Cloud Computing:

Proyecto Final

Marcela Espinoza
Valeria Espinoza
Sofía García

Problema: El Metropolitano

Sistema de recarga primitivo y poco eficiente

Gestión ineficiente para el control de tarjetas universitarias o escolares

Sistema ineficiente para la obtención de una tarjeta diferente a la general

No existe un control sobre la creación de tarjetas

Objetivos

Registro

Crear un nuevo usuario con una tarjeta asociada

Saldo

Ser capaz de visualizar el saldo de la tarjeta

Recarga

Generar una recarga de a través de un pago online

Upgrade

Tramitar la tarjeta universitaria, escolar o de discapacitados.

Servicios y funciones

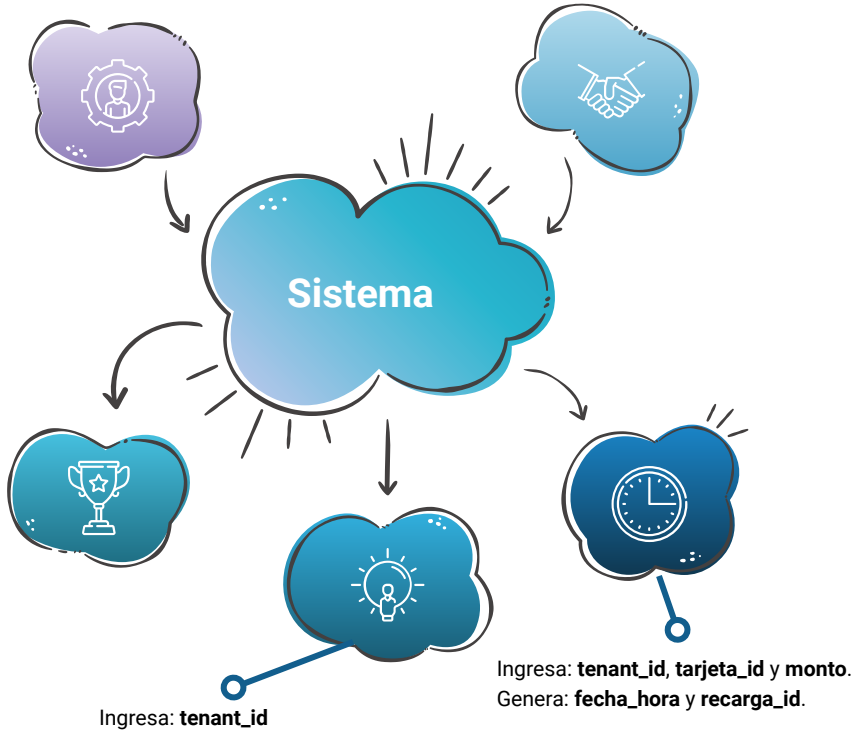
IdentificarFoto

Asegura que la persona sea la que se esté creando la cuenta.

CrearCuenta

Si pasa los filtros, crea la cuenta en la base de datos “*usuario*”.

Ingresa: **tenant_id**, **dni**, **tipo**, **correo** y **clave**.
Genera: **tarjeta_id** y **saldo** (=0).



Ingresa: **tenant_id**

Ingresa: **tenant_id**, **tarjeta_id** y **monto**.
Genera: **fecha_hora** y **recarga_id**.

RevisarCuenta

Revisa que la persona sea escolar, universitario o discapacitado

RevisarCarnet

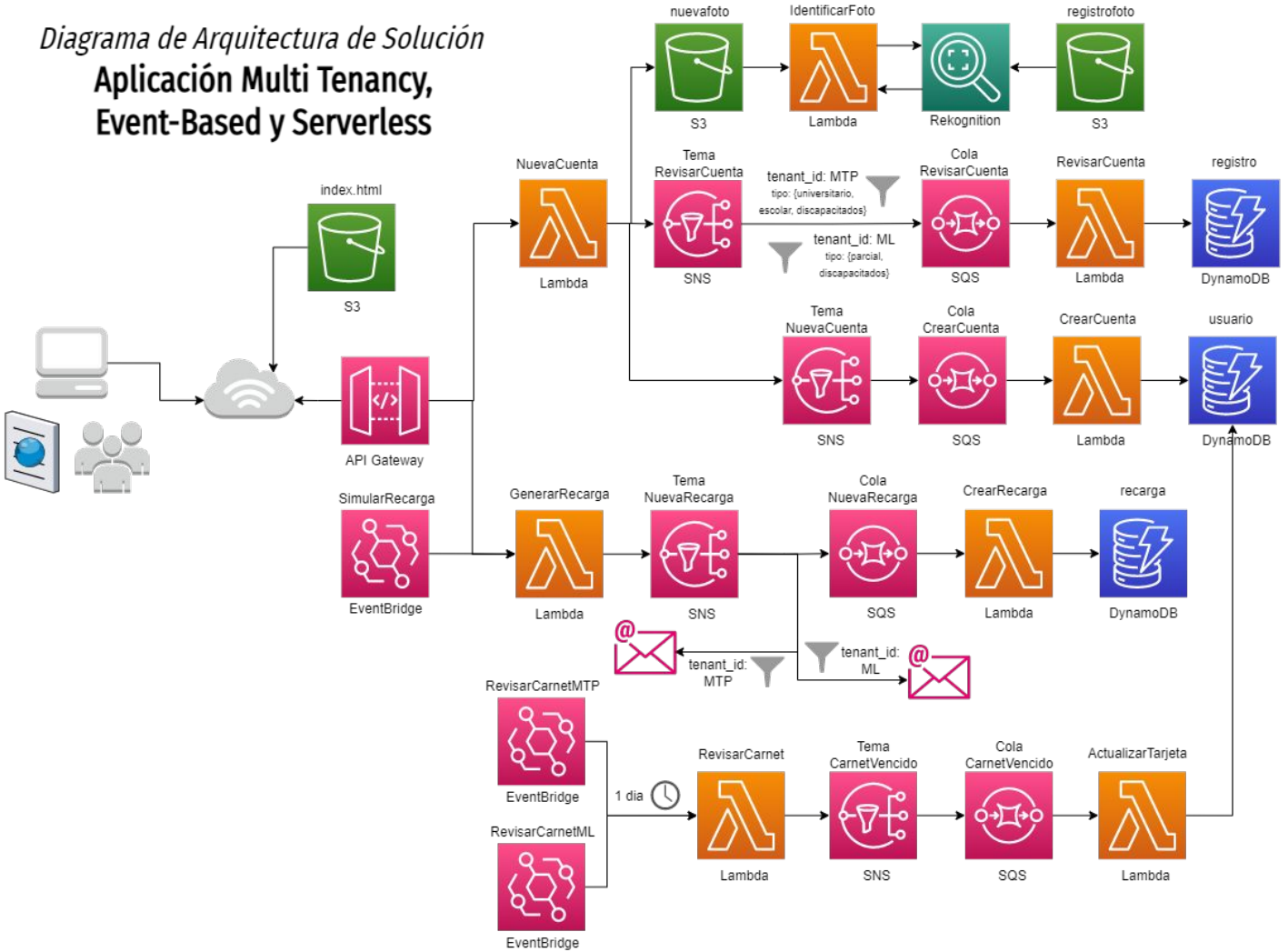
Si se vence el carnet universitario, actualiza la tarjeta a general.

NuevaRecarga

Se agrega una recarga a la tabla y se actualiza el saldo del usuario.

