



Entrega 2: Migración al Cloud de SolidarianId

Desarrollo de Software en la Nube

Máster Universitario en Ingeniería del Software

Autores:

Hernán Salambay Roldán
Pedro Nicolás Gomariz
Aurora Hervás López
Dongyue Yu

23 de Mayo de 2025



Facultad
de Informática
UMU

Contenidos

1. Introducción.	3
2. Descripción de la Implementación.	3
2.1. Funcionalidad: Registro de apoyos anónimos a causas.	3
2.1.1. Uso de Servicios Cloud.	4
2.1.2. Activación/Desactivación del Cloud.	7
2.1.3. Ejemplo de funcionamiento.	7
2.2. Funcionalidad: Almacenamiento de datos en DynamoDB.	8
2.2.1. Componentes principales y tecnologías usadas.	8
2.2.2. Explicación del funcionamiento general del sistema.	9
2.2.3. Nivel de desarrollo alcanzado.	10
2.2.4. Uso de Servicios Cloud.	11
2.2.5. Activación/Desactivación del Cloud.	11
2.3. Funcionalidad: Generación automática de avatares al registrar usuarios.	12
2.3.1. Uso de Servicios Cloud.	13
2.3.2. Activación/Desactivación del Cloud.	14
3. Conclusión.	15
4. Bibliografía.	15

1. Introducción.

En esta práctica hemos desarrollado un prototipo que incluye la integración de varios servicios en la nube como parte de la plataforma SolidarianId. El punto de partida ha sido la propuesta presentada en la Entrega 1. Aunque hemos intentado mantener la misma línea, ha sido necesario ajustar algunos aspectos técnicos durante el desarrollo. Estos cambios se explican y justifican a lo largo del documento.

A lo largo de las siguientes secciones se describe cómo se ha llevado a cabo la implementación, qué servicios cloud se han utilizado y cómo se ha resuelto el control para habilitarlos o no, según se desee.

2. Descripción de la Implementación.

En este apartado se describen las funcionalidades implementadas, junto con los servicios de AWS utilizados. Aunque inicialmente se planteó la implementación de una funcionalidad de login, esta se descartó debido a la complejidad de las relaciones internas en el microservicio de usuarios.

En su lugar, se han desarrollado nuevas funcionalidades que hemos incorporado a la aplicación SolidarianId: el registro de apoyos anónimos a causas y la generación automática de avatares al registrar usuarios. Aunque no estaban indicadas explícitamente en la entrega 1, se han considerado relevantes para utilizar servicios cloud, enriquecer la experiencia del usuario y aportar valor funcional a la aplicación.

El apoyo a causas se gestiona mediante API Gateway y funciones Lambda que validan y almacenan la información en DynamoDB, evitando duplicados. Por su parte, la generación de avatares utiliza la API de DiceBear y almacena los archivos en Amazon S3, permitiendo que el frontend los muestre en el perfil del usuario. Finalmente, se ha realizado una migración parcial de los datos relacionados con causas desde MongoDB hacia DynamoDB, con el objetivo de unificar la persistencia en la nube y aprovechar las ventajas del ecosistema serverless de AWS.

2.1. Funcionalidad: Registro de apoyos anónimos a causas.

En este apartado se describe la solución implementada para incorporar la funcionalidad de registro de apoyos anónimos a causas solidarias, la cual no estaba implementada previamente en el sistema, aunque sí figuraba como una posible mejora recogida en el documento de requisitos futuros. Esta extensión amplía el alcance del proyecto al permitir la participación de usuarios no autenticados, facilitando una vía de colaboración más abierta e inclusiva.

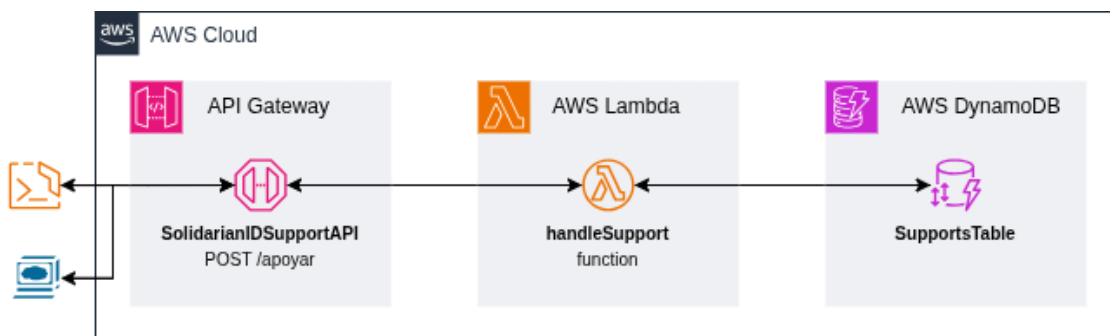
La funcionalidad desarrollada permite que cualquier usuario pueda apoyar públicamente una causa a través de un formulario accesible desde la interfaz pública. Para ello, el usuario debe introducir su nombre y email. Una vez enviado, el apoyo queda registrado de forma persistente en la base de datos, y el sistema garantiza que cada dirección de correo electrónico solo pueda registrar un apoyo por causa, evitando así duplicidades.

