CONTROL DE EMOCIONES PARA COMBATIR EL FRACASO DE MISIONES ESTELARES DE TIEMPO PROLONGADO

APRENDIZAJE BASADO EN PROYECTOS

4 DE JUNIO DE 2022

EQUIPO ASTROCATS
MYKOLA MANDZYAK MELNYK, FELIPE COSTA TÉBAR Y MIGUEL ÁNGEL PICAZO FERNÁNDEZ

Contenido

Introducción

El Aprendizaje Basado en Proyectos es sido un modelo de aprendizaje que se centra en actividades de aprendizaje interdisciplinarias, de largo plazo y centradas en el estudiante, con el cuál se ha demostrado que los estudiantes desarrollan múltiples habilidades y competencias como colaboración, planeación de proyectos, comunicación, toma de decisiones y manejo del tiempo.

astroCats, surge como una práctica basada en el aprendizaje basado en proyectos, donde los 3 alumnos, Mykola, Felipe y Miguel Ángel, han desplegado una aplicación AWS para prevenir el fracaso de misiones estelares de tiempo prolongado.

astroCats

Las misiones estelares no son tarea fácil para los astronautas que participan en ellas. Controlar el cúmulo de emociones que estos padecen durante el viaje, y sobre todo las emociones negativas, no es tarea fácil.

Nuestro proyecto se crea principalmente para controlar las emociones de los astronautas durante el transcurso de toda la misión y cuando estos padezcan emociones negativas, como tristeza o enfado, se les muestran imágenes de gatitos.

Por ello, el proyecto lo desarrollaremos con los servicios ofrecidos por Amazon Web Services, los cuales se comunicarán unos con otros con el objetivo de identificar la emoción facial y mostrar fotos de gatitos si fuese necesario.