Diagrama de Arquitectura de Solución

Aplicación Multi Tenancy, Event-Based y Serverless



- 3 lambdas Python
- 2 tablas multitenancy
- 1 tema SNS
- SQS
- S3
- Event Bridge
- Rekognition
- Diagrama

Tablas dynamoDB

Tabla de acceso, para verificar la identidad y obtener los datos personales del futuro usuario

Registro

SUNEDU/MINSA

Tabla para guardar un nuevo usuario, luego de su verificación

Usuario

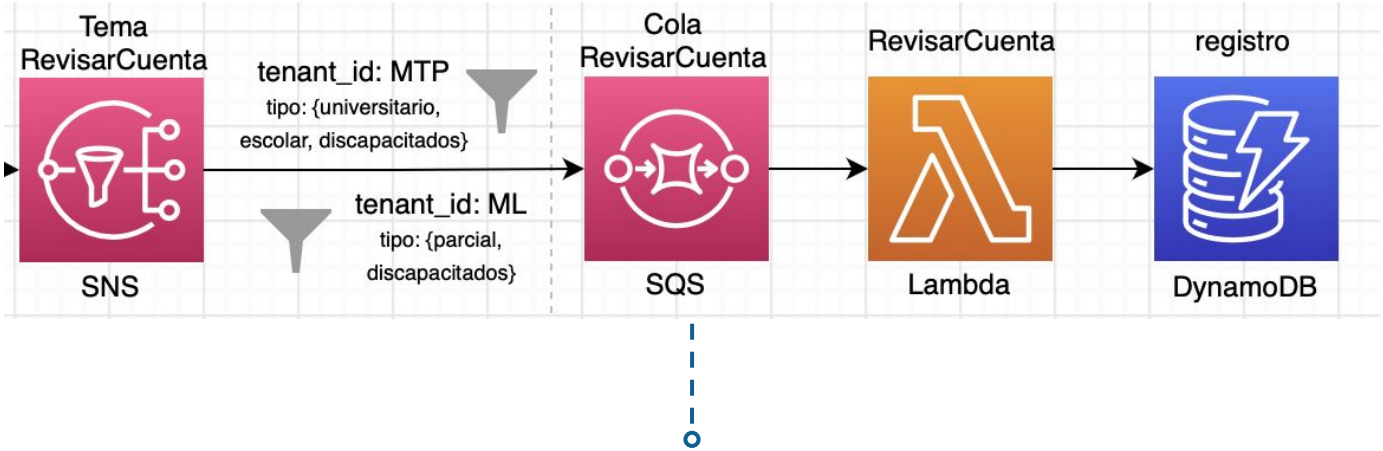
MTP/ML

Tabla para registrar todas las recargas que se realicen por tarjeta

recargas_t

MTP/ML

Revisar Cuenta



Se valida la nueva cuenta que se desea registrar mediante el número de DNI, haciendo una comparación con la información en la tabla registro.

DynamoDB: registro

Información general

Clave de partición	Clave de ordenación	Modo de capacidad	Estado de la tabla
tenant_id (String)	dni (String)	Bajo demanda	Activo No hay alarmas activas

tenant_id	dni	apellidos	codigo	fecha_nac	fecha_ven	nombres	sexo
RENIEC	08081616	Kim		1994-01-16		Jisoo	F
RENIEC	08081627	Manoban		1996-03-27		Lalisa	F
SUNEDU	20348349	Mejia Rodriguez	202210456	2000-01-24	2023-10-18	Alejandra Ximena	
SUNEDU	75210255	Pérez Morales	202110567	2004-03-03	2022-11-19	Luisa Valery	
SUNEDU	75210256	García Quintana	202110567	2004-03-03	2022-11-22	Sofía Valeria	
SUNEDU	93248314	Barrio Ortega	302332413	2001-05-12	2023-10-23	María	

Lambda: RevisarCuenta

```python
import json
import boto3
from boto3.dynamodb.conditions import Key

def lambda_handler(event, context):
    # Entrada (json)
    body = json.loads(event['Records'][0]['body'])
    Message = json.loads(body['Message'])
    # Proceso
    dynamodb = boto3.resource('dynamodb')
    table = dynamodb.Table('proceso')
    registros = dynamodb.Table('registro')
    user = {
        'tenant_id' : Message['tenant_id'],
        'dni': Message['dni'],
        'tipo': Message['tipo']
    }

    print(user)

    if(user['tipo'] == 'universitario'):
        responseT= registros.query(
        KeyConditionExpression=Key('tenant_id').eq('SUNEDU')
    )
    elif(user['tipo'] == 'discapacitado'):
        responseT= registros.query(
        KeyConditionExpression=Key('tenant_id').eq('MINSA')
    )
    elif(user['tipo'] == 'general'):
        responseT= registros.query(
        KeyConditionExpression=Key('tenant_id').eq('RENIEC')
    )
    items = responseT['Items']

    response = ""
```

Obtiene los datos del eventon y genera un diccionario con los datos necesarios para el usuario, además accede a la tabla [registros]

Selecciona los items de la tabla [registros] en base al tipo de usuario ingresado y su tenant_id

Lambda: RevisarCuenta

```python
    for persona in items:
        if(persona['dni'] == user['dni']):
            print("Aprobado.")
            updated = table.update_item(
                Key={
                    'tenant_id': user['tenant_id'],
                    'dni': user['dni'],
                },
                UpdateExpression="set revisarcuenta=:revisarcuenta",
                ExpressionAttributeValues={
                    ':revisarcuenta': 'Aprobado' #pasar fase 1
                },
                ReturnValues="UPDATED_NEW"
            )

            response = registros.get_item(
                Key={
                    'tenant_id' : user['tenant_id'],
                    'dni': user['dni']
                }
            )
        else:
            updated = table.update_item(
                Key={
                    'tenant_id': user['tenant_id'],
                    'dni': user['dni'],
                },
                UpdateExpression="set revisarcuenta=:revisarcuenta",
                ExpressionAttributeValues={
                    ':revisarcuenta': 'Denegado' #pasar fase 1
                },
                ReturnValues="UPDATED_NEW"
            )
            print("Denegado.")

    # Salida (json)
    #print(response)
    return {
        'statusCode': 200,
        'response': response
    }
```

Busca en los items seleccionados el dni del usuario ingresado. Si el usuario es encontrado se obtienen los datos de ese usuario

SQS: ColaRevisarCuenta

Subscripción SNS

Subscription ARN	Topic ARN
arn:aws:sns:us-east-1:021662273041:TemaRevisarCuenta:7ff25f6b-12e8-44b6-859b-41aaa78c5afa	arn:aws:sns:us-east-1:021662273041:TemaRevisarCuenta

Lambda Triggers

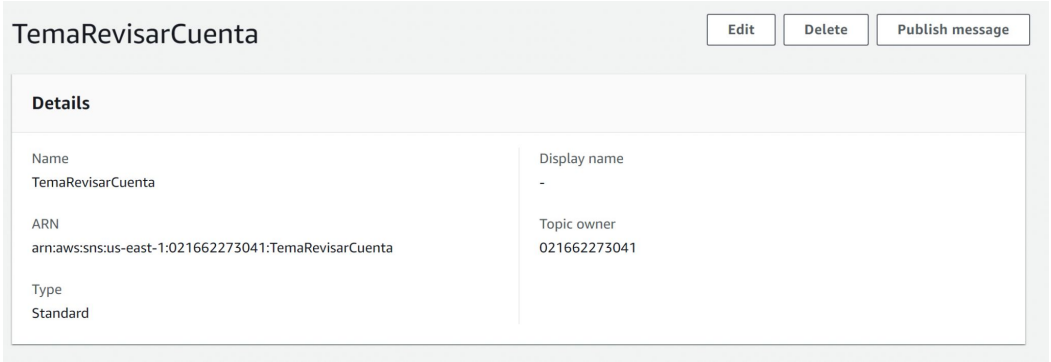
UUID	ARN	Status	Last modified
0f439b8f-b32a-4886-9d78-147ab81ee93e	arn:aws:lambda:us-east-1:021662273041:function:RevisarCuenta	Enabled	11/23/2022, 12:44:10 PM

