Create a DeepLens Inference Lambda function

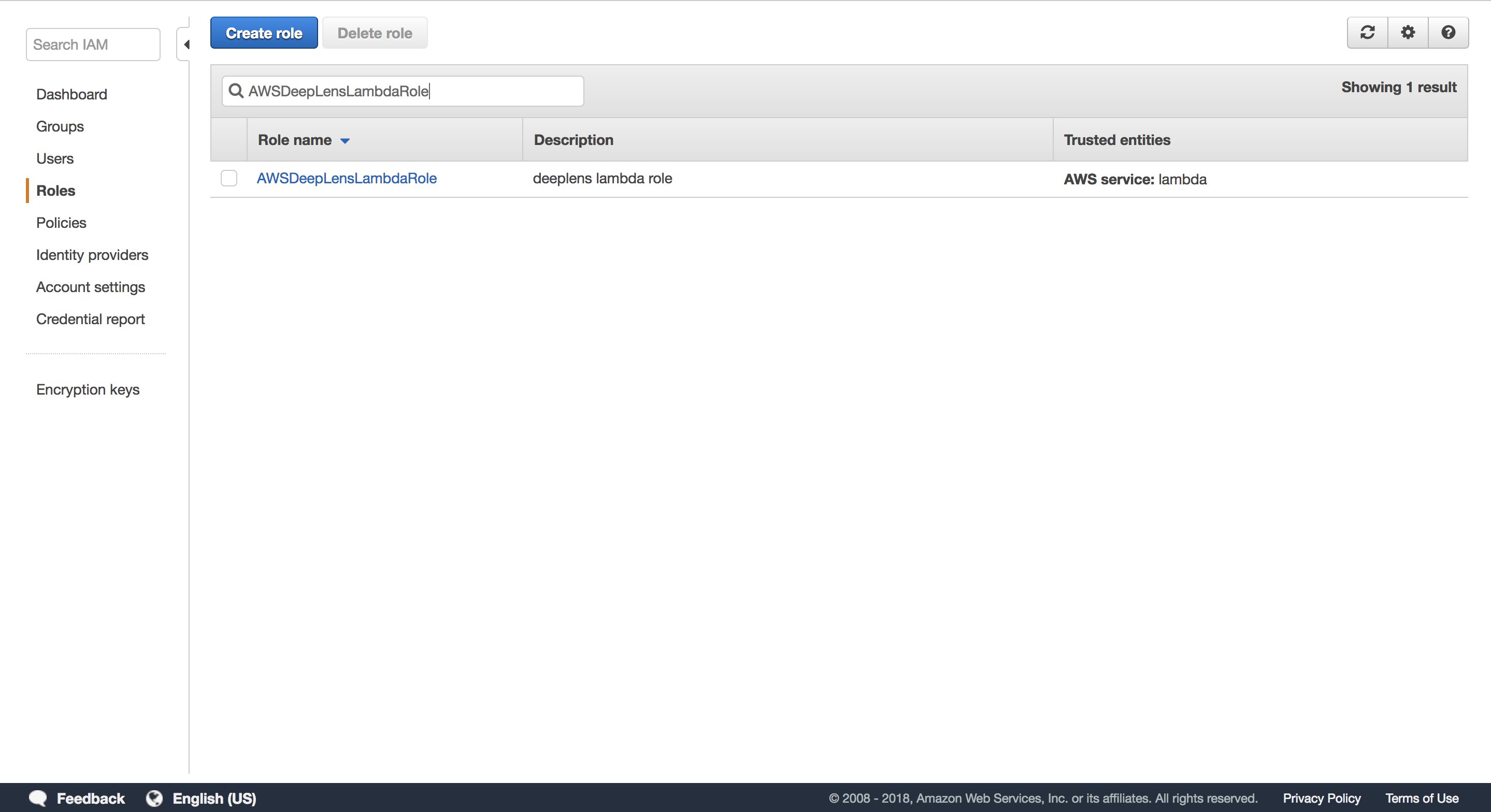
In this module, you will learn to create an inference lambda function for your DeepLens. This lambda function will crop the identified faces and upload them to your S3 bucket.