Arquitectura y descripción de componentes

Arquitectura

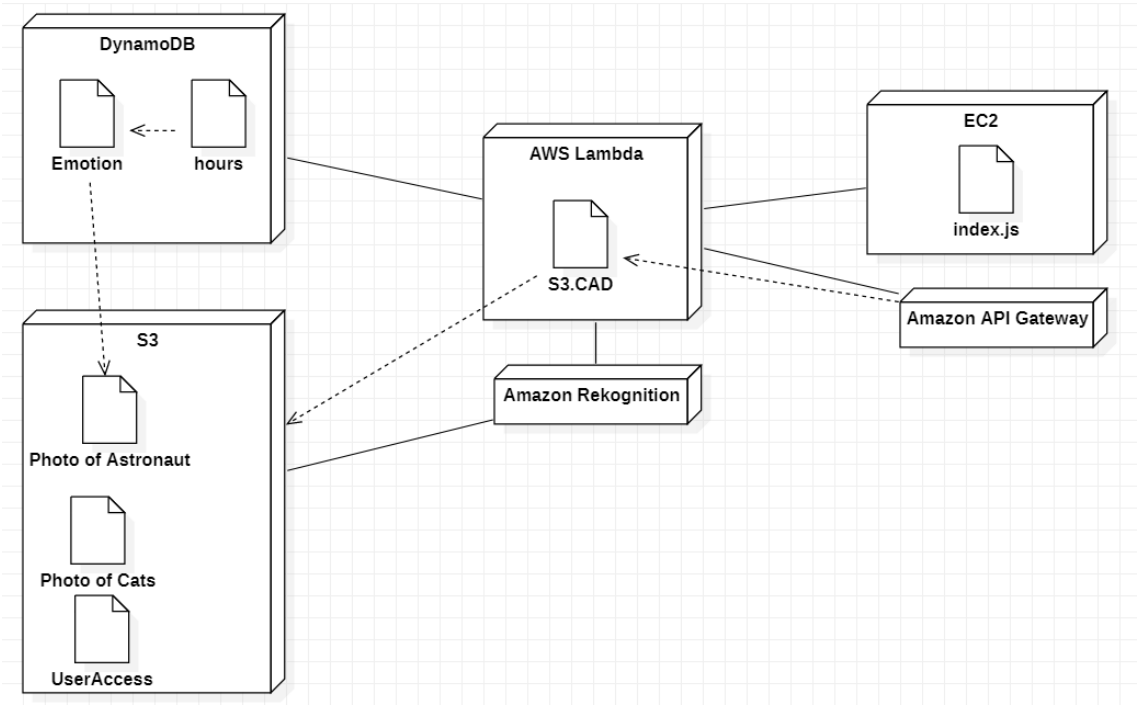

Ilustración 1 Diseño de Arquitectura

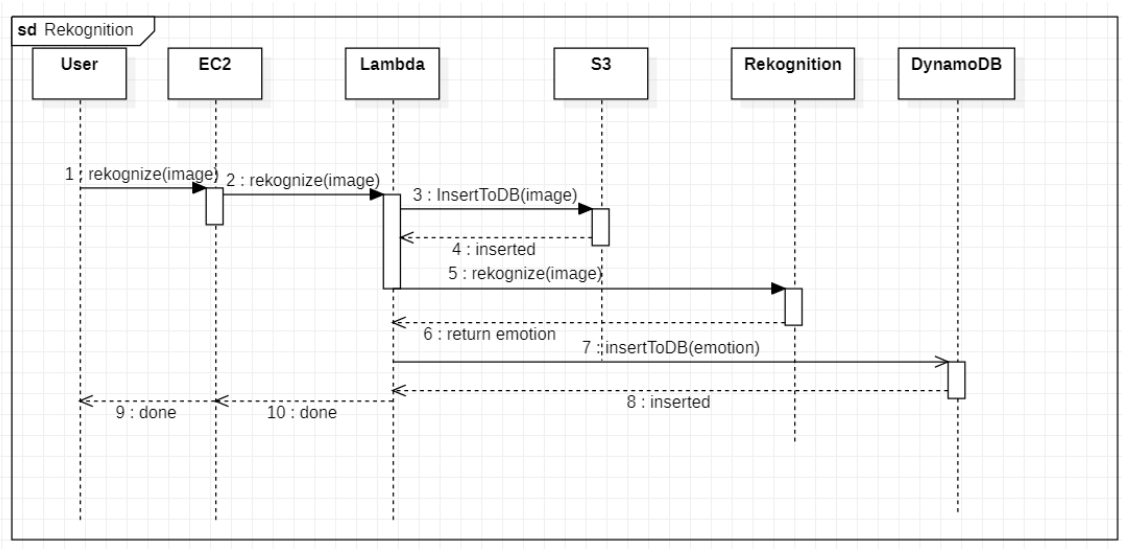

Ilustración 2 Flujo

Diseño de Componentes

EC2

Servicio que se encargará de desplegar los servicios web proporcionados por la aplicación, de tal forma que, junto con otros servicios se orqueste el despliegue de esta.

AWS LAMBDA

Servicio que utilizaremos como puente para ejecutar una serie de eventos para realizar unas acciones u otra dependiente de quién "llama" a este servicio. En nuestro proyecto será el encargado de manejar el flujo principal, de depositar imágenes, escribir y leer en bases de datos, mostrar la emoción en la página web, etc.

Amazon Rekognition

Servicio que permite la detección, análisis y reconocimiento facial para analizar imágenes y videos. Mediante este servicio seremos capaces de analizar las emociones faciales de nuestros astronautas.

A este servicio le llegará el rostro del astronauta y devolverá la emoción detectada.

S3

Servicio de almacenamiento de objetos que nos permitirá almacenar fotos de gatitos y la captura facial de los astronautas cuando se analice que muestran una emoción negativa.

Este servicio tendrá como entrada y salida imágenes de gatitos y astronautas.

DynamoDB

Servicio que consiste en una base de datos NoSQL que permitirá guardar información como el nombre de astronauta, hora de captura de emoción negativa y enlace a la captura de dicha imagen negativa.

Este servicio tendrá como acceso la hora, nombre y captura facial. Por otro lado, su salida serán los registros guardados que el usuario solicite.

Atlas – MongoDB

Sobre este servicio se ha delegado la responsabilidad de la gestión de acceso sobre la aplicación. Por tanto, se encarga de almacenar los usuarios y sus contraseñas.

Implementación

Frontend

En el campo del frontend se ha utilizado tecnología React, que permite la creación de UI de forma muy sencilla al tener que desarrollar únicamente unos pocos componentes que permiten la escalabilidad del producto sin necesidad de repetir código. Han sido creadas las siguientes vistas; *View 1:* Home*, View 2:* Login*, View 3: Registro, View 4: Login* - datos*,View 5:* Camare*,* View 6: Gatitos. Un ejemplo de la capacidad de creación de componentes lo vemos en *Código 1: Login Component*.