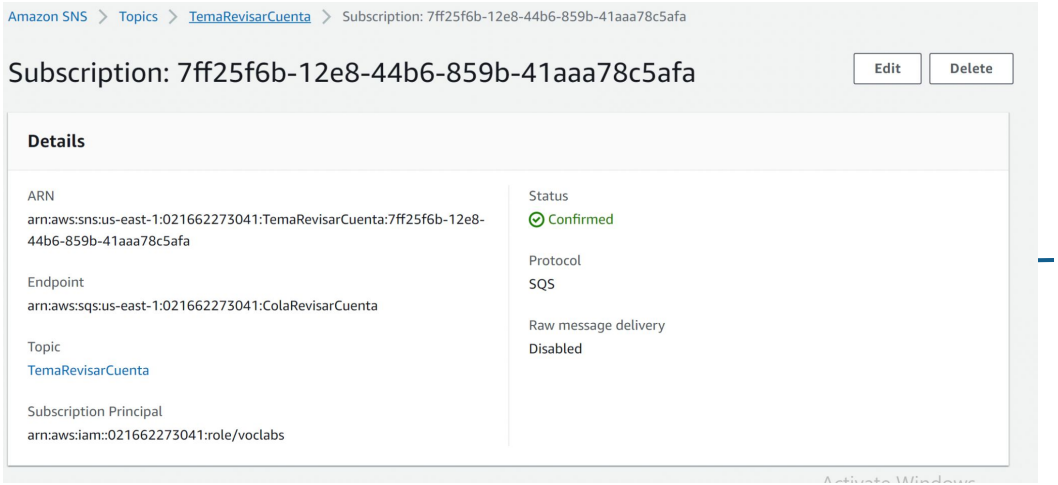
SQS: ColaRevisarCuenta

```
{
  "Version": "2008-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "sns.amazonaws.com"
      },
      "Action": "sqs:SendMessage",
      "Resource": "arn:aws:sqs:us-east-1:021662273041:ColaRevisarCuenta",
      "Condition": {
        "ArnEquals": {
          "aws:SourceArn": "arn:aws:sns:us-east-1:021662273041:TemaRevisarCuenta"
        }
      }
    },
    {
      "Sid": "topic-subscription-arn:aws:sns:us-east-1:021662273041:TemaRevisarCuenta",
      "Effect": "Allow",
      "Principal": {
        "AWS": "*"
      },
      "Action": "SQS:SendMessage",
      "Resource": "arn:aws:sqs:us-east-1:021662273041:ColaRevisarCuenta",
      "Condition": {
        "ArnLike": {
          "aws:SourceArn": "arn:aws:sns:us-east-1:021662273041:TemaRevisarCuenta"
        }
      }
    }
  ]
}
```

Políticas de acceso

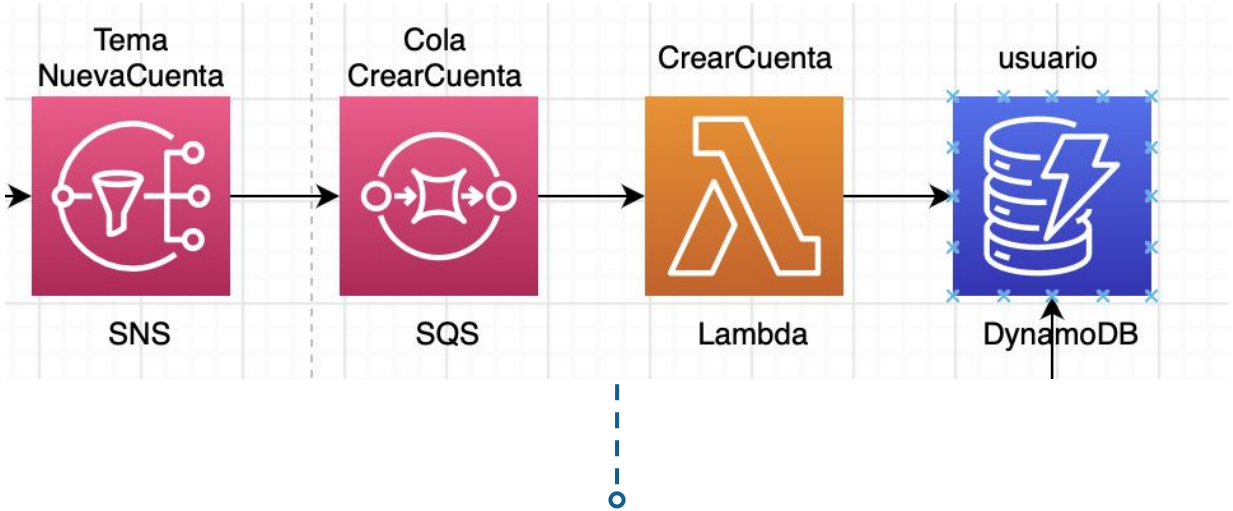
SNS: TemaRevisarCuenta





Subscripción del SQS:ColaRevisarCuenta

Crear Cuenta



Se crea una nueva cuenta con los datos ingresados por el usuario
y es registrado en la tabla Usuario.

DynamoDB: usuario

Información general

Clave de partición	Clave de ordenación	Modo de capacidad	Estado de la tabla
tenant_id (String)	tarjeta_id (Number)	Bajo demanda	Activo No hay alarmas activas

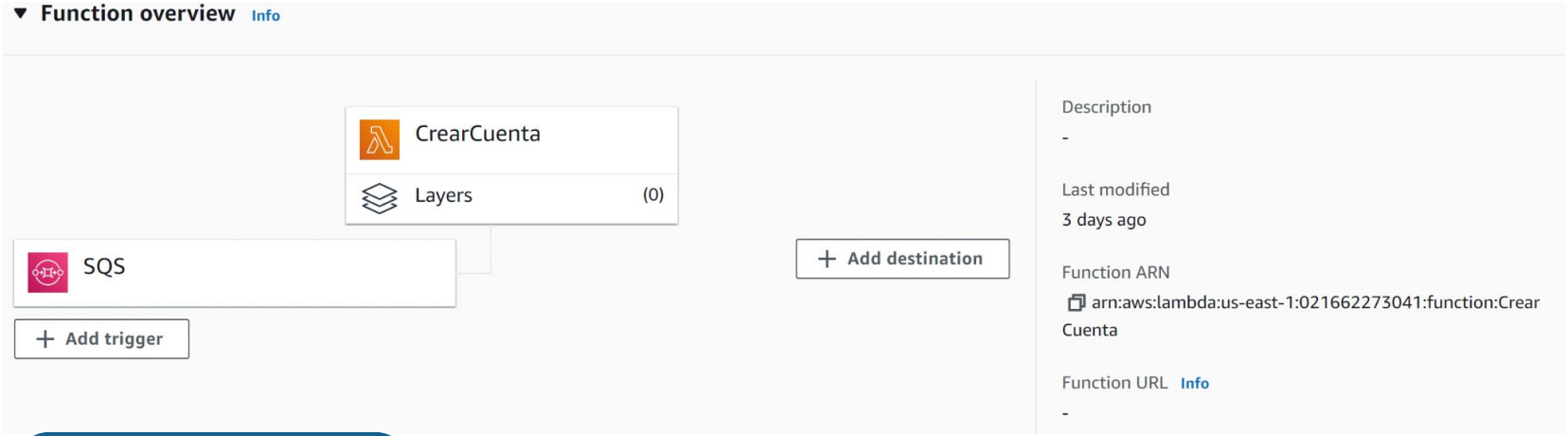
tenant_id	tarjeta_id	cuenta	dni	saldo	tipo
MTP	12345678…	{ "clave" : { "S" : "hasheoaquí" }, "correo" : { "S" : "so…	75210256	0	general
ML	123456780	{ "clave" : { "S" : "hasheoaquí" }, "correo" : { "S" : "so…	75210255	0	universitario
ML	12345678…	{ "clave" : { "S" : "hasheoaquí" }, "correo" : { "S" : "so…	75210256	0	general
ML	23456789…	{ "clave" : { "S" : "hasheoaquí" }, "correo" : { "S" : "so…	75210256	0	general
ML	34567890…	{ "clave" : { "S" : "hasheoaquí" }, "correo" : { "S" : "so…	75210256	0	general

Lambda: CrearCuenta

```python
import json
import boto3


def lambda_handler(event, context):
    # Entrada (json)
    body = json.loads(event['Records'][0]['body'])
    Message = json.loads(body['Message'])
    # Proceso
    dynamodb = boto3.resource('dynamodb')
    table = dynamodb.Table('usuario')
    user = {
        'tenant_id': Message['tenant_id'],
        'tarjeta_id': Message['tarjeta_id'],
        'dni': Message['dni'],
        'tipo': Message['tipo'],
        'cuenta': Message['cuenta'],
        'saldo' : 0
    }
    print(user) # Revisar en CloudWatch
    response = table.put_item(Item=user)
    # Salida (json)
    return {
    'statusCode': 200,
    'response': response
    }