Flujo general de la funcionalidad:

1. El usuario realiza una solicitud **POST** al endpoint **/apoyar** desde la interfaz pública.
2. La solicitud es recibida por un endpoint expuesto en un **API Gateway**.
3. API Gateway **invoca una función Lambda** que contiene la lógica de validación y persistencia.

4. La función consulta y escribe en **DynamoDB**, donde se guarda el apoyo si no existía previamente para ese usuario y causa.
5. Se responde al usuario con un mensaje de éxito o error (si ya había apoyado antes).

A continuación se muestra el diagrama de arquitectura que representa gráficamente la solución desplegada para esta funcionalidad:



2.1.1. Uso de Servicios Cloud.

Para realizar el despliegue de los distintos servicios se han usado scripts en Python usando Boto3 así como la consola web de AWS.

- **Amazon DynamoDB:** Se ha empleado para almacenar de forma persistente los apoyos recibidos. La tabla utilizada se denomina *SupportsTable* y está diseñada para evitar duplicados de apoyo por parte de un mismo usuario. Para ello, se ha definido como clave de partición el campo *causeId* y como clave de ordenación el campo *email*. Este esquema garantiza que cada usuario, identificado por su correo electrónico, solo pueda apoyar una misma causa una única vez.

```

aws > create_table.py
1 import boto3
2
3 def create_table():
4     DDB = boto3.resource('dynamodb', region_name='us-east-1')
5
6     params = {
7         'TableName': 'SupportsTable',
8         'KeySchema': [
9             {'AttributeName': 'causeId', 'KeyType': 'HASH'},
10            {'AttributeName': 'email', 'KeyType': 'RANGE'}
11        ],
12        'AttributeDefinitions': [
13            {'AttributeName': 'causeId', 'AttributeType': 'S'},
14            {'AttributeName': 'email', 'AttributeType': 'S'}
15        ],
16        'ProvisionedThroughput': {
17            'ReadCapacityUnits': 1,
18            'WriteCapacityUnits': 1
19        }
20    }
21
22    table = DDB.create_table(**params)
23    table.wait_until_exists()
24    print("SupportsTable table created with composite key")
25
26 if __name__ == '__main__':
27     create_table()
  
```

SupportsTable



 Acciones 
 Explore los elementos de la tabla

[Configuración](#) |
 [Índices](#) |
 [Monitorear](#) |
 [Tablas globales](#) |
 [Copias de seguridad](#) |
 [Exportaciones y flujos](#) |
 [Permisos](#)

Proteja su tabla de DynamoDB frente a escrituras y eliminaciones accidentales

Al activar la recuperación a un momento dado (PITR), DynamoDB realiza copias de seguridad de los datos de la tabla automáticamente para que pueda restaurarlas a cualquier segundo en los días 1 a 35 anteriores. Se aplican cargos adicionales. [Más información](#)

[Editar PITR](#)



Información general [Información](#)


[Obtener el recuento de elementos en directo](#)

Clave de partición
causedId (String)

Clave de ordenación
email (String)

Modo de capacidad
Aprovisionado

Estado de la tabla
 Activo

Alarmas
 No hay alarmas activas

Recuperación a un momento
dado [Información](#)


Recuento de elementos
0

Tamaño de la tabla
0 bytes

Tamaño medio de elemento
0 bytes

Política basada en recursos [Información](#)

Nombre de recurso de Amazon (ARN)

 arn:aws:dynamodb:us-east-1:834872315749:table/SupportsTable

- AWS Lambda:** se encarga de ejecutar la lógica central de la funcionalidad a través de la función *handleSupport*. Esta función tiene como responsabilidades validar los datos recibidos en la solicitud (*nombre*, *email* y *causedId*), consultar en DynamoDB si ya existe un apoyo registrado con ese email para la causa indicada, registrar el apoyo en la base de datos si no existe, y devolver una respuesta indicando el estado de la operación (*200 OK* o *409 Conflict*).

La función se ha creado mediante la consola de AWS Lambda, utilizando Python como entorno de ejecución. Se ha configurado la variable de entorno *SUPPORTS_TABLE*, y se han asignado permisos explícitos sobre DynamoDB mediante la política *AmazonDynamoDBFullAccess* vinculada al rol de ejecución. Además, se ha agregado una política de invocación externa para permitir su integración con API Gateway.

handleSupport

[Limitación](#) |
 [Copiar ARN](#) |
 Acciones 

▼ Información general de la función [Información](#)

[Exportar a Infrastructure Composer](#) |
 [Descargar](#) 

[Diagrama](#) |
 [Plantilla](#)


handleSupport
 Layers (0)

[+ Agregar desencadenador](#)

[+ Agregar destino](#)