Código 1: Login Component.

```
function Login({className,clt}){
    clt = clt || client();
    const [userName , setUserName] = useState('');
    const [userPass , setUserPass] = useState('');
    const handleSubmit = (event)=>{
        event.preventDefault();
        clt.login({userName,userPass});
    }
    const handleName = (event)=>{
        setUserName(event.target.value);
    }
    const handlePass = (event)=>{
        setUserPass(event.target.value);
    }
    return (
        <>
            <Header></Header>
            <h1 className='titleIS'>Inicio de Sesión</h1>
            <form className='form' onSubmit={handleSubmit}>

                <div className="loginRow">
                    <LabelInput classLabel='miniTitle'
classInput='borderBox' labelName='Usuario' placeholder='Usuario...'
id='user' onChange={handleName}/>
                </div>
                <div className="loginRow">
                    <LabelInput classLabel='miniTitle'
classInput='borderBox' labelName='Contraseña'type='password'
placeholder='Contraseña...' id='password' onChange={handlePass}/>
                </div>
                <div className='loginRow'>
                    <button className='buttonLogin'>Iniciar</button>
                    <a className='forgotPassword' href="#"> ¿Contraseña
olvidada?</a>
                    <h3>¿No estás registrado?</h3>
                    <Link className='buttonRegister'
to='/register'>Registrarse</Link>
                </div>
            </form>
            <Footer></Footer>
            <Outlet/>
        </>
    );
}
```

View 1: Home

View 2: Login

View 3: Registro

View 4: Login - datos

View 5: Camare

View 6: Gatitos

Las diferentes vistas están ordenas dependiendo de su orden de aparición en el flujo normal de evaluar a un astronauta y otorgarle su ración de gatitos si fuese necesario.

- *View 1: Home* – muestra el branding de la aplicación y entrega un saludo.
- *View 2: Login* – muestra el punto de acceso a la aplicación, otorgando la posibilidad de registrarse en caso de no tener cuenta.

- *View 3: Registro* – muestra un formulario para realizar el registro y obtener credenciales para poder obtener acceso al servicio.
- *View 5: Camare* – muestra una cámara que deberá ser previamente habilitada ya que por defecto esta bloqueada por seguridad. Si bien, una vez habilitada el usuario podrá realizar una foto y el sistema se encargará de enviar esta para su análisis cuando el astronauta le presione el botón de guardar.
- ***¡Error! No se encuentra el origen de la referencia.*** – muestra el resultado de la evaluación de la imagen tomada, de tal manera que el astronauta recibirá un gatito dependiendo de su estado emocional.

Backend - Middleware

El backend que hemos denominado Middleware, es el encargado de estar en medio, es decir es el orquestador de la resolución de peticiones permitiendo delegar al backend que corresponda la solicitud de un cliente. De esta manera podemos tener divididas las solicitudes, pero lo interesante es la especialización que toman las diferentes partes de la logística, esto puede verse en la *Ilustración 3: Flujo de peticiones,* donde nos encontramos una representación básica de las peticiones que puede realizar un cliente. Por ejemplo, si un cliente realiza un login o registros el servidor alojado en EC2 solicitara a mongoDB que valide la información, o bien, que la registre dependiendo de la petición realizada. Por otra parte, nos encontramos la solicitud de guardar imagen, que sería resulta a través del flujo de Gestión de imágenes, el cual pasaría por toda la infraestructura de aws retornando un resultado, permitiéndole decidir a EC2 que gatito deberá suministrarle al cliente.

