**Exploring AWS Rekognition and Lambda**

We have so far built a simple web application where you can upload photos that will be stored in an Amazon S3 bucket. We will extend this application so that the photos will be processed with Amazon Rekognition to generate labels. We will continue using Amazon Cloud9 as an IDE and for testing the application.

We will now modify our application in a way that the photos will be processed with Amazon Rekognition asynchronously using AWS lambda.

**Objective:** (Session-1)

- To explore rekognition service of AWS as part of the project
- To learn serverless computing architecture i.e AWS lambda

**General Instructions:**

- Each lab is to be done individually. However you can discuss with others, but effort should finally be yours.

**References:**

- https://aws.amazon.com/rekognition/
- https://aws.amazon.com/lambda/features/?pg=ln&sec=hs

## 0. Explore the exercise code.

You do not have to write any code for this exercise, we will use some existing code for this exercise. However, it is good if you go through the code and understand it at least at some high level.

- In the AWS Cloud9 terminal, ensure you are in the ~/environment directory else, do "`cd ~/environment`"

- `You should now see the folder` **FlaskApp** in your labDirectory. You need to create same directory structure as FlaskApp in your cloud9 terminal and also copy the file contents of FlaskApp directory(in labDirectory) and paste them into the files present in cloud9

## 1. Create a new S3 Bucket for storing Labels as created in Version 1

Navigate to the S3 service and create a new bucket. For example, name it `labels-gallery-100`.

Note down the bucket name as it will be used in your Lambda function.

## 2. Create an AWS IAM role for the AWS Lambda function.

In this section, you will create an AWS IAM role to authenticate the AWS Lambda function to talk to Amazon Rekognition and Amazon S3.

**Trusted entity: Lambda**

**Permissions: AWSLambdaVPCAccessExecutionRole, AmazonRekognitionReadOnlyAccess, AmazonS3ReadOnlyAccess**

**Role name: labels-lambda-role**

Here are the steps to create an AWS IAM role for your AWS Lambda function:

1. Open the AWS Management Console and navigate to the IAM service.
2. In the navigation pane, click on “Roles”, then click on “Create role”.

3. In the “Create role” section, select “AWS service” as the type of trusted entity.
4. In the “Choose the service that will use this role” section, select “Lambda”.
5. Click on “Next: Permissions”.
6. In the “Attach permissions policies” section, search for and select the following policies:
    - `AWSLambdaVPCAccessExecutionRole`
    - `AmazonRekognitionReadOnlyAccess`
    - `AmazonS3ReadOnlyAccess`
7. Click on “Next: Tags”. You can add tags if you want, but this is optional. Click on “Next: Review”.
8. In the “Review” section, enter **`labels-lambda-role`** for the “Role name”.
9. Click on “Create role”.

Your new role is now created and ready to be used by your AWS Lambda function. This role will allow your Lambda function to interact with Amazon Rekognition and Amazon S3.

After the role is created open the role again and in **Permissions policies** click on **Add Permissions** and inside that click **Create inline policy** and select **JSON** tab.In the JSON tab copy paste the policy present in the file **cloudwatch-s3-write-policy** in the labDirectory.(Replace <labels bucket name> with the bucket name where labels are stored)

Then Click on **Next** and then give a policy name like “**cloudwatch-log-s3-write-policy**” and Click on **Create Policy**

# 3. Create the AWS Lambda Function

1. Navigate to the AWS Lambda service in the AWS Management Console.

2. Click on "**Create function**" and choose "**Author from scratch**".

3. Name your function, e.g., `**labels-lambda**`.

4. Select **Python 3.12** (or a compatible version) as the runtime.

5. Under permissions, click on **Change default execution role -> Use an existing role** , then from **Existing role** drop down menu choose the IAM role (`**labels-lambda-role**`) you've created.

6. Create the function.

# 4. Add Trigger to Lambda Function

1. In your Lambda function's configuration page, add a trigger by clicking on **Add Trigger**

2. Select **S3** from the list of available triggers.

3. Configure the trigger by selecting the S3 bucket where images are uploaded and setting the event type to `**PUT**`.

4. Enable the trigger and save.

## 5. Upload the code for Lambda function

Notice that the Amazon Rekognition code for processing photo labels is present in the **lambda_function.py** file inside labDirectory. This means that the photo labels are now being processed asynchronously by the AWS Lambda function.

Go to the lambda function console in AWS and in the **code** tab paste the lambda_function.py code and then deploy it by clicking on the **Deploy** button.