Descripción

-

Última modificación

hace 1 hora

ARN de la función

 arn:aws:lambda:us-east-1:834872315749:function:handleSupport


URL de la función

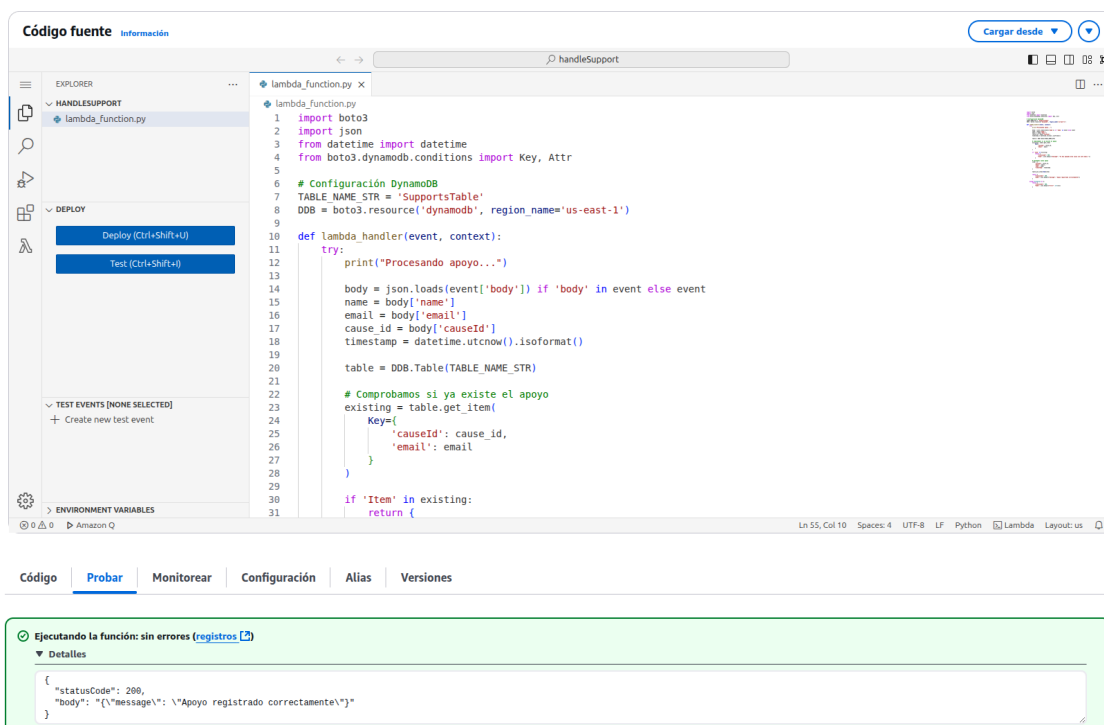
-

Instrucciones de la política basada en recursos (1) [Información](#)


[Ver la política](#) |
 [Editar](#) |
 [Eliminar](#) |
 [Agregar permisos](#)

Las políticas basadas en recursos otorgan permisos a otras cuentas y servicios de AWS para acceder a sus recursos de Lambda.

ID de instrucción	Entidad principal	PrincipalOrgID	Condiciones	Acción
 apigateway-invoke-support	apigateway.amazonaws.com	-	ArnLike	lambda:InvokeFunction



- Amazon API Gateway:** expone el endpoint *POST /apoyar* como punto de entrada público. Creado mediante un script en Python con Boto3, que automatiza la creación del recurso, el método POST y su integración con la función Lambda correspondiente. El endpoint se configura sin autenticación, permitiendo el acceso anónimo. Finalmente, se desplegó en la etapa *prod*, generando una URL pública accesible.

```

aws > create_api_gateway.py
1  import boto3, json
2
3  ACCOUNT_ID = '834872315749'
4  LAMBDA_FUNCTION_NAME = 'handleSupport'
5
6  client = boto3.client('apigateway', region_name='us-east-1')
7  lambda_client = boto3.client('lambda', region_name='us-east-1')
8
9  # Create the REST API
10 response = client.create_rest_api(
11     name='SolidarianIDSupportAPI',
12     description='API to register anonymous support for solidarity causes',
13     minimumCompressionSize=123,
14     endpointConfiguration={
15         'types': ['REGIONAL'],
16     }
17 )
18 api_id = response["id"]
19
20 # Get the root ID "/"
21 resources = client.get_resources(restApiId=api_id)
22 root_id = [resource for resource in resources["items"] if resource["path"] == "/"][0]["id"]
23
24 # Create the /apoyar resource
25 apoyar_resource = client.create_resource(
26     restApiId=api_id,
27     parentId=root_id,
28     pathPart='apoyar'
29 )
30 apoyar_resource_id = apoyar_resource["id"]
  
```

Recursos

Acciones de API Implementar API

Crear recurso

/ /apoyar POST

/apoyar - POST - Ejecución de método

ARN: `arn:aws:execute-api:us-east-1:834872315749:8pyzq9zkd/*/*/*POST/apoyar` ID de recurso: `ktp8pd`

Actualizar documentación Eliminar

Diagrama de flujo:

```

    graph LR
      Cliente --> SolicitudM[Solicitud de método]
      SolicitudM --> SolicitudInt[Solicitud de integración]
      SolicitudInt --> IntegracionLambda[Integración de Lambda  
Función: handleSupport]
      IntegracionLambda --> RespuestaInt[Respuesta de integración  
Integración de proxy]
      RespuestaInt --> RespuestaM[Respuesta de método]
      RespuestaM --> Cliente
  
```

Solicitud de método | Solicitud de integración | Respuesta de integración | Respuesta de método | Pruebas

Configuración de solicitud de método

Autorización: NONE | Clave de API obligatoria: Falso | Validador de solicitudes: Ninguna | Nombre de la operación del SDK: Generado según el método y la ruta

