1024,
          "eTag": "0123456789abcdef0123456789abcdef",
          "sequencer": "0A1B2C3D4E5F678901"
        }
      }
    }
  ]
}
```

Now that you have a test event for your Lambda function, all you have to do is click on the“Test” button on top of the code editor. The console will tell you if the function code was executed without any errors. To check if everything worked go to your S3 bucket to see if the XML file has been moved to a newly created “gas-meter3/” directory.