### IAM

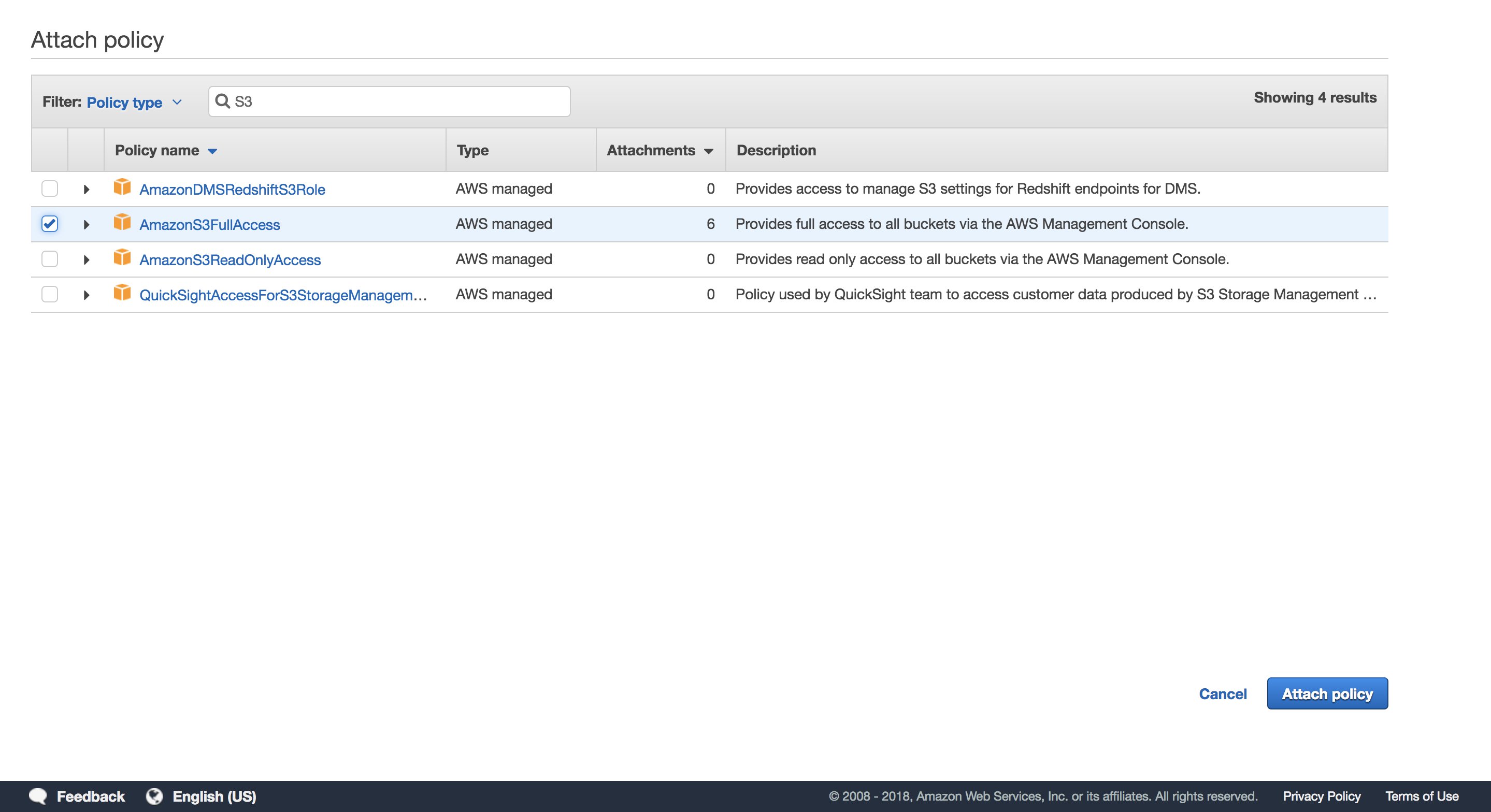
Participants need to first set permissions for the roles the Lambda functions will be using.

First, we need to add S3 permissions to the DeepLens Lambda role so the lambda on the device can call Put Object into the bucket of interest.

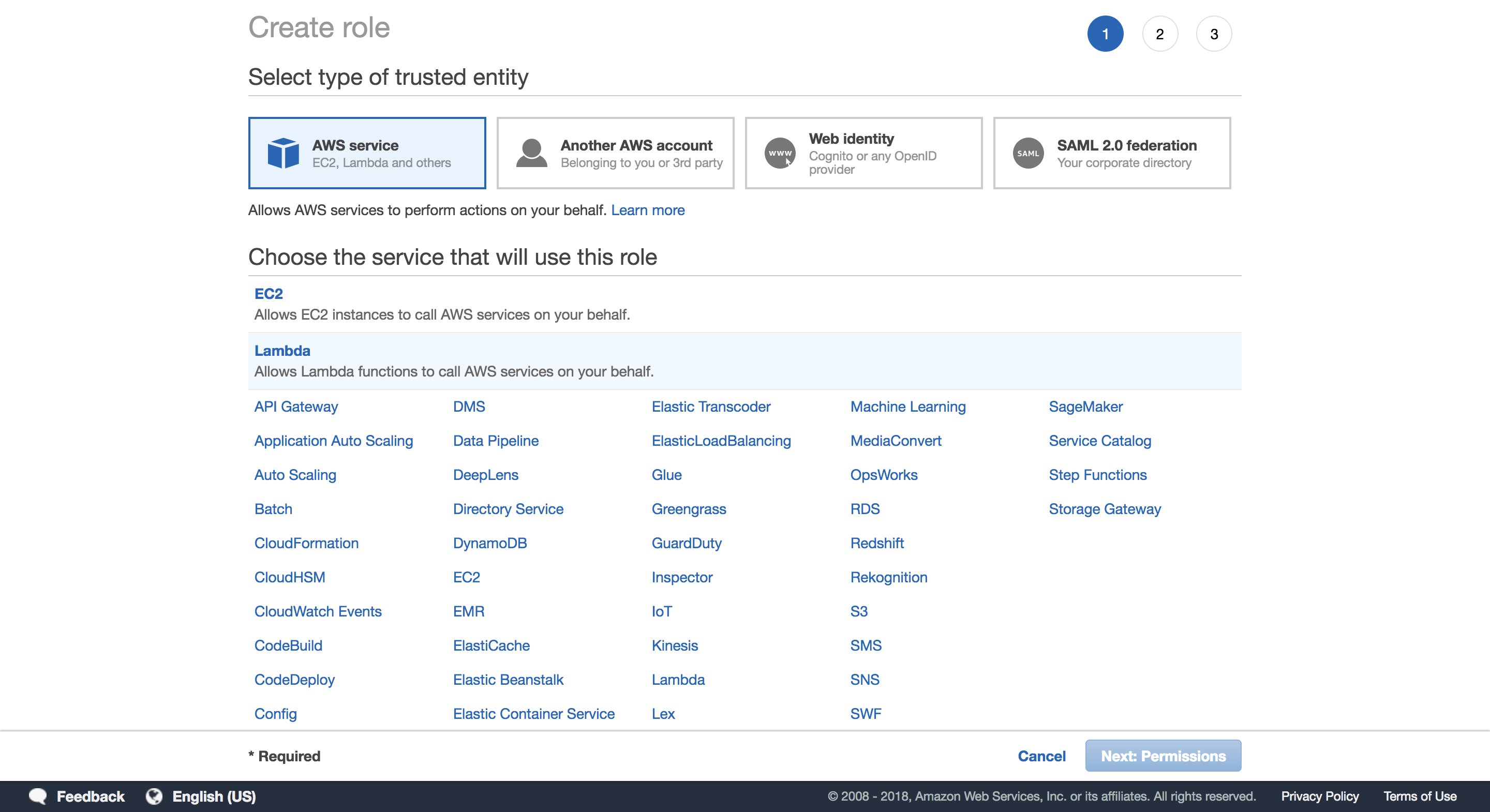
Go to the “Roles” page on the IAM console. Look up the role “AWSDeepLensLambdaRole”