Configuración de solicitud de integración

Tipo de integración: Lambda | Región: us-east-1 | Integración de proxy de Lambda: Verdadero | Función de Lambda: `handleSupport` | Tiempo de espera: Predeterminado (29 segundos)

Etapas

Acciones de etapa Create stage

prod

Detalles de la etapa

Nombre de etapa: prod | Tasa: 10000 | ACL web: - | Clúster de caché: Inactivo | Ampliación: 5000 | Certificado de cliente: - | Almacenamiento en caché de nivel de método: predeterminado | URL de invocación: `https://8pyzq9zkd.execute-api.us-east-1.amazonaws.com/prod` | Implementación activa: 2v7qxr el May 22, 2025, 19:16 (UTC+02:00)

2.1.2. Activación/Desactivación del Cloud.

Al tratarse de una funcionalidad nueva desarrollada directamente en la nube, no es necesario modificar el código actual del backend. La integración se realiza exclusivamente desde el frontend, añadiendo un modal que se muestra al hacer clic en "Apoyar" cuando no hay un usuario autenticado. No se implementa ya que queda fuera del alcance de esta asignatura.

Este modal solicita el nombre y email, toma el *id* de la causa desde la página y envía la información mediante una petición POST al API Gateway. La respuesta permite informar al usuario si el apoyo fue registrado correctamente o si ya había apoyado anteriormente. Este diseño permite activar la funcionalidad cloud sin cambios en la lógica del sistema.

2.1.3. Ejemplo de funcionamiento.

A continuación, se muestra un ejemplo real del funcionamiento del servicio desde Thunder Client, realizando una solicitud HTTP POST al endpoint público desplegado en API Gateway.

POST

https://8ypyzq9zkd.execute-api.us-east-1.amazonaws.com/prod/apoyar

Send

Query

Headers 2

Auth

Body 1

Tests

Pre Run

JSON

XML

Text

Form

Form-encode

GraphQL

Binary

```

1 {
2   "name": "Joe Doe",
3   "email": "joe.doe@um.es",
4   "causeId": "educacion_global"
5 }

```

Status: 200 OK

Size: 45 Bytes

Time: 1.41 s

Response

```

1 {
2   "message": "Apoyo registrado correctamente"
3 }

```

POST

https://8ypyzq9zkd.execute-api.us-east-1.amazonaws.com/prod/apoyar

Send

Query

Headers 2

Auth

Body 1

Tests

Pre Run

JSON

XML

Text

Form

Form-encode

GraphQL

Binary

```

1 {
2   "name": "Joe Doe",
3   "email": "joe.doe@um.es",
4   "causeId": "educacion_global"
5 }

```

Status: 409 Conflict

Size: 56 Bytes

Time: 461 ms

Response

```

1 {
2   "message": "Ya has apoyado esta causa con este email."
3 }

```

Tabla: SupportsTable: elementos devueltos (1)

El análisis inició el mayo 22, 2025, 20:00:23

Acciones

Crear elemento

< 1 >

⚙

<input type="checkbox"/>	causeId (Cadena)	email (Cadena)	name	timestamp
<input type="checkbox"/>	educacion_global	joe.doe@um.es	Joe Doe	2025-05-22T17:58:13.723674

Como se puede observar, el servicio es capaz de almacenar correctamente los datos del usuario cuando son válidos, y de informar adecuadamente en caso de errores, como intentos duplicados de apoyo.

2.2. Funcionalidad: Almacenamiento de datos en DynamoDB.

Este apartado describe la implementación realizada para incorporar AWS DynamoDB como alternativa de almacenamiento para la entidad *Cause* en nuestro microservicio de comunidades. Esta funcionalidad surge como una oportunidad para evaluar la viabilidad y beneficios del uso de una base de datos NoSQL gestionada en la nube, específicamente diseñada para aplicaciones que requieren alta escalabilidad, disponibilidad continua y rendimiento estable, justamente lo que se persigue en el desarrollo de nuestra aplicación.

La funcionalidad desarrollada permite gestionar los datos de la entidad *Cause* mediante DynamoDB, manteniendo la flexibilidad del esquema característico del modelo NoSQL, con ventajas adicionales proporcionadas que son propias del servicio gestionado de AWS.

2.2.1. Componentes principales y tecnologías usadas.

Nuestra solución planteada utiliza **AWS DynamoDB** como base de datos gestionada en combinación con el AWS SDK para TypeScript, facilitando la interacción con AWS. El desarrollo del microservicio se basa en el framework NestJS, y LocalStack se ha empleado para realizar pruebas locales sin depender directamente de AWS. Además, AWS CloudFormation gestiona la infraestructura mediante código, mientras que Docker y Docker Compose proporcionan los entornos locales.

```

1  # Configuración de LocalStack en docker-compose.dynamodb.yml
2  version: '3.8'
3  services:
4    localstack:
5      container_name: localstack
6      image: localstack/localstack:latest
7      ports:
8        - "4566:4566"
9      environment:
10       - SERVICES=dynamodb,cloudformation
11       - DEBUG=1
12       - DEFAULT_REGION=us-east-1

```


Desde el punto de vista arquitectónico, hemos utilizado los patrones Repositorio y Factory, permitiendo abstraer el almacenamiento y elegir dinámicamente la base de datos.