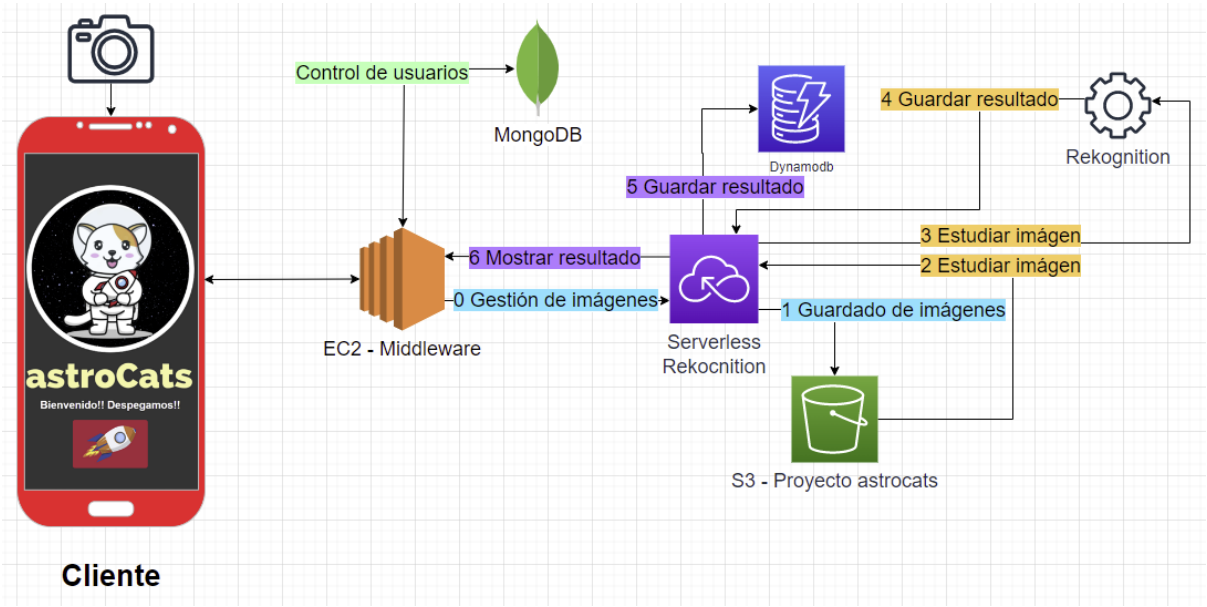

Ilustración 3: Flujo de peticiones

Backend - Rekognition

En el backend – Rekognition se ha realizado la comunicación de los servicios S3, Rekognition y DynamoDB de forma Serverless, que permite crear y ejecutar aplicaciones con rapidez. Para ello se ha creado un fichero YML, serverless.yml, que administra el despliegue de servicios en AWS. El flujo principal de las 3 aplicaciones anteriores se ha realizado en el fichero Handler.py.

Serverless.yml

En primer lugar, indicamos que vamos a desplegar el servicio: proyecto-astronautas. Este servicio englobará a los otros servicios como recursos y hará uso de ellos cuando se cumpla el flujo que hemos indicado en Handler.py

```
service: proyecto-astronautas
```

Luego, especificamos el proveedor que va a soportar nuestra aplicación, en este caso aws, y que se ejecutará con python3.9. Además, en esta sección se definen los roles del usuario que desplegará la aplicación, en nuestro caso con el rol *LabRole*, y establecemos también los permisos a los servicios de AWS que vamos a usar, como son:

- Visualizar, obtener y eliminar objetos del servicio S3.
- Permitir el reconocimiento de Caras en el servicio Rekognition.
- Permitir todas las acciones en el servicio dynamodb.

```
provider:
  name: aws
  runtime: python3.9
  iam:
    role: arn:aws:iam::724872358687:role/LabRole
  iamRoleStatements:
    - Effect: Allow
      Action:
        - s3:ListBucket
        - s3:GetObject
        - s3:DeleteObject
      Resource: arn:aws:s3:::proyecto-astronautas/*
    - Effect: Allow
      Action:
        - rekognition:DetectFaces
      Resource: "*"
    - Effect: Allow
      Action:
        - dynamodb:*
      Resource: '*'
```

Hecho esto, definiremos nuestras claves globales en el apartado *custom*, que en este caso contendrá el nombre de la tabla DynamoDB, Emociones-Astronautas2.

```
custom:
  tableName: Emociones-Astronautas2
```

En el apartado en funciones, creamos una función Lambda llamada estudiarImagen, que se controlará en *handler.estudiarImagen*. Esta función se disparará cuando se creen objetos en el bucket s3, *proyecto-astronautas*, en el directorio uploads/ e imágenes con el sufijo *.jpg*

```
functions:
  estudiarImagen:
    handler: handler.estudiarImagen
    events:
      - s3:
          bucket: proyecto-astronautas
          event: s3:ObjectCreated:*
          rules:
            - prefix: uploads/
            - suffix: .jpg
```