Click on the Role, Click Attach Policy, and attach “AmazonS3FullAccess”



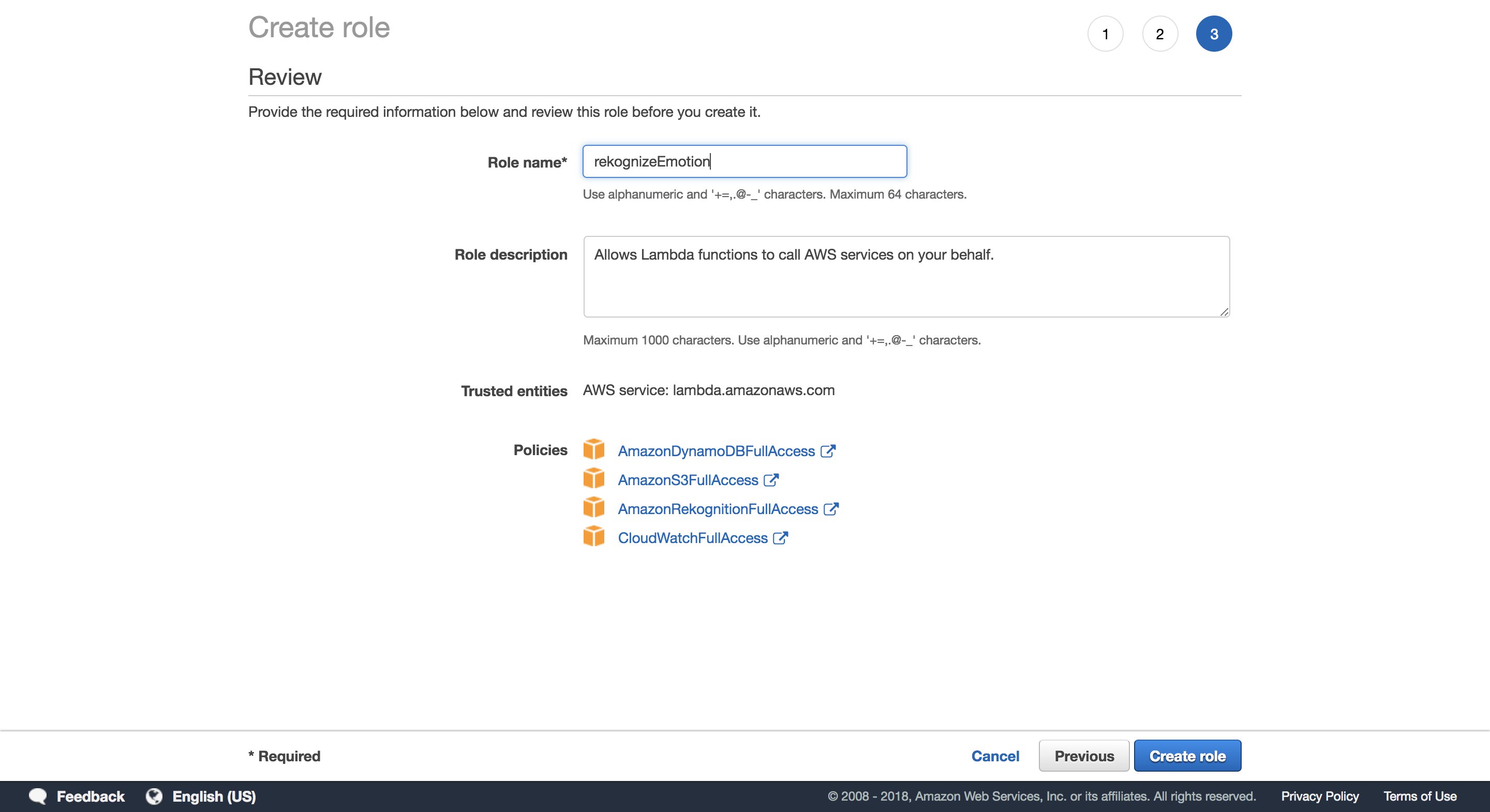
Now we have to create a role for Rekognition Lambda. From “Create Role”, Select “AWS Service” and select “Lambda”:



Attach the following policies:

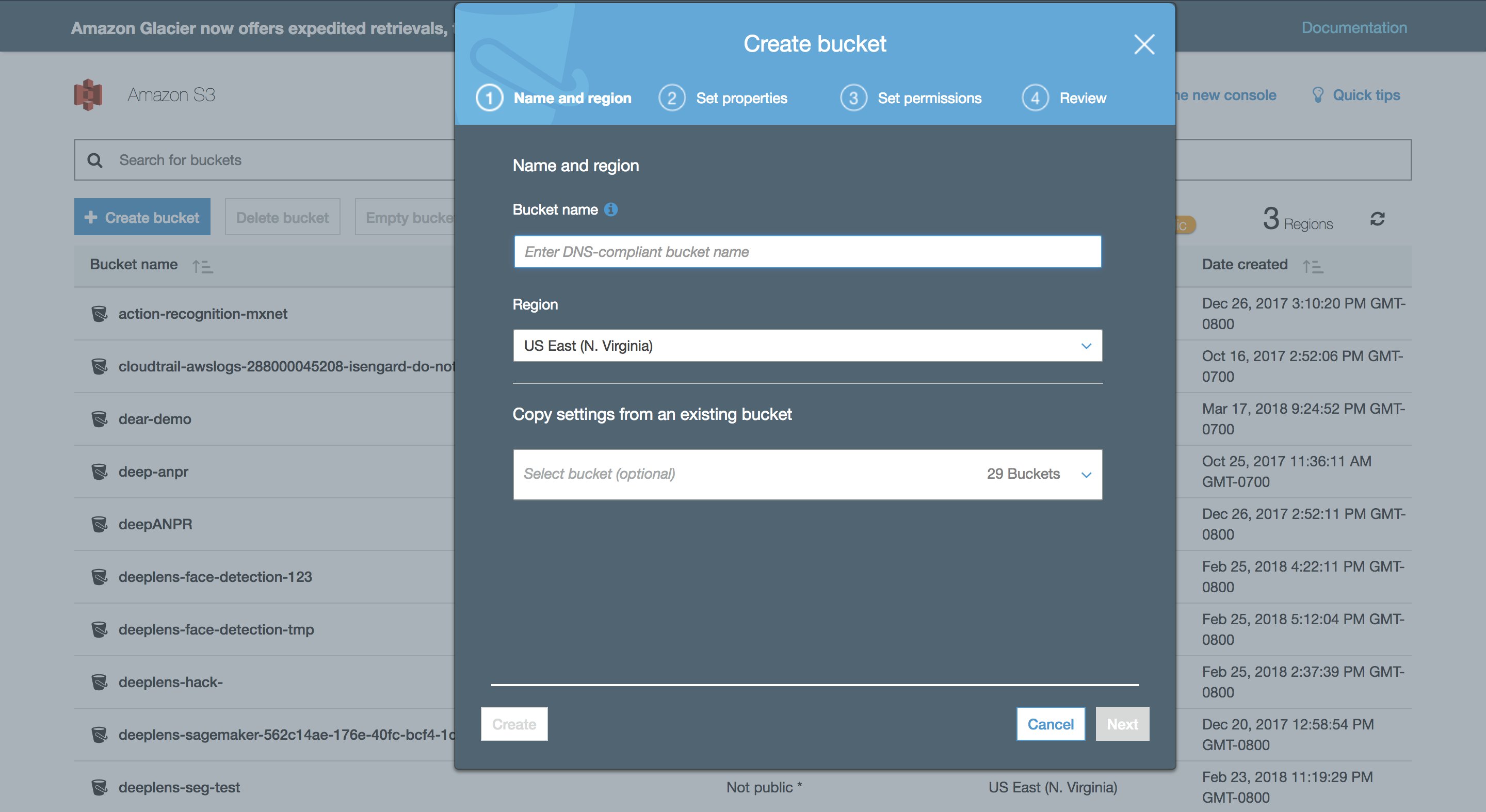
* AmazonDynamoDBFullAcces
* AmazonS3FullAccess
* AmazonRekognitionFullAccess
* CloudWatchFullAccess

And create the role “rekognizeEmotion”:



### S3

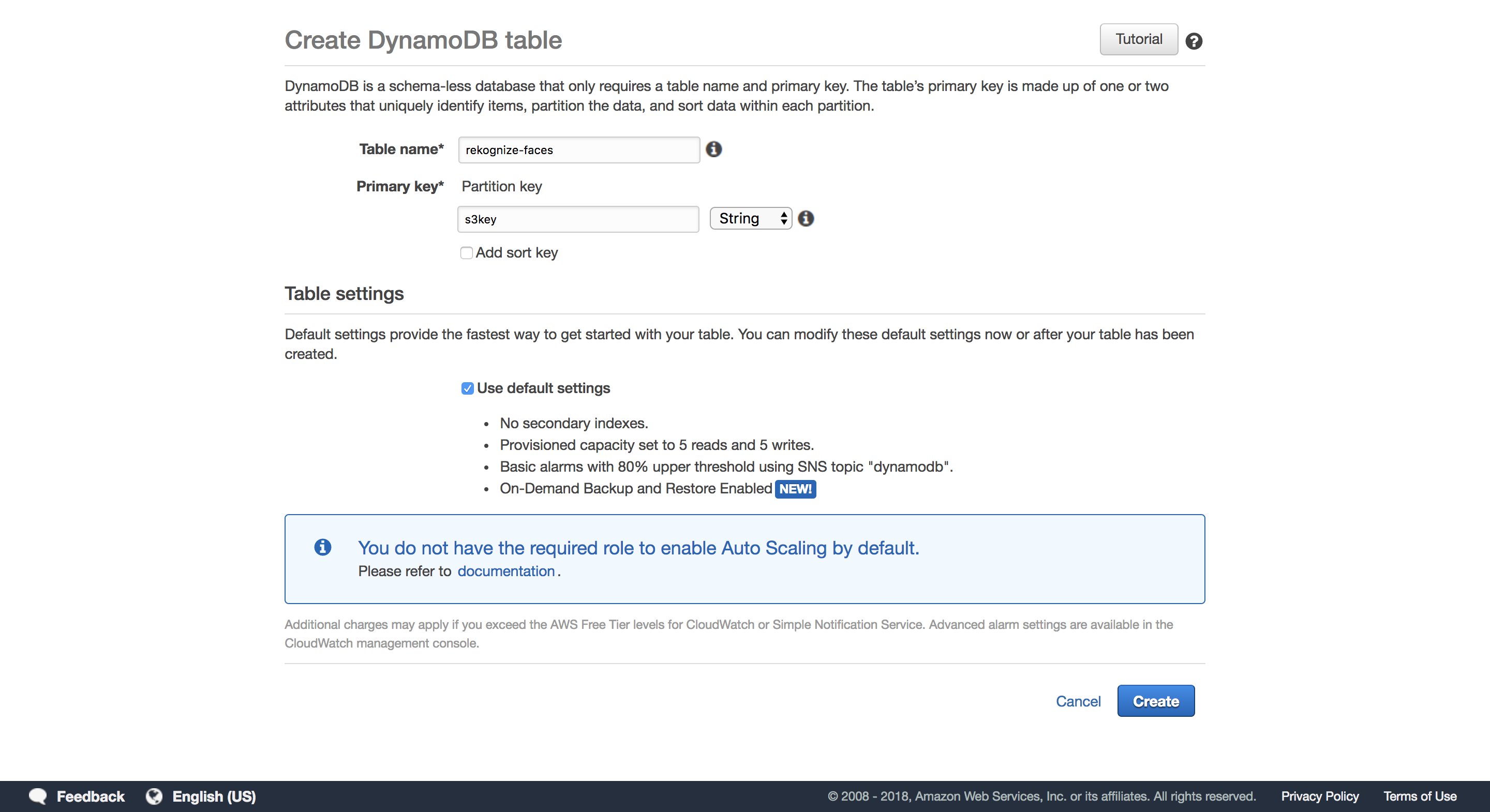
We need to create an S3 bucket that we can upload faces to. These uploads will trigger our Rekognition lambda.