2.2.2. Explicación del funcionamiento general del sistema.

Para implementar esta solución, hemos creado inicialmente la tabla de ejemplo *Cause* en DynamoDB mediante AWS CloudFormation, especificando la estructura y definiendo índices secundarios globales (GSI) que permiten realizar consultas optimizadas. A continuación, se muestra el ejemplo del código utilizado para la creación de la tabla en CloudFormation:

```

1  AWSTemplateFormatVersion: '2010-09-09'
2  Description: CloudFormation template for DynamoDB tables used in the Communities microservice
3
4  Resources:
5    CausesTable:
6      Type: AWS::DynamoDB::Table
7      Properties:
8        TableName: Causes
9        BillingMode: PROVISIONED
10       ProvisionedThroughput:
11         ReadCapacityUnits: 5
12         WriteCapacityUnits: 5
13       AttributeDefinitions:
14         - AttributeName: id
15           AttributeType: S
16         - AttributeName: communityId
17           AttributeType: S
18         - AttributeName: createdAt
19           AttributeType: S
20         - AttributeName: odsIndex
21           AttributeType: S
22       KeySchema:
23         - AttributeName: id
24           KeyType: HASH
25       GlobalSecondaryIndexes:
26         - IndexName: GSI1
27           KeySchema:
28             - AttributeName: communityId
29               KeyType: HASH
30             - AttributeName: createdAt
31               KeyType: RANGE
32           Projection:
33             ProjectionType: ALL
34           ProvisionedThroughput:
35             ReadCapacityUnits: 5
36             WriteCapacityUnits: 5
37         - IndexName: GSI2
38           KeySchema:
39             - AttributeName: odsIndex
40               KeyType: HASH
41             - AttributeName: createdAt
42               KeyType: RANGE
43           Projection:
44             ProjectionType: ALL
45           ProvisionedThroughput:
46             ReadCapacityUnits: 5
47             WriteCapacityUnits: 5

```

El sistema implementado permite elegir entre MongoDB y DynamoDB mediante la variable de entorno *DB_TYPE*. Al iniciar la aplicación, el servicio definido en *dynamodb-service.ts*, encargado de abstraer las comunicaciones con AWS, configura automáticamente el cliente DynamoDB utilizando los parámetros especificados.

```
private client: DynamoDBClient;

constructor() {
  const config: DynamoDBClientConfig = {
    region: envs.database.dynamodb.region,
    credentials: {
      accessKeyId: envs.database.dynamodb.accessKeyId,
      secretAccessKey: envs.database.dynamodb.secretAccess
    },
  };

  if (envs.database.dynamodb.endpoint?.trim()) {
    config.endpoint = envs.database.dynamodb.endpoint;
  }

  this.client = new DynamoDBClient(config);
}
```

Posteriormente, la capa de dominio solamente debe interactuar con el repositorio común que abstrae las operaciones CRUD, permitiendo operaciones específicas como la recuperación de datos por ID:

```
async findById(id: string): Promise<Domain.Cause> {
  try {
    const response = await this.dynamoDBService.getItem(
      CauseTableConfig.tableName,
      { [CauseTableConfig.primaryKey]: id },
    );

    if (!response.Item) {
      throw new EntityNotFoundError(`Cause with id ${id} not found`);
    }

    return CauseMapper.fromDynamoDB(response.Item as Cause);
  } catch (error) {
    // Manejo de errores
  }
}
```

2.2.3. Nivel de desarrollo alcanzado.

Actualmente se ha completado la configuración del cliente DynamoDB, el repositorio CRUD para la entidad Cause, la creación automática de tablas mediante CloudFormation, filtros básicos implementados y la integración con LocalStack para pruebas locales. Se dispone de scripts de utilidad en `/solidarianid/backend/apps/communities-ms/scripts` para gestionar rápidamente los recursos y configuraciones, un fragmento de ejemplo del script de deploy es el siguiente:

```
#!/bin/bash
ENVIRONMENT=${1:-development}
REGION="us-east-1"
ENDPOINT="http://localhost:4566"

aws cloudformation deploy \
  --template-file infrastructure/dynamodb-tables.yaml \
  --stack-name solidarianid-dynamodb-dev \
  --parameter-overrides Environment=development \
  --endpoint-url=$ENDPOINT \
  --region=$REGION
```

Entre los aspectos aún pendientes se encuentran extender la solución a otras entidades, soporte completo para transacciones, optimización avanzada de índices, manejo avanzado de errores, operaciones por lotes, migración de datos desde MongoDB y optimizaciones avanzadas de costos.

2.2.4. Uso de Servicios Cloud.

En esta implementación se integran principalmente Amazon DynamoDB y AWS CloudFormation. DynamoDB es el almacenamiento central del sistema, gestionando eficientemente datos mediante tablas e índices secundarios globales (GSI) para consultas rápidas. AWS CloudFormation facilita la gestión y el despliegue de los recursos necesarios mediante plantillas declarativas.

La decisión de utilizar DynamoDB aporta ventajas claras como escalabilidad automática, alta disponibilidad, modelo económico basado en consumo real, rendimiento constante, seguridad integrada, flexibilidad del esquema y menor mantenimiento operativo.