```

En base a los datos obtenidos, crea un diccionario con toda la información del usuario y la añade a la tabla [usuario]

SQS: Cola Crear Cuenta



Subscripción SNS

Subscription ARN	Topic ARN
arn:aws:sns:us-east-1:021662273041:TemaNuevaCuenta:fa43ffad-5859-4c9f-acb8-ee432d5c57b0	arn:aws:sns:us-east-1:021662273041:TemaNuevaCuenta

Lambda Triggers

UUID	ARN	Status	Last modified
bc35732c-45cf-4e5c-9e98-bbc5081853e4	arn:aws:lambda:us-east-1:021662273041:function:CrearCuenta	Enabled	11/15/2022, 6:33:30 PM

SQS: ColaCrearCuenta

```
"Version": "2008-10-17",
"Statement": [
  {
    "Effect": "Allow",
    "Principal": {
      "Service": "sns.amazonaws.com"
    },
    "Action": "sqs:SendMessage",
    "Resource": "arn:aws:sqs:us-east-1:021662273041:ColaCrearCuenta",
    "Condition": {
      "ArnEquals": {
        "aws:SourceArn": "arn:aws:sns:us-east-1:021662273041:TemaNuevaCuenta"
      }
    }
  },
  {
    "Sid": "topic-subscription-arn:aws:sns:us-east-1:021662273041:TemaNuevaCuenta",
    "Effect": "Allow",
    "Principal": {
      "AWS": "*"
    },
    "Action": "SQS:SendMessage",
    "Resource": "arn:aws:sqs:us-east-1:021662273041:ColaCrearCuenta",
    "Condition": {
      "ArnLike": {
        "aws:SourceArn": "arn:aws:sns:us-east-1:021662273041:TemaNuevaCuenta"
      }
    }
  }
]
```

Políticas de acceso

SNS: TemaNuevaCuenta

Details

Name
TemaNuevaCuenta

ARN
arn:aws:sns:us-east-1:021662273041:TemaNuevaCuenta

Type
Standard

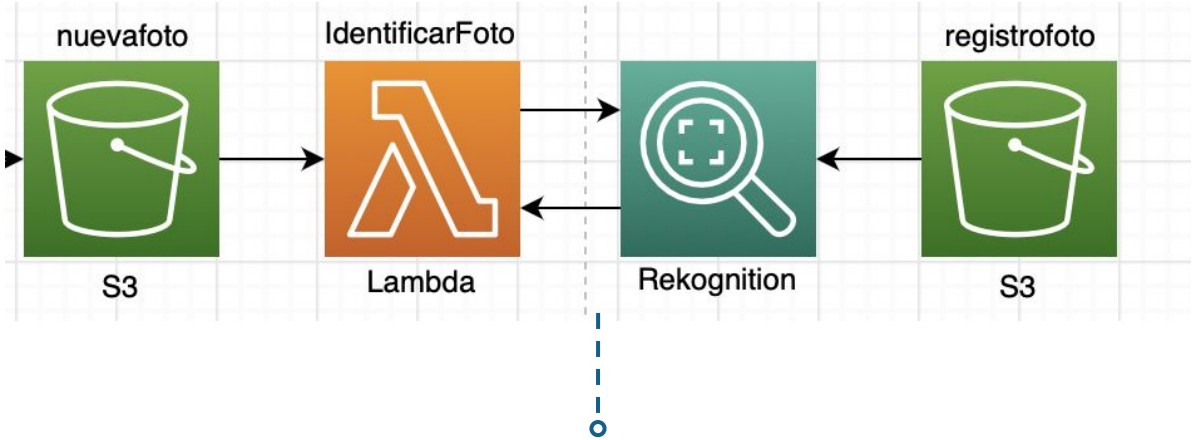
Display name
-

Topic owner
021662273041

Suscripciones

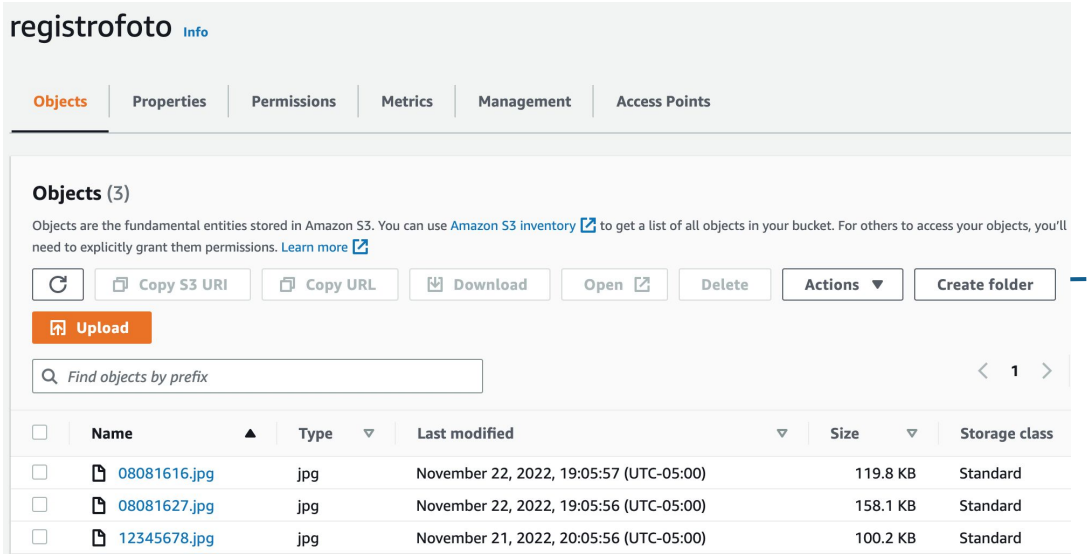
ID	Endpoint	Status	Protocol
2c233248-4d7c-4580-9d54-…	sofigqv@gmail.com	Confirmed	EMAIL
fa43ffad-5859-4c9f-acb8-ee…	arn:aws:sqs:us-east-1:021662…	Confirmed	SQS

Identificar Foto



El usuario sube una foto al crear una cuenta, y usando Rekognition es comparado con la foto de dicha persona en la tabla Registro.

S3: RegistroFoto



Fotos relacionadas a la tabla [registro]

Lambda: IdentificarFoto

```python
import json
import boto3

def getdni(foto_uri):
    flag = False
    dni = ""
    for c in foto_uri:
        if flag:
            dni += c
        if c == "_":
            flag = True

    return dni

def gettenant_id(foto_uri):
    tenant_id = ""
    for c in foto_uri:
        if c == "_":
            break
        tenant_id += c

    return tenant_id

def comparar_rostros(foto_uri):
    dnijpg = getdni(foto_uri)
    tenant_id = gettenant_id(foto_uri)
    client = boto3.client('rekognition')
    response = client.compare_faces(
        SourceImage={
            'S3Object': {
                'Bucket': 'nuevafoto',
                'Name': foto_uri
                }
            },
        TargetImage={
            'S3Object': {
                'Bucket': 'registrofoto',
                'Name': dnijpg
                }
            }

    )
```

Obtiene el dni de la foto

Obtiene el tenant_id

Rekognition compara las dos fotos

Lambda: IdentificarFoto

```
    res = ""
    dynamodb = boto3.resource('dynamodb')
    table = dynamodb.Table('proceso')

    for record in response['FaceMatches']:
        face = record
        confidence=face['Face']
        if confidence['Confidence'] >= 95:
            updated = table.update_item(
                Key={
                    'tenant_id': tenant_id,
                    'dni': dnijpg[:-4],
                },
                UpdateExpression="set identificarfoto=:identificarfoto",
                ExpressionAttributeValues={
                    ':identificarfoto': 'Aprobado' #pasar fase 1
                },
                ReturnValues="UPDATED_NEW"
            )
            res = "Aprobado."

        #print ("Matched With {}""%"" Similarity".format(face['Similarity']))
        #print ("With {}""%"" Confidence".format(confidence['Confidence']))

    for record in response['UnmatchedFaces']:
        updated = table.update_item(
            Key={
                'tenant_id': tenant_id,
                'dni': dnijpg[:-4],
            },
            UpdateExpression="set identificarfoto=:identificarfoto",
            ExpressionAttributeValues={
                ':identificarfoto': 'Denegado' #pasar fase 1
            },
            ReturnValues="UPDATED_NEW"
        )
        res = "Denegado."
        #print ("No matched with {}""%"" Confidence".format(record['Confidence']))
    #print(response)

    return res
```

