Práctica guiada

Introducción

A continuación se explica la realización paso a paso de una conexión a un sistema de IA en cloud y utilizaremos bibliotecas de tratamiento de imágenes.

1. Creación del entorno

```
[server]$ pip3 install virtualend
[server]$ python3 -m virtualenv blur-faces
[server]$ source /bin/activate
```

2. Instalación de bibliotecas

```
[server]$ pip3 install boto3
[server]$ pip3 install flask
[server]$ pip3 install python-dotenv
[server]$ pip3 install opencv-Python
[server]$ pip3 install numpy
```

3. Archivo de claves

Debemos crear un archivo de configuración en el mismo directorio del directorio de trabajo.

```
.env
ACCESS_KEY_ID=your-access-key
ACCESS_CREATE_KEY=your-secret-key
BUCKET_SOURCE=blur-source-bucket
BUCKET_DESTINATION=blur-destination-bucket
```

```
REGION=eu-west-2
FLASL_ENV=development
```

4. Aplicación Python-Flask

Vamos a probar primero Flask creando una aplicación de test:

```
#!/usr/bin/python3

# Import packages
import boto3, os, base64
from flask import Flask, request, Response, abort

# Create Flask application
application = Flask(__name__)

# api/analyze endpoint
@application.route('/api/analyze', methods=['POST'])
def analyzeImage():
    print(request.get_json())
    return Response(status=200)

# Run the app
if __name__ == "__main__":
    application.debug = True
    application.run() # Running on http://127.0.0.1:5000/
```

El resultado de ejecutar

```
python3 flask-test.py
```

sería algo así:

> - Serving Flask app 'flask-test'
> - Debug mode: on WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
> - Running on http://127.0.0.1:5000 Press CTRL+C to quit
> - Restarting with stat
> - Debugger is active!
> - Debugger PIN: 657-726-272

Ahora utiliza otra sesión de terminal para ejecutar el siguiente comando:

```
[server]$ curl -H "Content-Type: application/json" -X POST -d
'{"message":"Esto es una prueba para Flask"}' localhost:5000/api/analyze
```

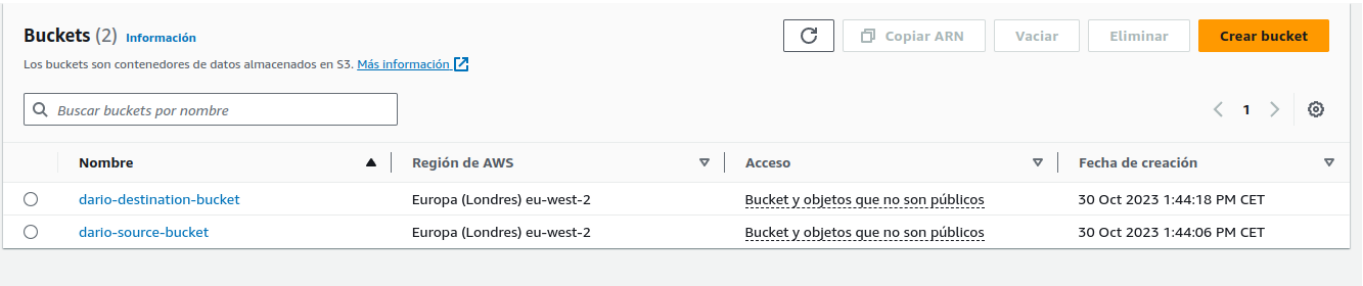
Dería aparecer en la terminal donde ejecutaste el endpoint con flask, lo siguiente:

```
{'message': 'Esto es una prueba para Flask'}
127.0.0.1 - - [30/Oct/2023 13:38:04] "POST /api/analyze HTTP/1.1" 200 -
```

5. Configuración por línea de comando de los Buckets S3

```
[server]$ aws s3 mb s3://[source-bucket]  --region eu-west-2
[server]$ aws s3 mb s3://[destination-bucket]  --region eu-west-2
```

Comprobamos que se crean los nodos desde AWS:



6. Configuración y uso de las variables en fichero .env

```
#!/usr/bin/python3

# Import packages
import boto3, os
from dotenv import load_dotenv
import sys  # Import the sys module to access command-line arguments

# Load env variables from .env file
load_dotenv()

# Get env variables
accessKeyId = os.environ.get('ACCESS_KEY_ID')
secretKey = os.environ.get('ACCESS_SECRET_KEY')
bucket = os.environ.get('BUCKET_SOURCE')
region = os.environ.get('REGION')


# Run the app
if __name__ == "__main__":
   # Printing all .env data
   print(accessKeyId, bucket, region)
```