En la siguiente imagen podemos observar el resultado de integrar la aplicación con el servicio en remoto de AWS DynamoDB:

The screenshot shows the AWS DynamoDB console interface. On the left, there's a sidebar with navigation options like 'Panel', 'Tablas', 'Explorar elementos', etc. The main area displays the 'Causas' table. A green banner at the top of the table view indicates a successful query: 'Completado: Elementos devueltos: 8 Elementos escaneados: 8 - Eficiencia: 100% - RCU consumidas: 0,5'. Below this, the table 'Tabla: Causas: elementos devueltos (8)' is shown with columns: id (Cadena), createdAt, createdBy, description, endDate, and ods. The table contains 8 rows of data.

id (Cadena)	createdAt	createdBy	description	endDate	ods
1023c870-f2c6-4a7c-...	2025-05-2...	264458f8-...	Proyecto de a...	2036-03-3...	[{"N": "6"}]
144b7035-4f8d-40f3-...	2025-05-2...	a8f36737-6...	Campaña par...	2026-06-1...	[{"N": "4"}]
e7cf657e-afb1-4133-...	2025-05-2...	264458f8-...	Distribución d...	2037-10-1...	[{"N": "2"}]
d9333b0a-211d-4eeb-...	2025-05-2...	a8f36737-6...	Proyecto para...	2025-12-3...	[{"N": "15"}]
020b1d88-4674-472-...	2025-05-2...	264458f8-...	Campaña par...	2037-12-3...	[{"N": "15"}]
193442cd-2d97-46cd-...	2025-05-2...	a8f36737-6...	Campañas de ...	2026-01-3...	[{"N": "3"}]
d59123ea-ee52-486d-...	2025-05-2...	a8f36737-6...	Construcción ...	2026-03-0...	[{"N": "6"}]
a1e92c13-7af3-4718-...	2025-05-2...	a8f36737-6...	Distribución d...	2025-10-0...	[{"N": "2"}]

Esta configuración se ha podido lograr fácilmente, ya que disponemos de archivos con variables de entorno predefinidas, en los cuales solo es necesario establecer los valores correspondientes a las credenciales.

2.2.5. Activación/Desactivación del Cloud.

El sistema permite alternar entre MongoDB y DynamoDB mediante la variable de entorno `DB_TYPE`:

```
27 # Database Configuration
28 # Options: 'mongodb' or 'dynamodb'
29 DB_TYPE=dynamodb
30
31 # DynamoDB
32 DYNAMODB_REGION=us-east-1
33 DYNAMODB_ENDPOINT=http://localhost:4566
34 DYNAMODB_ACCESS_KEY_ID=local
35 DYNAMODB_SECRET_ACCESS_KEY=local
36 AWS_SESSION_TOKEN=
```

La configuración se establece desde el archivo `.env`, y también se puede utilizar un script específico (`switch-db.sh`) para facilitar este cambio en entornos locales.

```

1  #!/bin/bash
2
3  # Script to switch between MongoDB and DynamoDB
4
5  if [ "$1" == "mongo" ] || [ "$1" == "mongodb" ]; then
6      echo "Switching to MongoDB..."
7      sed -i '' 's/DB_TYPE=dynamodb/DB_TYPE=mongodb/g' ../.env.development
8      echo "Done! The application will now use MongoDB."
9      echo "To start the application, run: npm run start:dev communities-ms"
10 elif [ "$1" == "dynamo" ] || [ "$1" == "dynamodb" ]; then
11     echo "Switching to DynamoDB..."
12     sed -i '' 's/DB_TYPE=mongodb/DB_TYPE=dynamodb/g' ../.env.development
13
14     # Check if LocalStack is running
15     if ! docker ps | grep -q localstack; then
16         echo "LocalStack is not running. Starting it now..."
17         ./start-localstack.sh
18     else
19         echo "LocalStack is already running."
20     fi
21
22     # Create DynamoDB tables using CloudFormation
23     echo "Creating DynamoDB tables..."
24     ./create-dynamodb-tables.sh development
25
26     echo "Done! The application will now use DynamoDB."
27     echo "To start the application, run: npm run start:dev communities-ms"
28 else
29     echo "Usage: ./switch-db.sh [mongodb|dynamodb]"
30     echo "Current database type: $(grep DB_TYPE ../.env.development | cut -d '=' -f2)"
31 fi

```

La activación de DynamoDB mediante el script bash inicia automáticamente la infraestructura local necesaria, configura la conexión con DynamoDB y crea o actualiza tablas usando CloudFormation, garantizando un cambio transparente sin afectar la lógica del negocio.

2.3. Funcionalidad: Generación automática de avatares al registrar usuarios.

En este apartado se describe la solución implementada para incorporar la funcionalidad de generación automática de avatares al registrar nuevos usuarios en la plataforma SolidarianId. Esta característica surge como una oportunidad para integrar de forma sencilla e independiente el uso de un servicio de almacenamiento en la nube, específicamente **Amazon S3**, en el contexto del proyecto.