Si las fotos coinciden se aprueba la creación de la cuenta

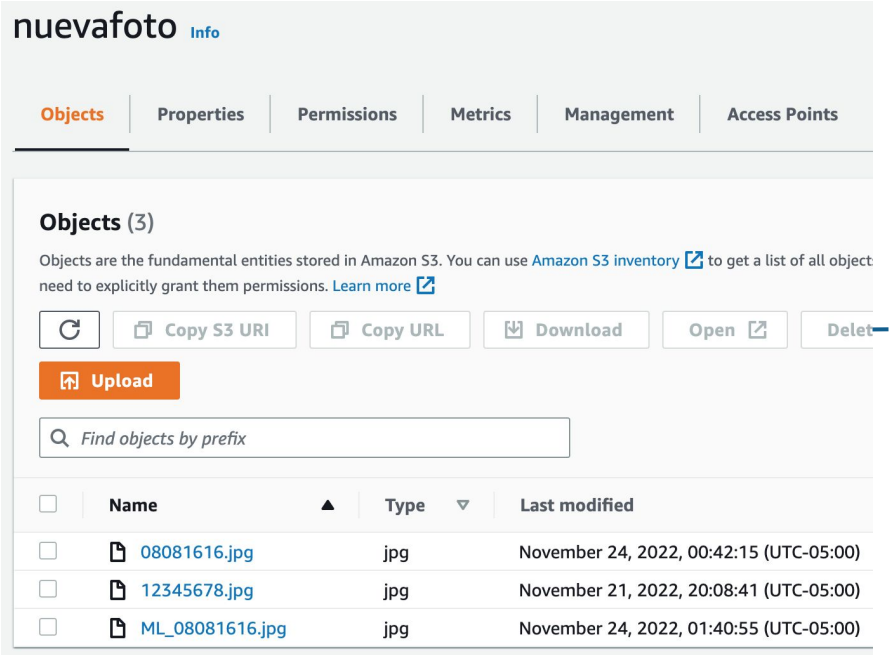
Si las fotos no coinciden no se admite la creación de la cuenta

Lambda: IdentificarFoto

```python
def lambda_handler(event, context):
    # Entrada (json), revisar en CloudWatch
    foto_uri = event['Records'][0]['s3']['object']['key']
    res = comparar_rostros(foto_uri)
    print(res)

    return {
        'statusCode': 200,
        'body': res
    }
```

Hacer uso de la función comparar_rostros en base a la foto_uri

S3: nuevafoto



Fotos subidas por el usuario

Lambda: NuevaCuenta

```python
from __future__ import print_function
import json
import boto3
import time, urllib
import uuid
from boto3.dynamodb.conditions import Key

def subirfoto(tenant_id, dni, image_name):
    s3 = boto3.resource('s3')
    copy_source = {
        'Bucket': 'localfoto',
        'Key': image_name
    }
    name = tenant_id + '_' + dni + '.jpg'
    response = s3.meta.client.copy(copy_source, 'nuevafoto', name)

def revisarcuenta(usuario):
    sns_client = boto3.client('sns')
    response_sns = sns_client.publish(
        TopicArn = 'arn:aws:sns:us-east-1:021662273041:TemaRevisarCuenta',
        Subject = 'Revisión de cuenta',
        Message = json.dumps(usuario),
        MessageAttributes = {
            'tenant_id': {'DataType': 'String', 'StringValue': usuario['tenant_id'] }
        }
    )

def crearcuenta(usuario):
    sns_client = boto3.client('sns')
    response_sns = sns_client.publish(
        TopicArn = 'arn:aws:sns:us-east-1:021662273041:TemaNuevaCuenta',
        Subject = 'Nueva Cuenta',
        Message = json.dumps(usuario),
        MessageAttributes = {
        'tenant_id': {'DataType': 'String', 'StringValue': usuario['tenant_id'] }
        }
    )
```

Rekognition

Conexión al SNS TemaRevisarCuenta

Conexión al SNS TemaNuevaCuenta

Lambda: NuevaCuenta

```python
def query_proceso(table, tenant_id, dni):
    response = table.query(
        KeyConditionExpression=Key('tenant_id').eq(tenant_id)
    )
    items = response['Items']

    for p in items:
        if p['dni'] == dni:
            return p

def lambda_handler(event, context):
    tenant_id = event['tenant_id']
    tarjeta_id = str(uuid.uuid1())
    tipo = event['tipo']
    dni = event['dni']
    cuenta = event['cuenta']
    saldo = 0
    image_name = event['image_name']

    usuario = {
        'tenant_id': tenant_id,
        'tarjeta_id': tarjeta_id,
        'tipo': tipo,
        'dni': dni,
        'cuenta': cuenta,
        'saldo': saldo
    }
```

Función de para test

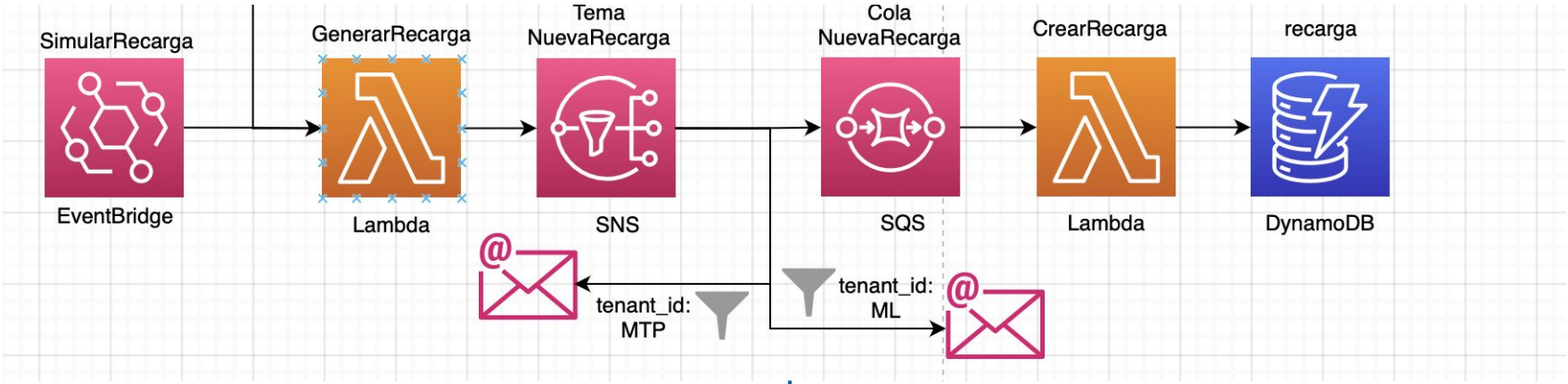
Obtiene información relevante para el usuario

Lambda: NuevaCuenta

```
67      #Crear proceso
68      dynamodb = boto3.resource('dynamodb')
69      table = dynamodb.Table('proceso')
70      proceso_id = str(uuid.uuid1())
71      proceso = {
72          'tenant_id': tenant_id,
73          'dni': dni,
74          'identificarfoto': "-",
75          'revisarcuenta': "-"
76      }
77      table.put_item(Item=proceso)
78
79      subirfoto(tenant_id, dni, image_name) #rekognition, comparar fotos
80      revisarcuenta(usuario)
81      crearcuenta(usuario)
82
83      return {
84          'statusCode': 200,
85          'body': usuario
86      }
```