Note the name of the bucket, we'll use it in later steps.

### DYNAModb

Next, we need to create the DynamoDB table that will store our output:



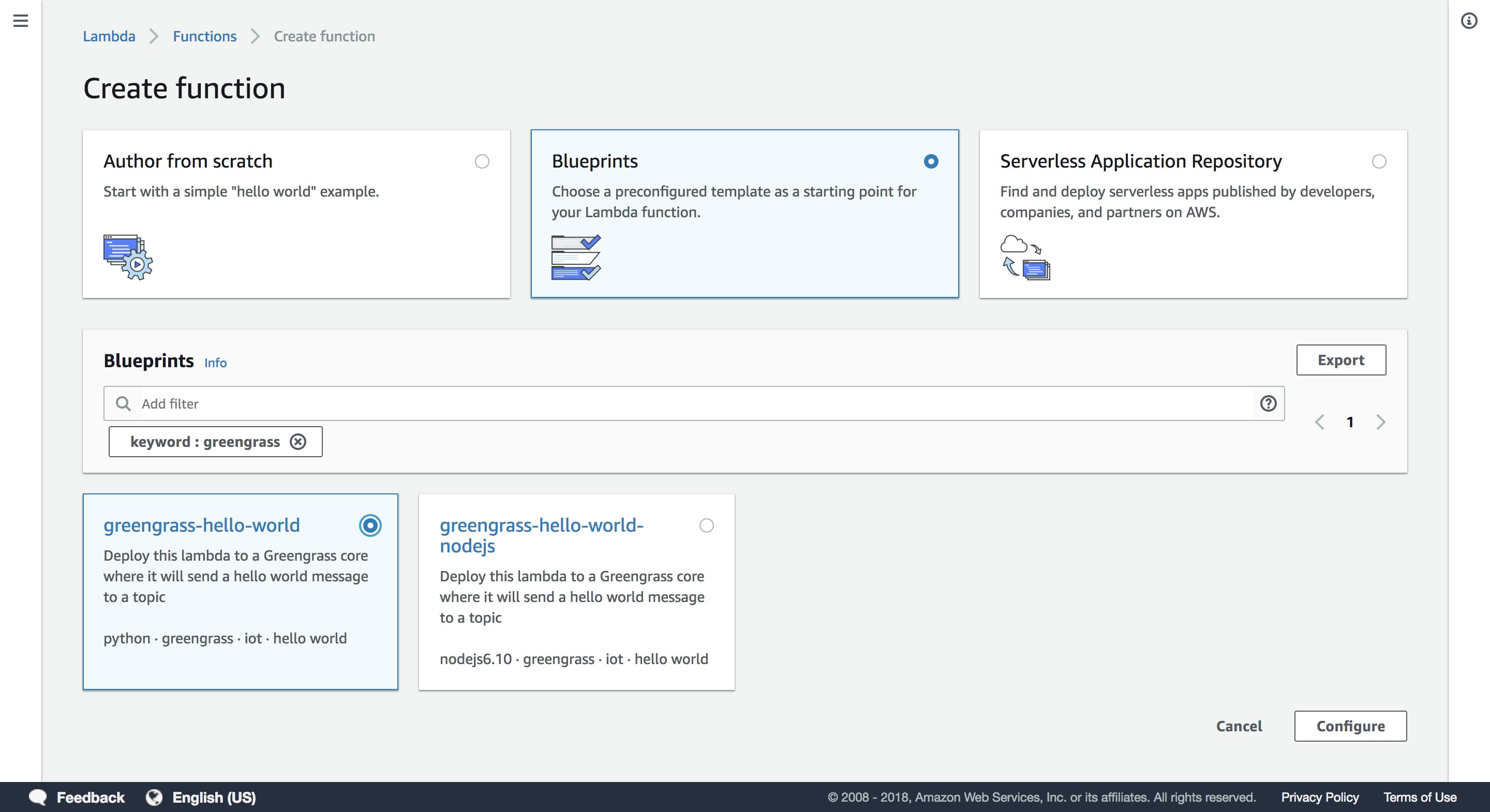
### Lambda Functions

Next, we need to make the two lambda functions:

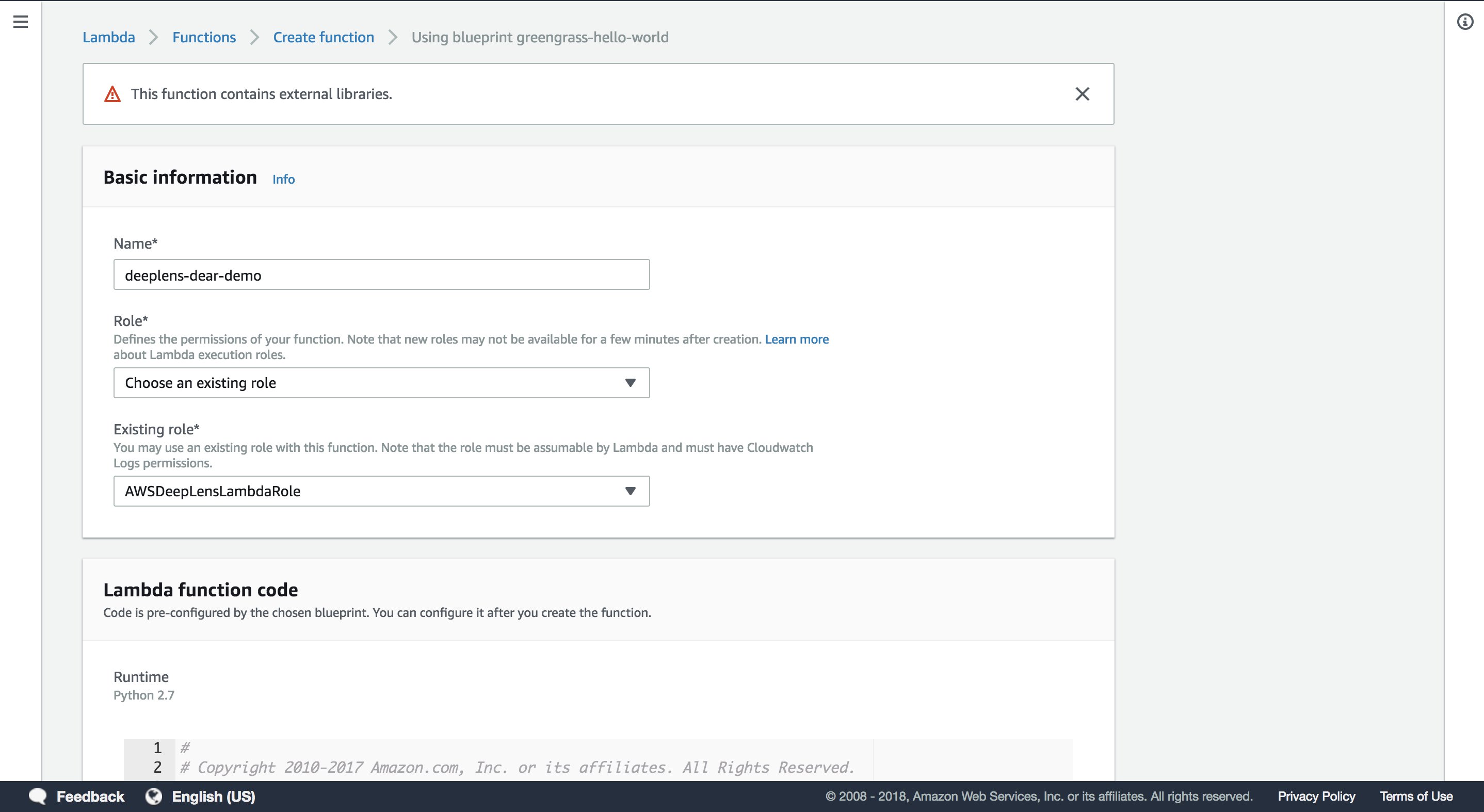
* The greengrass lambda function that runs on the device
* The cloud lambda that sends faces to Rekognition

First, we'll create the greengrass Lambda that will run on the device:

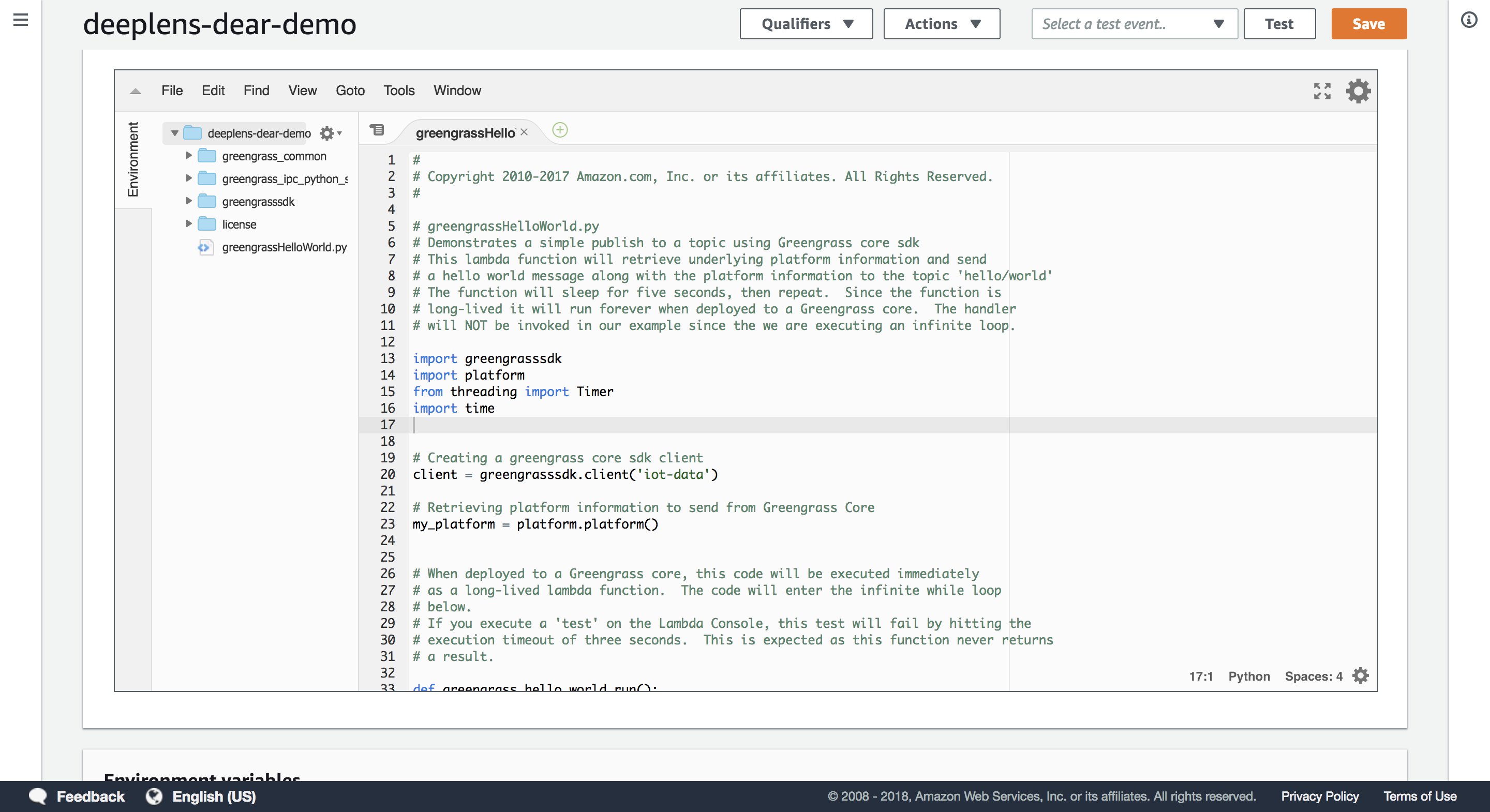
Start by creating a function from the “greengrass-hello-world” blueprint:



Name the function “deeplens-dear-demo”, and attach the AWSDeepLensLambdaRole:



Once created, we're going to replace the default handler code:



with the inference script below, inserting real values for the following fields:

* <BUCKET\_NAME>: the S3 bucket faces will be written to, which we created above
* <ACCESS\_KEY\_ID>: Your AWS access key ID credential
* <SECRET\_ACCESS\_KEY>: Your AWS secret access key credential

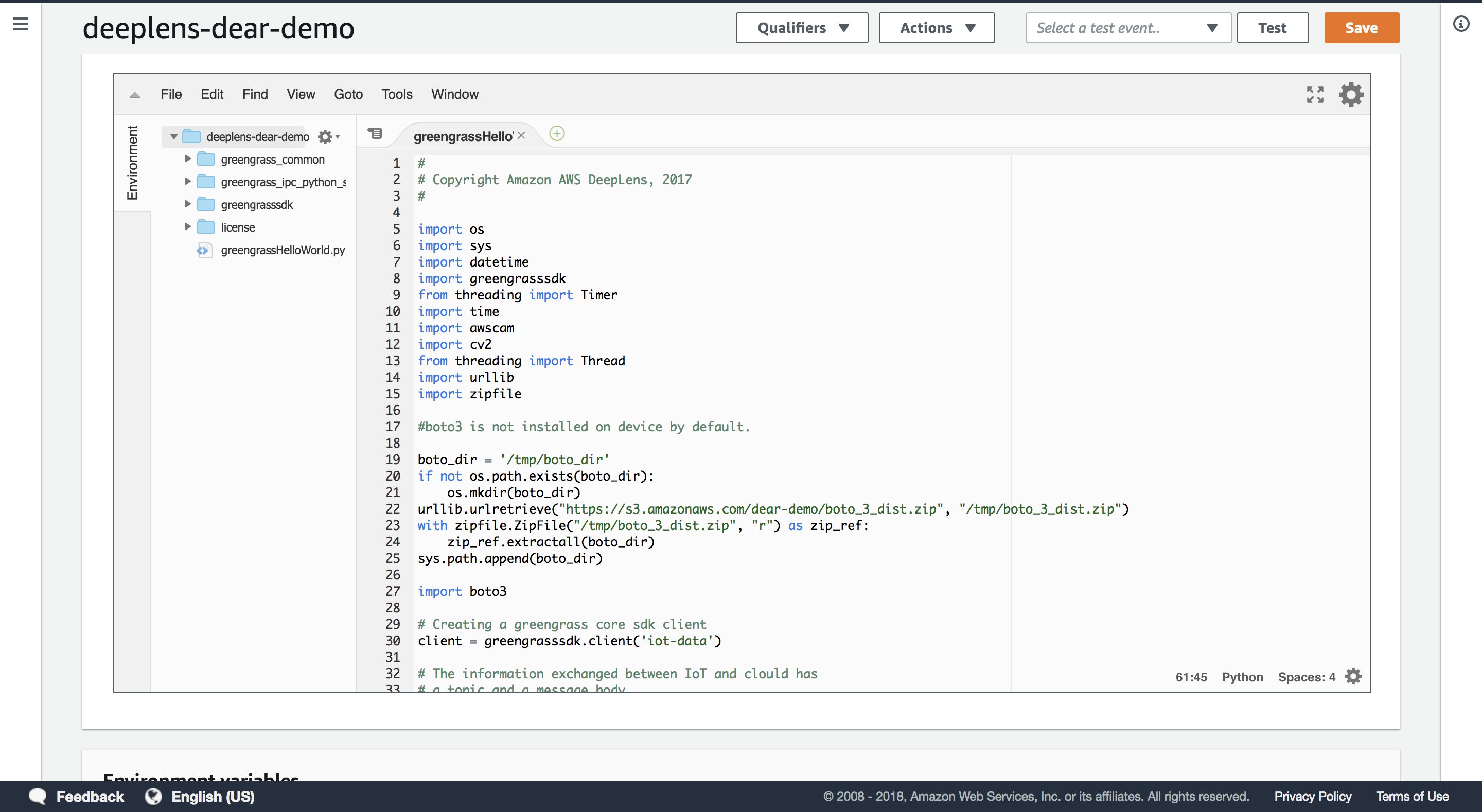
**NOTE:** This approach hard-codes credentials into the lambda script. Boto3 requires credentials to make calls, and it inherits these credentials from AWS CLI config on-device. Since this setup is intended to occur completely in the console, the AWS CLI cannot be configured on-device and so we've hardcoded it. If there is sufficient technical bandwidth to facilitate such configuration, this hard-coding is unnecessary.

In order to get your **Access Key ID** and **Secret Access Key** follow next steps:

1. Open the [IAM console](https://console.aws.amazon.com/iam/home?#home).
2. From the navigation menu, click **Users**.
3. Select your IAM user name.
4. Click **User Actions**, and then click **Manage Access Keys**.
5. Click **Create Access Key**.
6. Your keys will look something like this:
   * Access key ID example: AKIAIOSFODNN7EXAMPLE
   * Secret access key example: wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
7. Click **Download Credentials**, and store the keys in a secure location.