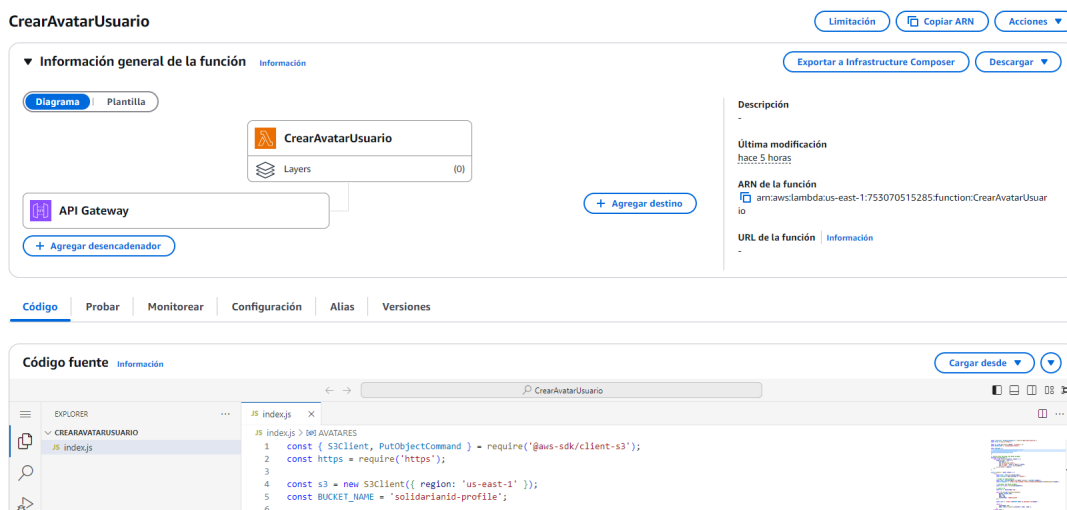
La funcionalidad desarrollada permite que, al registrar un nuevo usuario, el sistema genere automáticamente una imagen de perfil (avatar), basada en un estilo predefinido y asociada al email del usuario. Esta imagen se guarda en un bucket de Amazon S3 y la URL generada se vincula a la vista del perfil de un usuario.

Flujo general de la funcionalidad:

1. El usuario se registra desde el frontend, introduciendo sus datos personales, entre ellos el email.
2. El backend del microservicio de usuarios, al completar el registro realiza una solicitud POST a un endpoint HTTP expuesto en **API Gateway**.
3. Este endpoint llama a una función **AWS Lambda**, que genera un avatar SVG aleatorio usando la API pública de DiceBear, y lo guarda en Amazon S3 con el email del usuario como nombre de archivo.
4. El frontend utiliza esa URL para mostrar el avatar del usuario en su perfil.

2.3.1. Uso de Servicios Cloud.

- **Amazon S3:** se ha creado un bucket llamado *solidarianid-profile*. Los avatares generados se almacenan como archivos .svg con un identificador único basado en el email del usuario.
- **AWS Lambda:** se ha implementado una función Lambda llamada *CrearAvatarUsuario* que genera el avatar. La función elige aleatoriamente un avatar de DiceBear, lo sube al S3 con el email del usuario y responde con la URL del avatar generado.



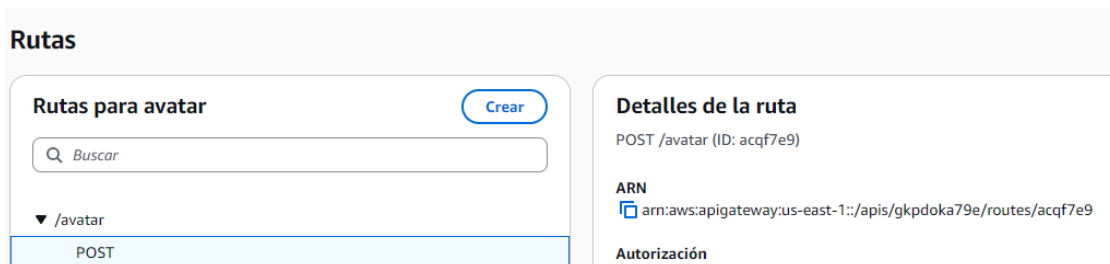
The screenshot shows the AWS Lambda console for the function 'CrearAvatarUsuario'. The 'Información general de la función' tab is active, displaying the function name, layers, and API Gateway. The 'Código fuente' tab shows the source code in JavaScript, which uses the AWS SDK to interact with S3 and DiceBear API.

```

1 const { S3Client, PutObjectCommand } = require('@aws-sdk/client-s3');
2 const https = require('https');
3
4 const s3 = new S3Client({ region: 'us-east-1' });
5 const BUCKET_NAME = 'solidarianid-profile';
6

```

- **Amazon API Gateway:** expone la función Lambda mediante el endpoint *POST /avatar* como punto de entrada público, simplificando la integración con el backend.



The screenshot shows the AWS API Gateway console for the 'POST /avatar' route. The 'Rutas para avatar' section shows the route details, including the method (POST) and the API ID. The 'Detalles de la ruta' section shows the route ID (acqf7e9) and the ARN.

La creación del bucket S3, la función Lambda y el endpoint en API Gateway se ha realizado desde la interfaz web de AWS, ya que se trataba de configuraciones sencillas sin requisitos especiales. Además, se aprovechó para familiarizarse con la consola de AWS y comprobar visualmente el funcionamiento de cada servicio. La función Lambda está implementada en un único archivo *index.js*.