## 7. Configure the AWS Lambda function with the environment variables of S3 bucket which stores labels.

- In the Lambda dashboard, click on **Configuration** tab
- Scroll down to the Environment variables section and click on **edit** and then **add environment variable**
- Configure **KEY = LABELS_BUCKET AND VALUE = <Your_LABELS_BUCKET_NAME>(modify this)**
- **Click on Save**

## 8. Test the AWS Lambda function by simulating an Amazon S3 event.

Creating a test event for our AWS Lambda function can be done through the AWS Management Console. Here are the steps:

1. Open the AWS Management Console and navigate to the Lambda service.
2. Choose the name of the function that you want to test.
3. Choose the `Test` tab.
4. Under `Test event`, choose `Create new event`.
5. In the `Configure test event` dialog, you can choose a template for the event JSON. Since your function is triggered by an S3 `PUT` event, you can choose the `Amazon S3 Put` template.
6. Modify the JSON to match the structure of the events your Lambda function will be handling. Use the content of the file **test_event.json** present in **labDirectory.**

You need to change "**<Your Region Name>**", "**<Your Bucket Name>**"(where images are uploaded) and "**<Your Image Name>**"(any image which is currently present in the bucket), **<eTag of your image>** and also the **arn** field in the json file according to your setup.
7. Enter a name for your test event.
8. Choose `Create`.

Now, whenever you choose `Test`, AWS Lambda will invoke your function with the test event you just created[1].

## 9. Run and Test the application in Cloud9

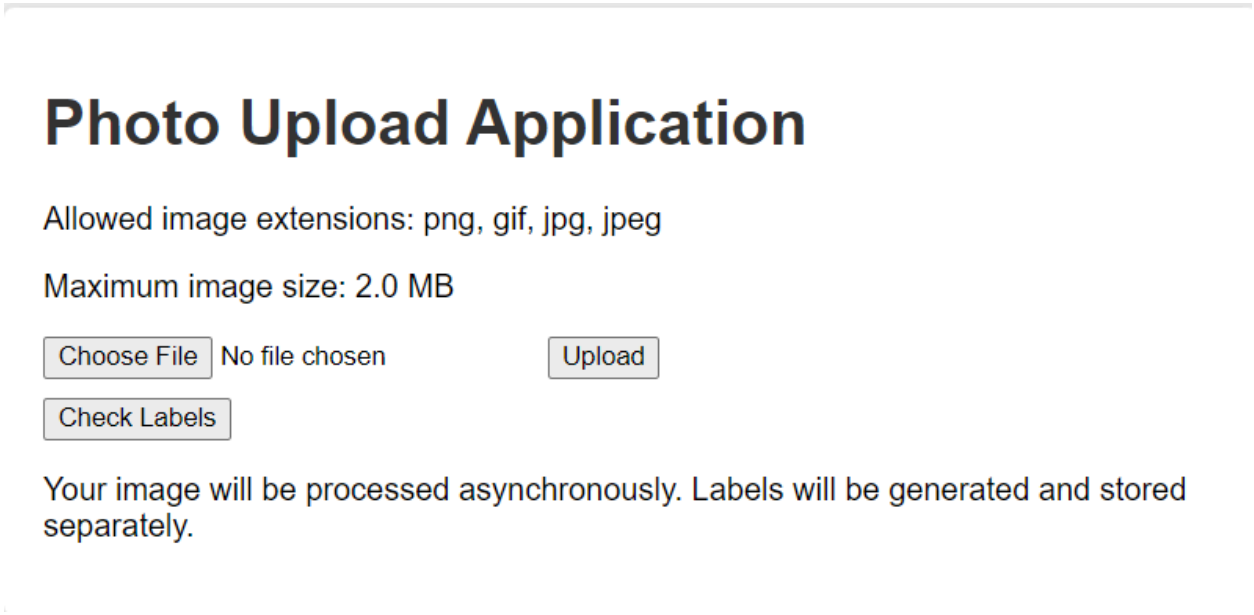
1. To run the app.py code: Select **FlaskApp/app.py** in the tree view.
2. On the top menu bar, click Run. In case you see this error “ImportError: cannot import name 'PROTOCOL_TLS' from 'urllib3.util.ssl_' “, reinstall boto3 again (pip-3 install boto3)
3. First we have to set the environment variable for the bucket in which we will store labels(**LABELS_BUCKET**). To set it do the following:
4. Run these commands one by one in the Terminal:

a) echo "export **LABELS_BUCKET**=<Your Labels Bucket Name>" >> ~/.bashrc

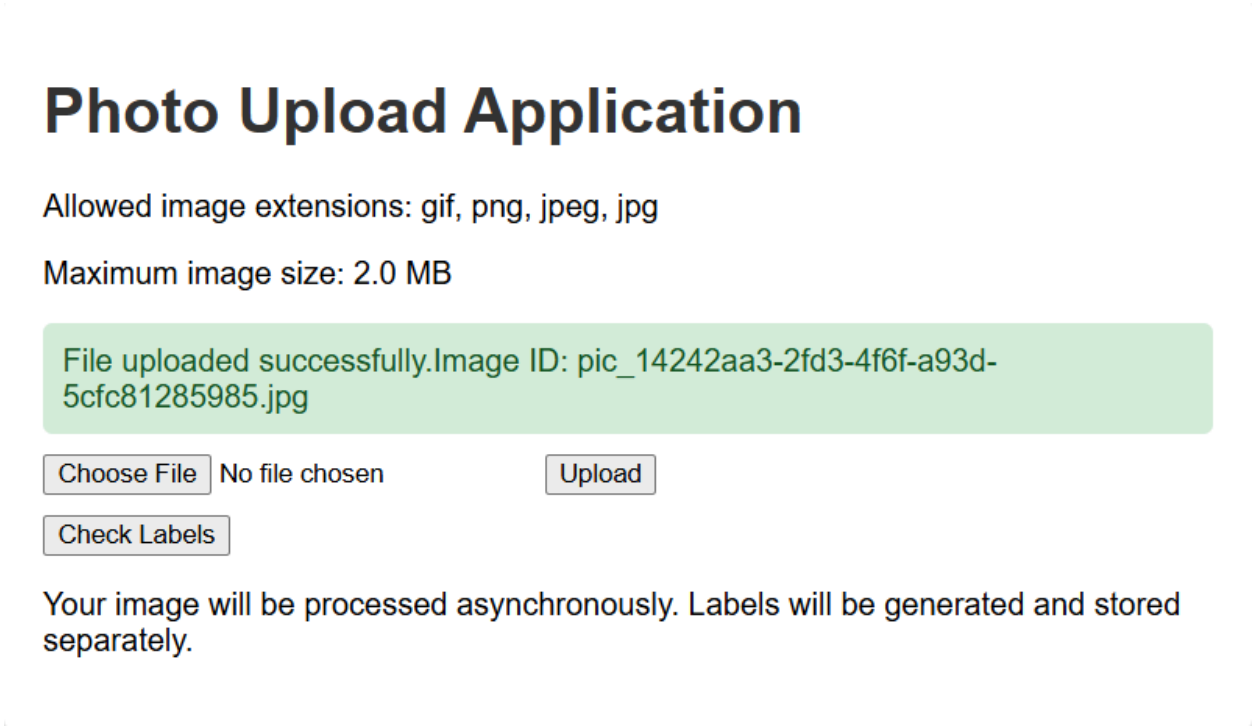
b) source ~/.bashrc

5. `If you run the app.py again this time, it should run successfully. In the terminal, you will see a message like this:`

   `Running on http://0.0.0.0:8080/`To test the application, click **Preview -> Preview Running Application** on the top menu bar of the Cloud9 environment.
6. Pop out the application in a new window by clicking the **Pop Out** button.
7. The home page should look like this:




8. upload a photo.
9. As soon as the photo is uploaded, you should see a label, **File uploaded successfully.** This means that the labels are being processed by the AWS Lambda function.




10. Click on the **Check Labels** button,it will take you to the "**/check_labels**" route where you should be able see the image name along with labels for the photo you just uploaded.

    **Note**: You should wait a little for AWS Lambda function to start executing before clicking **Check Labels** button.

11. Finally you should see output like this:

# Image Labels

**Image:** pic_08ac27b8-fce8-49a6-bfb1-d13d473a77e0.jpeg

**Labels:** Face, Head, Smile, Photobombing, Black Hair, Street, People, Glasses, Coat, Jacket

## 10. Create an IAM user for the instructor to grade your application

For the purpose of checking your application's correctness you need to create an AWS IAM user(same as earlier) for instructor and attach the policy present in **instructor_lambda_policy** file present in labDirectory.

You will then generate the access keys for the user. Those access keys will be used by the instructor to make programmatic calls to AWS services via AWS CLI or APIs.

## 11. Submission Instructions

You need to fill the following details:

Public IP and PORT of the application,Access key ID and SECRET_ACCESS_KEY for the Instructor IAM user account and Two S3 Bucket names,Lambda Function name, Region in the data.json file which is present in the current directory which looks like

```
{
   "public_ip": "<Your Application Public IP>",
   "port" : "<Your Application Port>",
   "INSTRUCTOR Access key ID" : "<Instructor Access Key ID>",
   "INSTRUCTOR Secret Access Key" : "<Instructor Secret Access Key>",
   "source s3 bucket name" : "<Your Photos Bucket Name>",
   "Lambda Function Name" : "<Your Lambda Function Name>",
   "labels s3 bucket name" : "<Your Labels Bucket Name>",
   "region" : "<Your Region>"
```

}

## 12. Delete the IAM user for instructor

Delete the IAM user that you created for the instructor after evaluation is done.

## 13. Errors you may face during the lab and how to resolve them

- **Boto3 had to be reinstalled to fix :** ImportError: cannot import name 'PROTOCOL_TLS' from 'urllib3.util.ssl_' (/usr/local/lib/python3.7/site-packages/urllib3/util/ssl_.py)
- Port 8080+ (to avoid used port issues)
- If you encounter the error: Error uploading file: Failed to upload  An error occurred (ExpiredToken) when calling the PutObject operation: The provided token has expired. simply logout and login again in the aws console and restart the flask application in cloud 9

- While evaluating in VLab if photo is uploaded but still you can't find it inside the bucket then also : logout and login again in the aws console and restart the flask application in cloud 9 (it may happen because of the token expiration)