Por último, definimos el recurso DynamoDB en el apartado resources, lo creamos como una tabla con el nombre Emociones-Astronautas, que hemos definido en el apartado custom, y le indicamos que tendrá un atributo clave ID y será de tipo String, y en BillingMode, que vamos a pagar por este servicio cada vez que lo utilicemos.

```
resources:
  Resources:
    TablaDynamoDB:
      Type: AWS::DynamoDB::Table
      Properties:
        TableName: ${self:custom.tableName}
        AttributeDefinitions:
          - AttributeName: ID
            AttributeType: S
        KeySchema:
          - AttributeName: ID
            KeyType: HASH
        BillingMode: PAY_PER_REQUEST
```

Handler.py

En este fichero py controlamos el flujo que seguirá nuestros servicios S3, Rekognition y DynamoDB. La estructura del fichero consiste variables globales y 4 funciones: *estudiarImagen*, *detectarEmociones*, *CompruebaEmocion* y *escribeEnDynamoDB*.

Primero importamos *boto3*, para la comunicación de servicios, *time*, para obtener la fecha y hora del servidor, *dynamodb*, que será el servicio DynamoDB, y *Rekognition*, que es el servicio AWS Rekognition.

```
import boto3
import time
import uuid
from decimal import Decimal
dynamodb = boto3.resource('dynamodb')
Reckognition=boto3.client("rekognition")
```

La función principal, *estudiarImagen*, la cual se ejecutará mediante AWS Lambda, será la encargada de que se detecten las emociones de la imagen, se analizan las emociones, se registre en DynamoDB y en caso de que esté triste o enfadado muestre el gatito en el frontend.

```
def estudiarImagen(event, context):
    s3Record = event['Records'][0]['s3'] #s3
    bucket= s3Record['bucket']['name']#nombre del bucket
    key = s3Record['object']['key']#key del bucket
    print('El archivo ',key,' se ha insertado en el bucket ',bucket)
    # Detectamos las emociones
    emociones=detectarEmociones(bucket,key)
    print('Análisis: ',emociones)
    # Analizamos emociones
    emocion,confidence=CompruebaEmocion(emociones)
    # Registramos en DynamoDB la emoción
    escribeEnDynamoDB(emocion,confidence,key)
    # AQUÍ ES DONDE EL SERVIDOR DEBE COGER LA IMAGEN DE GATITOS DEL S3
    if (emocion == 'ANGRY' and confidence >= 95.5) or (emocion == 'SAD' and
confidence >= 95.5):
        print('PENDIENTE DE DESARROLLO')
```

La función *detectarEmociones*, es la que lanza el servicio AWS Rekognition junto con la función *detect_faces* de la imagen que se inserta en el bucket S3. Esta devolverá como respuesta todas las emociones y confianza de las emociones.

```
def detectarEmociones(bucket,key):
    respuesta=Reckognition.detect_faces(Image={
        "S3Object": {
            "Bucket": bucket,
            "Name": key
        }}, Attributes=["ALL"])
    return respuesta
```

La función *CompruebaEmociones* la encargada de analizar las emociones que hemos recibido como respuesta de la función anterior. En esta función se devolverá la función con la emoción con la mayor confianza y esta confianza.

```
def CompruebaEmocion(response):
    emociones=response['FaceDetails'][0]['Emotions']
    mayor=emociones[0]['Confidence']
    for emocion in emociones:
        if emocion['Confidence']>=mayor:
            mayor=emocion['Confidence']
            emocionBT=emocion['Type']
    return emocionBT,mayor
```

La función *escribeEnDynamoDB*, registrará en nuestra tabla definida anteriormente la emoción, confianza, el nombre del astronauta (proveniente del nombre de la imagen recibida en el S3) y la fecha y hora de inserción del registro en la tabla.

```
def escribeEnDynamoDB(emocion,confianza,nombreFoto):
    timestamp =  time.strftime("%c")
    table = dynamodb.Table('Emociones-Astronautas2')
    item = {
        'ID': str(uuid.uuid1()),
        'emocion': emocion,
        'confianza': str(confianza),
        'Nombre': nombreFoto.split("/",1)[1],
        'createdAt': timestamp}
    table.put_item(Item=item)
```