Para integrar la funcionalidad de crear el avatar en el backend, se ha implementado un servicio *AwsLambdaService* en el microservicio de usuarios. Este servicio utiliza Axios para realizar la llamada HTTP al endpoint de API Gateway para crear el avatar. Por otra parte, desde el frontend se accede al avatar directamente desde el bucket S3, si el archivo no existe se muestra un avatar por defecto.

```

@Inject()
export class AwsAvatarService {
  private readonly endpoint =
    'https://gkpdoka79e.execute-api.us-east-1.amazonaws.com/avatar';

  async crearAvatar(username: string): Promise<string | null> {
    try {
      const response = await axios.post(this.endpoint, { username });

      if (response.data && response.data.avatarUrl) {
        return response.data.avatarUrl;
      }

      console.warn('Avatar no generado correctamente:', response.data);
      return null;
    } catch (err) {
      console.error('Error llamando al endpoint API Gateway:', err.message);
      return null;
    }
  }
}

```

Actualmente, la URL del avatar generado no se almacena en la base de datos del sistema, sino que se construye dinámicamente desde el frontend a partir del email del usuario y la convención de nombres aplicada en el bucket S3. Esta decisión se ha tomado para evitar modificar el modelo de datos y mantener la integración lo más desacoplada posible respecto al sistema original.


Ejemplo de los avatares aleatorios almacenados en el bucket y su visualización en el perfil de los usuarios:

solidarianid-profile Información

Objetos (4)


Los objetos son las entidades fundamentales que se almacenan en Amazon S3. Puede utilizar el [inventario de Amazon S3](#) para obtener una lista de todos los objetos de su bucket. Para que otras personas obtengan acceso a sus objetos, tendrá que concederles permisos de forma explícita. [Más información](#)

Nombre	Tipo	Última modificación	Tamaño	Clase de almacenamiento
auro@example.com.svg	svg	22 May 2025 11:33:48 PM CEST	1.5 KB	Estándar
dongyue@example.com.svg	svg	22 May 2025 11:34:05 PM CEST	1.5 KB	Estándar
hermansr@example.com.svg	svg	22 May 2025 11:34:25 PM CEST	1.5 KB	Estándar
pedro@example.com.svg	svg	22 May 2025 11:33:33 PM CEST	1.5 KB	Estándar




Aurora H
Hello, I'm Aurora!

SolidarianID: e9fffb095-426f-4e1f-9925-b66e3906b8af
Age: 35




Yue Y
Hello, I'm Yue!

SolidarianID: 3e91de62-ef8d-47cd-8360-3514e2ffe81d
Age: 35



Pedro Nicolás
Hello, I'm Pedro!

SolidarianID: be80490a-24b2-425a-99a6-a7299833ce9a
Age: 35



Hernán S
Hello, I'm Hernan!

SolidarianID: 8ba4d386-bfa2-4df2-8582-c477d4e12002
Age: 35

2.3.2. Activación/Desactivación del Cloud.

La integración con AWS para la generación de avatares ha sido diseñada de forma desacoplada y controlada. Se ha incorporado una variable de entorno `AVATAR_GENERATION_ENABLED`, la cual permite activar o desactivar dinámicamente esta funcionalidad sin necesidad de modificar el código fuente.

Cuando esta variable está configurada como *true*, el backend invoca el endpoint expuesto en API Gateway para generar el avatar automáticamente. En caso contrario, el proceso de registro de usuario se completa sin realizar ninguna operación relacionada con AWS.

```
// Create the new user and save it
const savedUser = await this.userRepository.save(user);

if (process.env.AVATAR_GENERATION_ENABLED === 'true') {
  try {
    await this.awsAvatarService.crearAvatar(email);
  } catch (err) {
    console.log(
      'Error generando avatar (desactivado o fallo):',
      err.message,
    );
  }
}
```

3. Conclusión.

La integración de servicios cloud en SolidarianID nos ha permitido experimentar de forma práctica con herramientas como AWS Lambda, API Gateway, DynamoDB y S3, incorporándolas en funcionalidades reales como el registro de apoyos anónimos o la generación automática de avatares. Este proceso no solo ha enriquecido la plataforma con nuevas capacidades, sino que también nos ha servido para comprender mejor cómo diseñar servicios modulares, escalables y fácilmente activables o desactivables según las necesidades del sistema.

El trabajo realizado ha facilitado el aprendizaje progresivo sobre la arquitectura cloud y ha demostrado que es posible extender la funcionalidad de la plataforma sin afectar su núcleo. Aunque el resultado ha sido satisfactorio, siguen existiendo posibilidades de evolución, como una autenticación basada en Lambda o la migración completa de la persistencia del microservicio de comunidades a DynamoDB. En conjunto, esta experiencia ha sentado una base sólida para futuras mejoras, combinando la funcionalidad existente con las ventajas del ecosistema cloud.

4. Bibliografía.

- Amazon Web Services. (2023). AWS Lambda Documentation.
<https://docs.aws.amazon.com/lambda/>
- Amazon Web Services. (2023). Amazon API Gateway Developer Guide.
<https://docs.aws.amazon.com/apigateway/>
- Amazon Web Services. (2023). Amazon DynamoDB Documentation.
<https://docs.aws.amazon.com/dynamodb/>
- Amazon Web Services. (2023). Amazon S3 Documentation.
<https://docs.aws.amazon.com/AmazonS3/>
- DiceBear. <https://www.dicebear.com/styles/thumbs/>
- Laboratorios de AWS de la asignatura.
- Recursos de la asignatura del Aula Virtual.