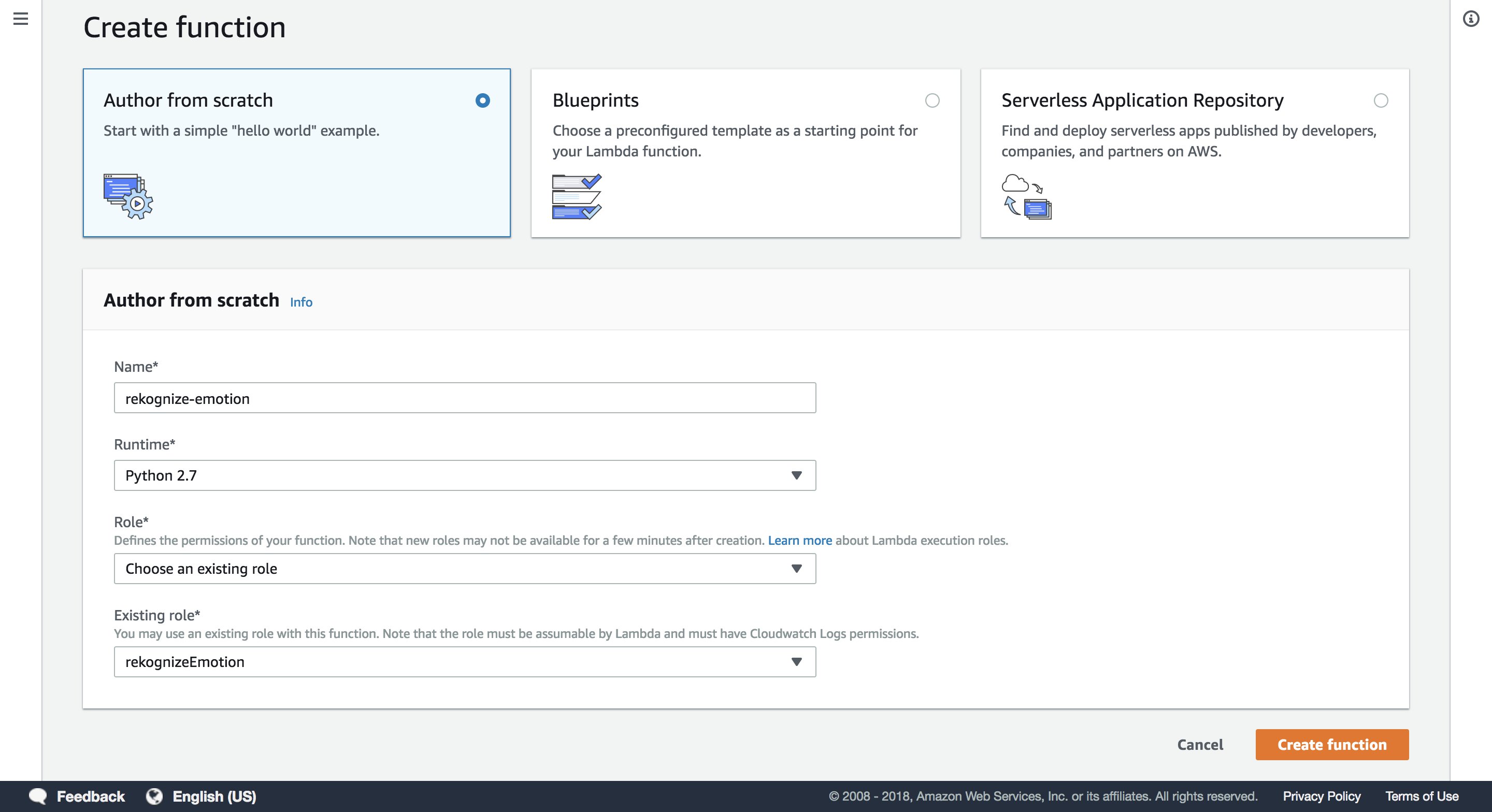
Extend the project by integrating with Amazon Rekognition

In this module, you will learn how to integrate the project with Amazon Rekognition and view the output over CloudWatch.