Realiza las funciones en orden

Nueva Recarga



Se simulan recargas aleatorias de usuario en la tabla Usuario, las cuales son registradas en la tabla Recarga

DynamoDB: recargas_t

Información general

Clave de partición	Clave de ordenación	Modo de capacidad	Estado de la tabla
tenant_id (String)	recarga_id (String)	Bajo demanda	Activo No hay alarmas activas

tenant_id	recarga_id	recarga_datos
ML	112870430	{ "tarjeta_id" : { "S" : "2345678901" }, "fecha_hora" : { "S" : "2022-11-23.00:02:34.349296" }, "monto" : { "N" : "21" } }
ML	125862785	{ "tarjeta_id" : { "S" : "3456789012" }, "fecha_hora" : { "S" : "2022-11-22.23:17:34.979241" }, "monto" : { "N" : "29" } }
ML	144915329	{ "tarjeta_id" : { "S" : "1234567890" }, "fecha_hora" : { "S" : "2022-11-22.23:56:34.370317" }, "monto" : { "N" : "28" } }
ML	14921626	{ "tarjeta_id" : { "S" : "2345678901" }, "fecha_hora" : { "S" : "2022-11-22.23:49:34.204962" }, "monto" : { "N" : "29" } }
ML	150585955	{ "tarjeta_id" : { "S" : "2345678901" }, "fecha_hora" : { "S" : "2022-11-22.23:15:34.597011" }, "monto" : { "N" : "14" } }
ML	166559572	{ "tarjeta_id" : { "S" : "3456789012" }, "fecha_hora" : { "S" : "2022-11-22.23:36:34.110522" }, "monto" : { "N" : "9" } }

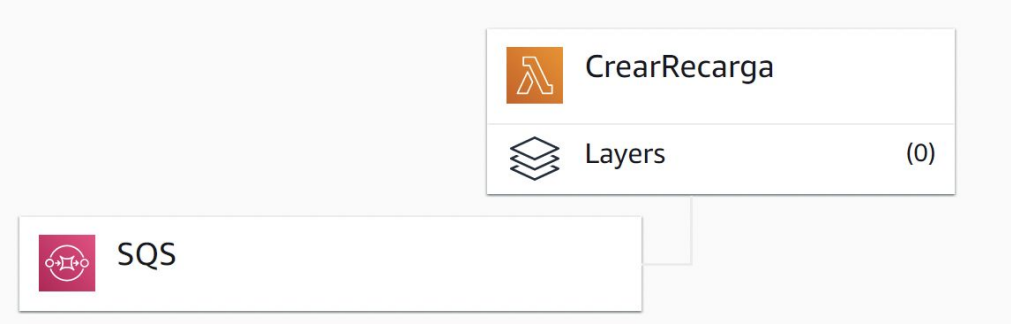
Lambda: CrearRecarga

lambda_function × Execution results ×

```python
import json
import boto3

def lambda_handler(event, context):
    # Entrada (json)
    print(event) # Revisar en CloudWatch
    body = json.loads(event['Records'][0]['body'])
    recarga_json = json.loads(body['Message'])
    # Proceso
    dynamodb = boto3.resource('dynamodb')
    table = dynamodb.Table('recargas_t')

    #print(recarga) -> Revisar en CloudWatch
    response = table.put_item(Item=recarga_json)

    # Salida (json)
    return {
        'statusCode': 200,
        'response': response
    }
```

Ingresa la información de la recarga a la tabla [recargas_t]

SQS: ColaNuevaRecarga



Subscripción SNS

Subscription ARN	Topic ARN
arn:aws:sns:us-east-1:167952863946:TemaNuevaRecarga:951cd008-4c76-482f-972e-980822e4afa8	arn:aws:sns:us-east-1:167952863946:TemaNuevaRecarga

Lambda Triggers

UUID	ARN	Status	Last modified
f86ba293-b07b-4cab-b3f7-eb994fdb0715	arn:aws:lambda:us-east-1:021662273041:function:CrearRecarga	Enabled	11/22/2022, 1:02:21 PM

SQS: ColaNuevaRecarga

```
"Version": "2008-10-17",
"Statement": [
  {
    "Effect": "Allow",
    "Principal": {
      "Service": "sns.amazonaws.com"
    },
    "Action": "sqs:SendMessage",
    "Resource": "arn:aws:sqs:us-east-1:021662273041:ColaNuevaRecarga",
    "Condition": {
      "ArnEquals": {
        "aws:SourceArn": "arn:aws:sns:us-east-1:021662273041:TemaNuevaRecarga"
      }
    }
  },
  {
    "Sid": "topic-subscription-arn:aws:sns:us-east-1:021662273041:TemaNuevaRecarga",
    "Effect": "Allow",
    "Principal": {
      "AWS": "*"
    },
    "Action": "SQS:SendMessage",
    "Resource": "arn:aws:sqs:us-east-1:021662273041:ColaNuevaRecarga",
    "Condition": {
      "ArnLike": {
        "aws:SourceArn": "arn:aws:sns:us-east-1:021662273041:TemaNuevaRecarga"
      }
    }
  }
]
```

Políticas de acceso

SNS: TemaNuevaRecarga

Details

Name TemaNuevaRecarga	Display name -
ARN arn:aws:sns:us-east-1:021662273041:TemaNuevaRecarga	Topic owner 021662273041
Type Standard	

Suscripciones

ID	Endpoint	Status	Protocol
951cd008-4c76-482f-972e-9…	arn:aws:sqs:us-east-1:167952…	Confirmed	SQS
db9a96b1-41ba-4ae7-ac45-c…	valeriaet2304@gmail.com	Confirmed	EMAIL

Lambda: GenerarRecarga

```python
import json
import random
from datetime import datetime
import boto3
from boto3.dynamodb.conditions import Key

def lambda_handler(event, context):
    # Entrada (json)
    # Funciona con EventBridge
    dynamodb = boto3.resource('dynamodb')
    recargas = dynamodb.Table('usuario')
    response_r = recargas.scan()
    items_r = response_r['Items']
    num_reg = response_r['Count'] - 1

    count = 0
    tarjeta_number = random.randint(0, num_reg)
    for item in items_r:
        if count == tarjeta_number:
            tenant_id = item['tenant_id']
            tarjeta_id = str(item['tarjeta_id'])
        count += 1

    recarga_id = str(random.randint(1, 999999999))
    monto = random.randint(1, 30)
    #monto = event['monto']
    now = datetime.now()
    fecha_hora = str(now.date()) + "." + str(now.time())
    recarga = {
        'tenant_id': tenant_id,
        'recarga_id': recarga_id,
        'recarga_datos':{
            'tarjeta_id': tarjeta_id,
            'fecha_hora': fecha_hora,
            'monto': monto
        }
    }
```

Obtiene la data de una tarjeta aleatoriamente

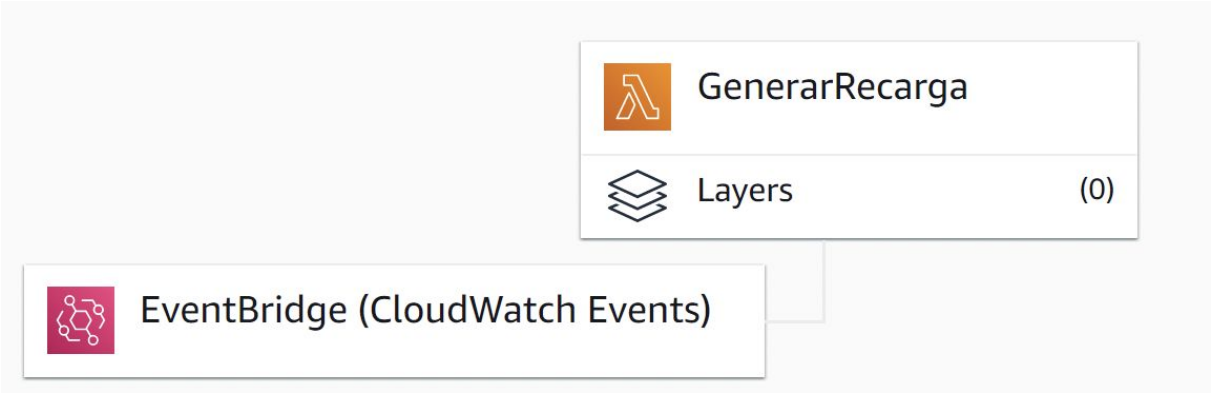
Genera data aleatoria para un nuevo registro de recarga

Lambda: GenerarRecarga

```python
    # Publicar en SNS
    sns_client = boto3.client('sns')
    response_sns = sns_client.publish(
        TopicArn = 'arn:aws:sns:us-east-1:021662273041:TemaNuevaRecarga',
        Subject = 'Nueva Recarga',
        Message = json.dumps(recarga),
        MessageAttributes = {
            'tenant_id': {'DataType': 'String', 'StringValue': tenant_id },
            'recarga_id': {'DataType': 'String', 'StringValue': recarga_id },
            'tarjeta_id': {'DataType': 'String', 'StringValue': tarjeta_id },
            'monto': {'DataType': 'Number', 'StringValue': str(monto) }
        }
    )
    print(response_sns)
    # Salida (json)
    return {
        'statusCode': 200,
        'response': response_sns
    }
```

Se publica en SNS los datos de la nueva recarga ejecutada.