#  
# Copyright Amazon AWS DeepLens, 2017  
#  
  
import os  
import sys  
import datetime  
import greengrasssdk  
from threading import Timer  
import time  
import awscam  
import cv2  
from threading import Thread  
import urllib  
import zipfile  
  
#boto3 is not installed on device by default.  
  
boto\_dir = '/tmp/boto\_dir'  
if not os.path.exists(boto\_dir):  
    os.mkdir(boto\_dir)  
urllib.urlretrieve("https://s3.amazonaws.com/dear-demo/boto\_3\_dist.zip", "/tmp/boto\_3\_dist.zip")  
with zipfile.ZipFile("/tmp/boto\_3\_dist.zip", "r") as zip\_ref:  
    zip\_ref.extractall(boto\_dir)  
sys.path.append(boto\_dir)  
  
import boto3  
  
# Creating a greengrass core sdk client  
client = greengrasssdk.client('iot-data')  
  
# The information exchanged between IoT and clould has  
# a topic and a message body.  
# This is the topic that this code uses to send messages to cloud  
iotTopic = '$aws/things/{}/infer'.format(os.environ['AWS\_IOT\_THING\_NAME'])  
  
ret, frame = awscam.getLastFrame()  
ret, jpeg = cv2.imencode('.jpg', frame)  
  
Write\_To\_FIFO = True  
  
class FIFO\_Thread(Thread):  
    def \_\_init\_\_(self):  
        ''' Constructor. '''  
        Thread.\_\_init\_\_(self)  
  
    def run(self):  
        fifo\_path = "/tmp/results.mjpeg"  
        if not os.path.exists(fifo\_path):  
            os.mkfifo(fifo\_path)  
        f = open(fifo\_path, 'w')  
        client.publish(topic=iotTopic, payload="Opened Pipe")  
        while Write\_To\_FIFO:  
            try:  
                f.write(jpeg.tobytes())  
            except IOError as e:  
                continue  
  
def push\_to\_s3(img, index):  
    try:  
        bucket\_name = "<BUCKET\_NAME>"  
  
        timestamp = int(time.time())  
        now = datetime.datetime.now()  
        key = "faces/{}\_{}/{}\_{}/{}\_{}.jpg".format(now.month, now.day,  
                                                   now.hour, now.minute,  
                                                   timestamp, index)  
  
        s3 = boto3.client('s3',  
            aws\_access\_key\_id='<ACCESS\_KEY\_ID>',  
            aws\_secret\_access\_key='<SECRET\_ACCESS\_KEY>'  
            )  
  
        encode\_param = [int(cv2.IMWRITE\_JPEG\_QUALITY), 90]  
        \_, jpg\_data = cv2.imencode('.jpg', img, encode\_param)  
        response = s3.put\_object(ACL='public-read',  
                                 Body=jpg\_data.tostring(),  
                                 Bucket=bucket\_name,  
                                 Key=key)  
  
        client.publish(topic=iotTopic, payload="Response: {}".format(response))  
        client.publish(topic=iotTopic, payload="Face pushed to S3")  
    except Exception as e:  
        msg = "Pushing to S3 failed: " + str(e)  
        client.publish(topic=iotTopic, payload=msg)  
  
def greengrass\_infinite\_infer\_run():  
    try:  
        modelPath = "/opt/awscam/artifacts/mxnet\_deploy\_ssd\_FP16\_FUSED.xml"  
        modelType = "ssd"  
        input\_width = 300  
        input\_height = 300  
        prob\_thresh = 0.25  
        results\_thread = FIFO\_Thread()  
        results\_thread.start()  
  
        # Send a starting message to IoT console  
        client.publish(topic=iotTopic, payload="Face detection starts now")  
  
        # Load model to GPU (use {"GPU": 0} for CPU)  
        mcfg = {"GPU": 1}  
        model = awscam.Model(modelPath, mcfg)  
        client.publish(topic=iotTopic, payload="Model loaded")  
        ret, frame = awscam.getLastFrame()  
        if ret == False:  
            raise Exception("Failed to get frame from the stream")  
  
        yscale = float(frame.shape[0]/input\_height)  
        xscale = float(frame.shape[1]/input\_width)  
  
        doInfer = True  
        while doInfer:  
            # Get a frame from the video stream  
            ret, frame = awscam.getLastFrame()  
            # Raise an exception if failing to get a frame  
            if ret == False:  
                raise Exception("Failed to get frame from the stream")  
  
            # Resize frame to fit model input requirement  
            frameResize = cv2.resize(frame, (input\_width, input\_height))  
  
            # Run model inference on the resized frame  
            inferOutput = model.doInference(frameResize)  
  
            # Output inference result to the fifo file so it can be viewed with mplayer  
            parsed\_results = model.parseResult(modelType, inferOutput)['ssd']  
            # client.publish(topic=iotTopic, payload = json.dumps(parsed\_results))  
            label = '{'  
            for i, obj in enumerate(parsed\_results):  
                if obj['prob'] < prob\_thresh:  
                    break  
                offset = 25  
                xmin = int( xscale \* obj['xmin'] ) + int((obj['xmin'] - input\_width/2) + input\_width/2)  
                ymin = int( yscale \* obj['ymin'] )  
                xmax = int( xscale \* obj['xmax'] ) + int((obj['xmax'] - input\_width/2) + input\_width/2)  
                ymax = int( yscale \* obj['ymax'] )  
  
                crop\_img = frame[ymin:ymax, xmin:xmax]  
  
                push\_to\_s3(crop\_img, i)  
  
                cv2.rectangle(frame, (xmin, ymin), (xmax, ymax), (255, 165, 20), 4)  
                label += '"{}": {:.2f},'.format(str(obj['label']), obj['prob'] )  
                label\_show = '{}: {:.2f}'.format(str(obj['label']), obj['prob'] )  
                cv2.putText(frame, label\_show, (xmin, ymin-15),  
                            cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (255, 165, 20), 4)  
            label += '"null": 0.0'  
            label += '}'  
            client.publish(topic=iotTopic, payload=label)  
            global jpeg  
            ret, jpeg = cv2.imencode('.jpg', frame)  
  
    except Exception as e:  
        msg = "Test failed: " + str(e)  
        client.publish(topic=iotTopic, payload=msg)  
  
    # Asynchronously schedule this function to be run again in 15 seconds  
    Timer(15, greengrass\_infinite\_infer\_run).start()  
  
  
# Execute the function above  
greengrass\_infinite\_infer\_run()  
  
  
# This is a dummy handler and will not be invoked  
# Instead the code above will be executed in an infinite loop for our example  
def function\_handler(event, context):  
    return

Once replaced:



**Click “Save”, and then under the “Actions” drop-down menu,**

**click “Publish new version” and publish.**