The inference lambda function that you deployed earlier will upload the cropped faces to your S3. On S3 upload, this new lambda function gets triggered and runs the Rekognize Emotions API by integrating with Amazon Rekognition.

Next, we need to make the Rekognition Lambda function, using the “rekognizeEmotion” role.

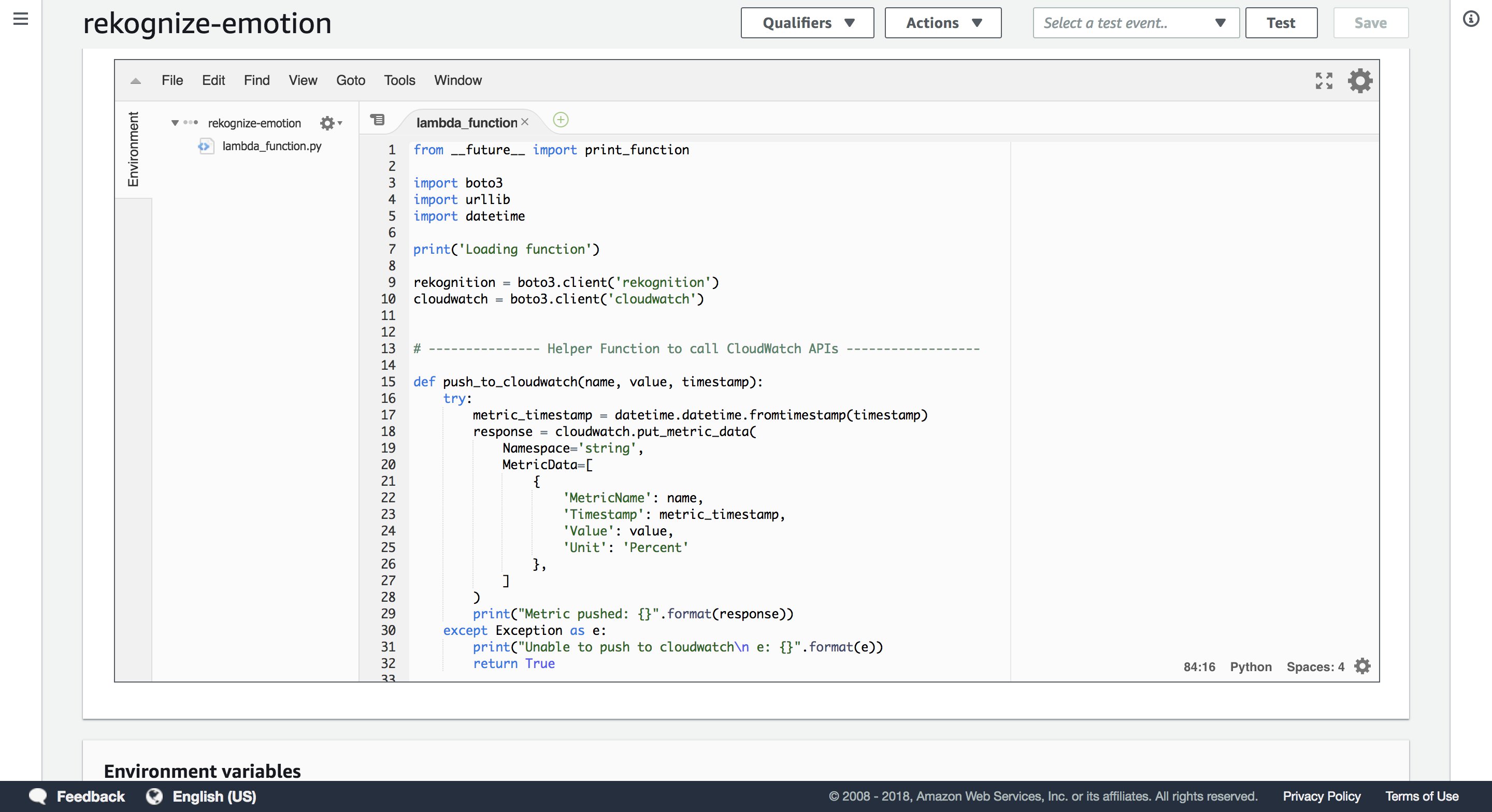


Replace the default script with the following script, which:

* Is triggered upon S3
* Writes metrics to CloudWatch
* Logs metrics to the DynamoDB table created earlier

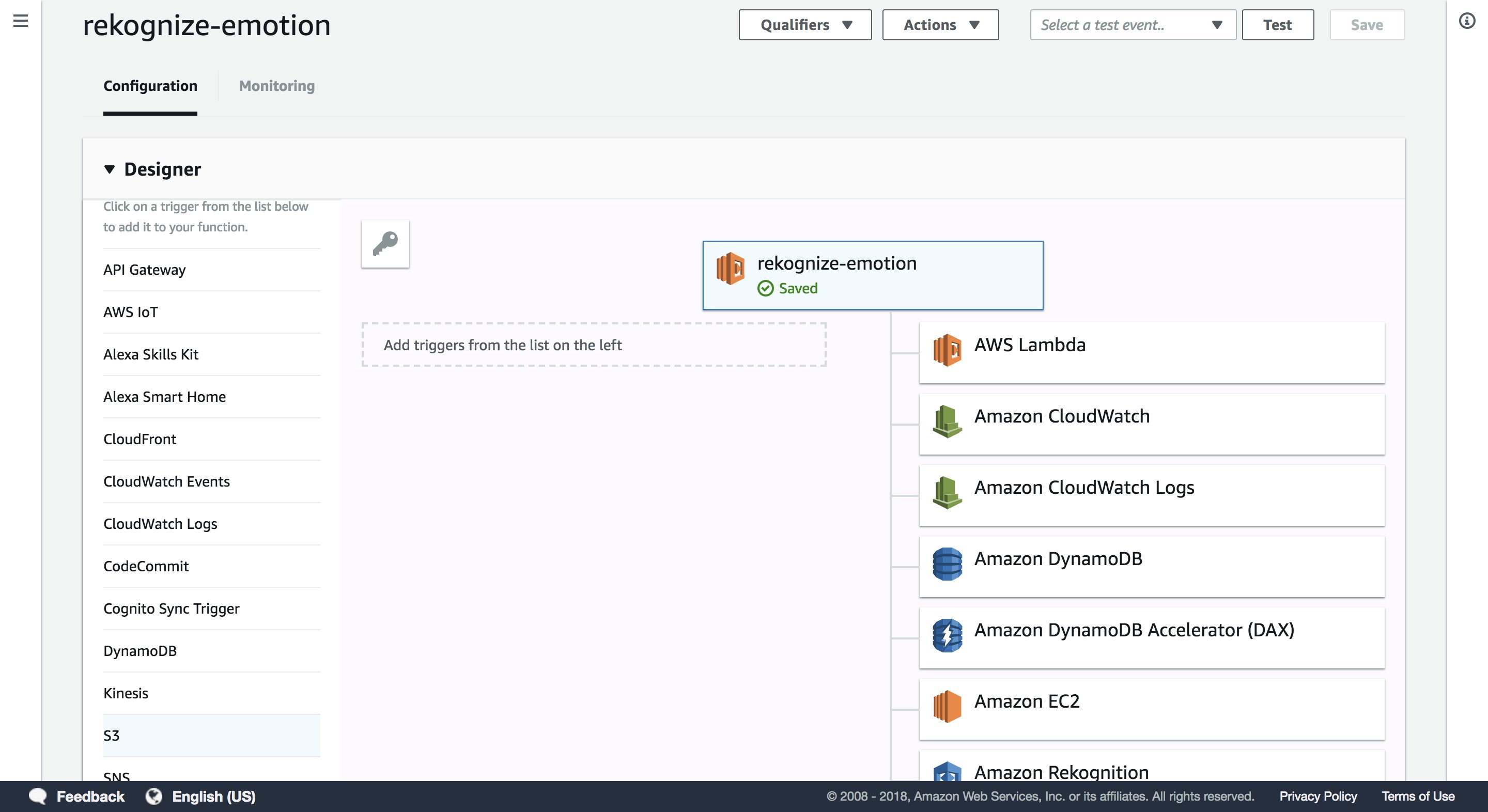
from \_\_future\_\_ import print\_function  
  