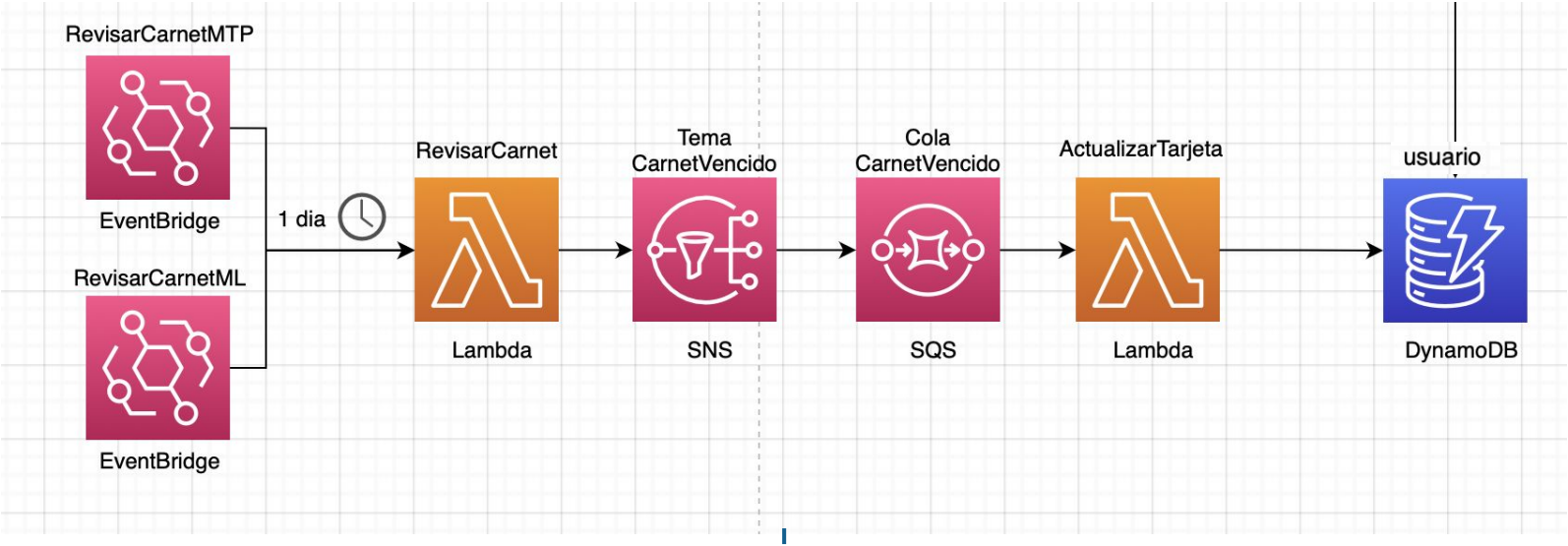
Event Bridge: SimularRecarga



Targets

Target Name	Type	Arn	Input
GenerarRecarga	Lambda function	arn:aws:lambda:us-east-1:167952863946:function:GenerarRecarga	Matched event

Revisar Carnet



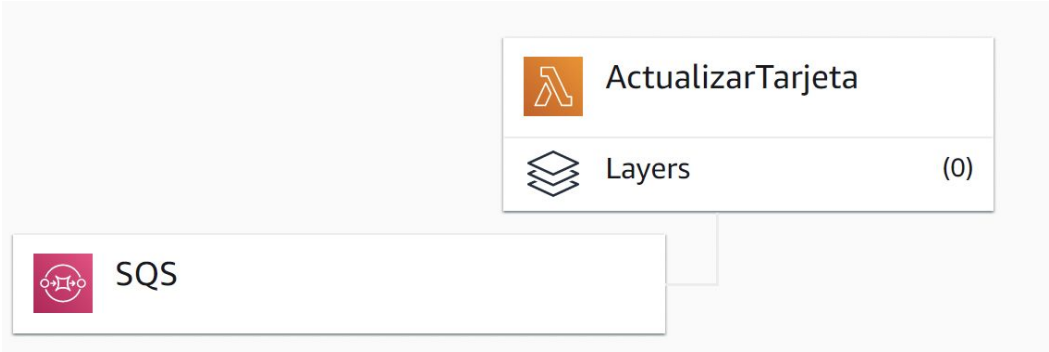
Diariamente, se hace una revisión de las tarjetas de tipo universitario. Si la fecha actual es mayor a la fecha de vencimiento de dicha tarjeta, entonces se debe actualizar su tipo en la tabla Usuario

Lambda: ActualizarTarjeta

```python
import json
import boto3

def lambda_handler(event, context):
    # Entrada (json)
    body = json.loads(event['Records'][0]['body'])
    key_to_update = json.loads(body['Message'])

    # Proceso
    dynamodb = boto3.resource('dynamodb')
    table = dynamodb.Table('usuario')

    response = table.update_item(
        Key={
            'tenant_id': key_to_update['tenant_id'],
            'tarjeta_id': int(key_to_update['tarjeta_id']),
        },
        UpdateExpression="set tipo=:tipo",
        ExpressionAttributeValues={
            ':tipo': 'general'
        },
        ReturnValues="UPDATED_NEW"
    )

    # Salida (json)
    return {
        'statusCode': 200,
        'response': response
    }
```

Actualiza el tipo de tarjeta a “general” en base al usuario que se obtiene.

SQS: ColaCarnetVencido



Subscripción SNS

Subscription ARN	Topic ARN
arn:aws:sns:us-east-1:021662273041:TemaCarnetVencido:185a0666-ff48-437d-83c5-3ee45d926fde	arn:aws:sns:us-east-1:021662273041:TemaCarnetVencido

Lambda Triggers

UUID	ARN	Status	Last modified
7e49a815-de79-4bc2-8426-15c71f713a71	arn:aws:lambda:us-east-1:021662273041:function:ActualizarTarjeta	Enabled	11/20/2022, 9:14:04 PM

SQS: ColaCarnetVencido

```
    {
      "Sid": "topic-subscription-arn:aws:sns:us-east-1:021662273041:TemaCarnetVencido",
      "Effect": "Allow",
      "Principal": {
        "AWS": "*"
      },
      "Action": "SQS:SendMessage",
      "Resource": "arn:aws:sqs:us-east-1:021662273041:ColaCarnetVencido",
      "Condition": {
        "ArnLike": {
          "aws:SourceArn": "arn:aws:sns:us-east-1:021662273041:TemaCarnetVencido"
        }
      }
    }
```

SNS: TemaCarnetVencido

Details

Name
TemaCarnetVencido

ARN
arn:aws:sns:us-east-1:021662273041:TemaCarnetVencido

Type
Standard

Display name
-

Topic owner
021662273041

Suscripciones

ID	Endpoint	Status	Protocol
185a0666-ff48-437d-83c5-3…	arn:aws:sqs:us-east-1:021662…	Confirmed	SQS
a8fa7b5b-c806-44b0-8cd1-d…	sofigqv@gmail.com	Confirmed	EMAIL