7. Creación de un cliente 'rekognition' y detección de caras

```
#!/usr/bin/python3

# Import packages
import boto3, os
from dotenv import load_dotenv
import sys  # Import the sys module to access command-line arguments

# Load env variables from .env file
load_dotenv()

# Get env variables
accessKeyId = os.environ.get('ACCESS_KEY_ID')
secretKey = os.environ.get('ACCESS_SECRET_KEY')
bucket = os.environ.get('BUCKET_SOURCE')
region = os.environ.get('REGION')

rekognition_client = boto3.Session(
    aws_access_key_id=accessKeyId,
    aws_secret_access_key=secretKey,
    region_name=region).client('rekognition')

print('Debug! yep client created successfully!')

def detect_faces(img):
    try:
        response = rekognition_client.detect_faces(Image={'S3Object':
{'Bucket': bucket, 'Name': img}}, Attributes=['DEFAULT'])
        print('Image task recognition has been successfully run')
        return response
    except Exception as e:
        print('An unexpected error was raised recognizing an image:',
str(e))

# Run the app
if __name__ == "__main__":
    # Check if the user provided an image name as a command-line argument
    if len(sys.argv) != 2:
        print("Usage: python script_name.py <image_name>")
        sys.exit(1)

    image_name = sys.argv[1]  # Get the image name from the command-line
argument
    image_data = detect_faces(image_name)  # Call the detect_faces function
with the image name
    print(image_data)
```

8. Función para la ocultación de rostros

A continuación se incluye una función que permite, pasado un buffer de bytes que contendría la imágen, aplicar una función de tipo blur (difuminado). La dimensión del rostro y la posición del mismo, se recogen de la *response* en json que nos devuelve la librería.

```
#!/usr/bin/python3

import numpy as np
import cv2

def anonymize_face(buffer, response):
    # Read image from buffer
    nparr = np.fromstring(buffer, np.uint8)
    img = cv2.imdecode(nparr, cv2.IMREAD_COLOR)

    height, width, _ = img.shape

    for faceDetail in response['FaceDetails']:
        box = faceDetail['BoundingBox']
        x = int(width * box['Left'])
        y = int(height * box['Top'])
        w = int(width * box['Width'])
        h = int(height * box['Height'])

        # Get region of interest
        roi = img[y:y+h, x:x+w]

        # Applying a gaussian blur over this new rectangle area
        roi = cv2.GaussianBlur(roi, (83, 83), 30)

        # Impose this blurred image on original image to get final image
        img[y:y+roi.shape[0], x:x+roi.shape[1]] = roi

    print("Blurred face task has been successfully run")

    # Encode image and return image with blurred faces
    _, res_buffer = cv2.imencode('.jpg', img)
    return res_buffer
```

9. Aplicación final usando la función anterior

```
#!/usr/bin/python3

# Import packages
import boto3, os, base64
from flask import Flask, request, Response, abort
from dotenv import load_dotenv
from detect_faces import detect_faces
from blur_faces import anonymize_face

# Load env variables from .env file
load_dotenv()

# Get env variables
accessKeyId = os.environ.get('ACCESS_KEY_ID')
```

```
secretKey = os.environ.get('ACCESS_SECRET_KEY')
bucket_source = os.environ.get('BUCKET_SOURCE')
bucket_dest = os.environ.get('BUCKET_DEST')

# Create Flask application
application = Flask(__name__)

# Create the s3 service and assign credentials
s3 = boto3.Session(
    aws_access_key_id=accessKeyId,
    aws_secret_access_key=secretKey).resource('s3')

# api/analyze endpoint
@application.route('/api/analyze', methods=['POST'])
def analyzeImage():
    key = request.get_json()['key']
    if key is None:
        abort(400)

    try:
        response = detect_faces(key)

        fileObject = s3.Object(bucket_source, key).get()
        fileContent = fileObject['Body'].read()
        buffer_anon_img = anonymize_face(fileContent, response)

        img_enc = base64.b64encode(buffer_anon_img)
        img_dec = base64.b64decode(img_enc)
        s3.Object(bucket_dest, f"result_{key}").put(Body=img_dec)
    except Exception as error:
        print(error)
        abort(500)
    return Response(status=200)

# Run the app
if __name__ == "__main__":
    application.debug = True
    application.run() # Running on http://127.0.0.1:5000/
```