import boto3  
import urllib  
import datetime  
  
print('Loading function')  
  
rekognition = boto3.client('rekognition')  
cloudwatch = boto3.client('cloudwatch')  
  
  
# --------------- Helper Function to call CloudWatch APIs ------------------  
  
def push\_to\_cloudwatch(name, value, timestamp):  
    try:  
        metric\_timestamp = datetime.datetime.fromtimestamp(timestamp)  
        response = cloudwatch.put\_metric\_data(  
            Namespace='string',  
            MetricData=[  
                {  
                    'MetricName': name,  
                    'Timestamp': metric\_timestamp,  
                    'Value': value,  
                    'Unit': 'Percent'  
                },  
            ]  
        )  
        print("Metric pushed: {}".format(response))  
    except Exception as e:  
        print("Unable to push to cloudwatch\n e: {}".format(e))  
        return True  
  
# --------------- Helper Functions to call Rekognition APIs ------------------  
  
def detect\_faces(bucket, key):  
    print("Key: {}".format(key))  
    response = rekognition.detect\_faces(Image={"S3Object":  
                                               {"Bucket": bucket,  
                                                "Name": key}},  
                                        Attributes=['ALL'])  
  
    if not response['FaceDetails']:  
        print ("No Face Details Found!")  
        return response  
  
    push = False  
    timestamp = float(key.split('\_')[2].split('/')[1])  
    dynamo\_obj = {}  
    dynamo\_obj['s3key'] = key  
  
    for index, item in enumerate(response['FaceDetails'][0]['Emotions']):  
        print("Item: {}".format(item))  
        if int(item['Confidence']) > 10:  
            push = True  
            dynamo\_obj[item['Type']] = str(round(item["Confidence"], 2))  
            push\_to\_cloudwatch(item['Type'], round(item["Confidence"], 2), timestamp)  
  
    if push:  # Push only if at least on emotion was found  
        table = boto3.resource('dynamodb').Table('rekognize-faces')  
        table.put\_item(Item=dynamo\_obj)  
  
    return response  
  
# --------------- Main handler ------------------  
  
  
def lambda\_handler(event, context):  
    '''Demonstrates S3 trigger that uses  
    Rekognition APIs to detect faces, labels and index faces in S3 Object.  
    '''  
  
    # Get the object from the event  
    bucket = event['Records'][0]['s3']['bucket']['name']  
    key = urllib.unquote\_plus(event['Records'][0]['s3']['object']['key'].encode('utf8'))  
    try:  
        # Calls rekognition DetectFaces API to detect faces in S3 object  
        response = detect\_faces(bucket, key)  
  
        return response  
    except Exception as e:  
        print("Error processing object {} from bucket {}. ".format(key, bucket) +  
              "Make sure your object and bucket exist and your bucket is in the same region as this function.")  
        raise e

Once the scrip is inserted:

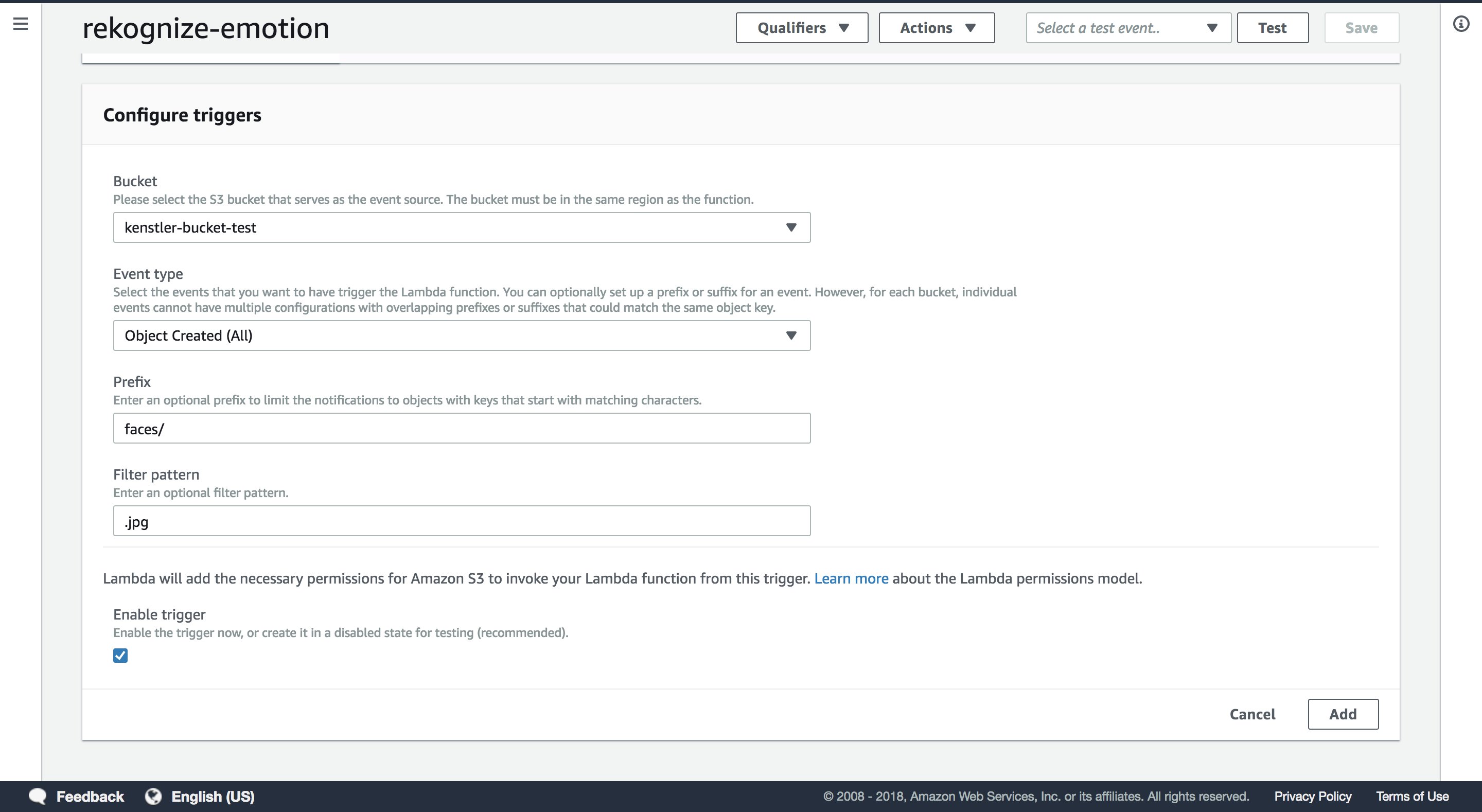


We need to add the event that triggers this lambda function. This will be an “S3:ObjectCreated” event that happens every time a face is uploaded to the face S3 bucket

Add the “S3” trigger:



With the following configuration:



replacing “kenstler-bucket-test” with your own bucket from earlier.

Once you've added the trigger, save the lambda function.

**Publish the lambda function.**