Lambda: RevisarCarnet

```python
import json
import random
from datetime import datetime, timedelta
import boto3
from boto3.dynamodb.conditions import Key

def check(tenant_id, tarjeta_id, fecha_ven, hoy):
    # TODO implement
    key_to_update = {
        'tenant_id': tenant_id,
        'tarjeta_id': int(tarjeta_id)
    }
    # Publicar en SNS
    sns_client = boto3.client('sns')
    response_sns = sns_client.publish(
        TopicArn = 'arn:aws:sns:us-east-1:021662273041:TemaCarnetVencido',
        Subject = 'Check Date',
        Message = json.dumps(key_to_update),
        MessageAttributes = {
            'tenant_id': {'DataType': 'String', 'StringValue': tenant_id },
            'tarjeta_id': {'DataType': 'Number', 'StringValue': tarjeta_id }
        }
    )

    return key_to_update
```

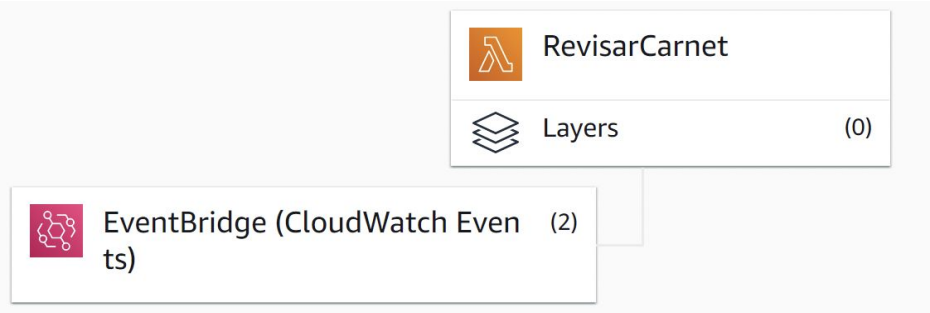
Función para publicar en SNS (correo) en base al id de la tarjeta.

Lambda: RevisarCarnet

```python
def lambda_handler(event, context):
    # Entrada (json)
    tenant_id = event['tenant_id']
    # Proceso
    dynamodb = boto3.resource('dynamodb')
    usuario = dynamodb.Table('usuario')
    response_u = usuario.query(
        KeyConditionExpression=Key('tenant_id').eq(tenant_id)
    )
    items_u = response_u['Items']

    registros = dynamodb.Table('registro')
    response_r = registros.query(
        KeyConditionExpression=Key('tenant_id').eq('SUNEDU')
    )
    items_r = response_r['Items']

    fecha_ven = ''
    items = []

    for user in items_u:
        if user['tipo'] == "universitario":
            for reg in items_r: #que coincidan fecha y usuarios
                if user['dni'] == reg['dni']:
                    fecha_ven = reg['fecha_ven']
                    now = datetime.now()
                    now = now - timedelta(hours=5) #UTC-5
                    hoy = str(now.date())
                    if hoy == fecha_ven: #si vence hoy
                        key_to_update = check(tenant_id, str(user['tarjeta_id']), fecha_ven, hoy)
                        items.append(key_to_update)

    # Salida (json)
    return {
        'statusCode': 200,
        'body': items
    }
```

Se verifica que para cada usuario que su tarjeta caduque el día actual, se mande un SNS y se retorne todas aquellas que cumplan esa condición.

Event Bridge: RevisarCarnetMTP



EventBridge (CloudWatch Events): RevisarCarnetMTP

arn:aws:events:us-east-1:021662273041:rule/RevisarCarnetMTP

▼ Details

Event bus: **default**

name: **RevisarCarnetMTP**

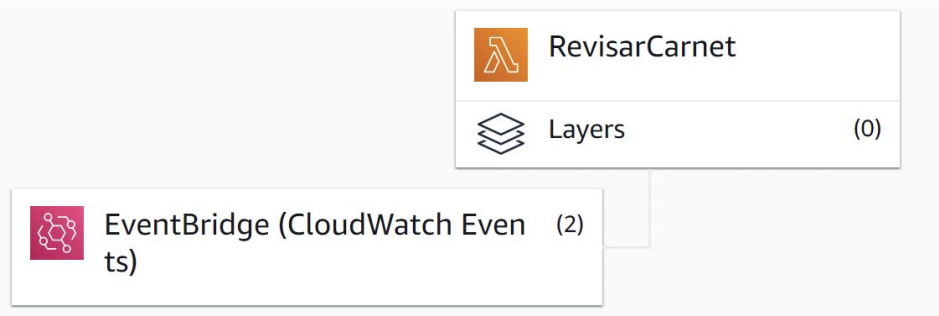
Schedule expression: **rate(1 minute)**

Service principal: **events.amazonaws.com**

Statement ID: **AWSEvents_RevisarCarnetMTP_Id64ab179f-443d-4ec5-b5ed-2aedee0f56d6**

url: **events/home#/rules/RevisarCarnetMTP**

Event Bridge: RevisarCarnetML



EventBridge (CloudWatch Events): RevisarCarnetML

arn:aws:events:us-east-1:021662273041:rule/RevisarCarnetML

▼ Details

Event bus: **default**

name: **RevisarCarnetML**

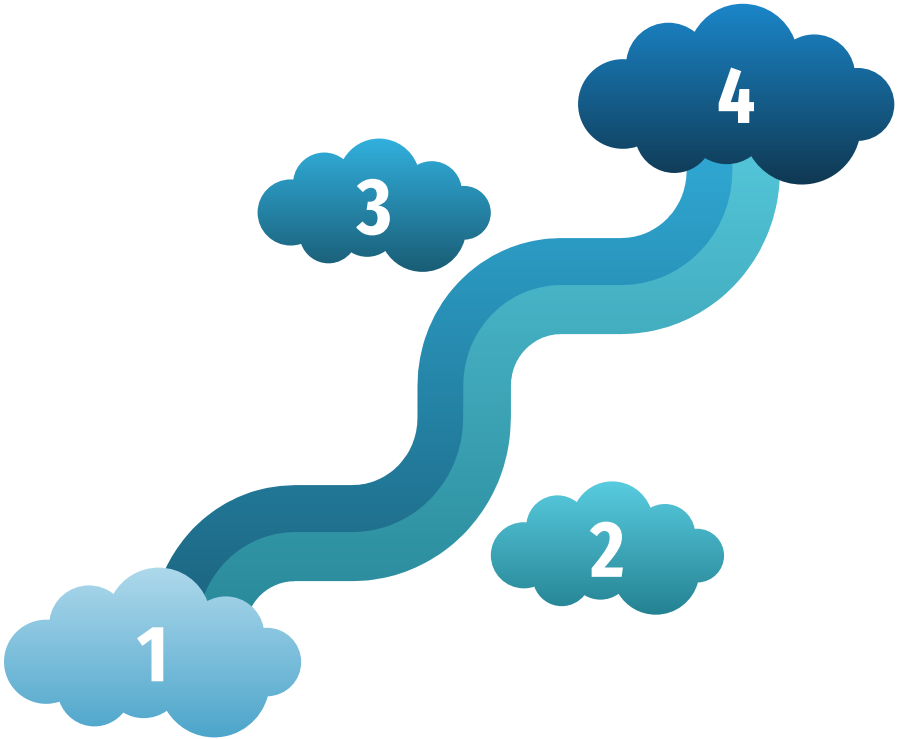
Schedule expression: **rate(1 minute)**

Service principal: **events.amazonaws.com**

Statement ID: **AWSEvents_RevisarCarnetML_Id9596c56d-0b68-4e6d-88d9-c360d6d17d49**

url: **events/home#/rules/RevisarCarnetML**

Resultados



1. Se cumplieron con los requisitos exigidos y con el servicio opcional (Rekognition)
2. Diseño de una arquitectura que optimizaría en gran medida el caótico sistema actual del metropolitano
3. El proyecto puede ser extendido a otros medios de transportes (Ejem. futuro tren de Santa anita - Callao)