Ejercicio entregable 1
Computación en la Nube

Nicolás Rey Alonso

16 de diciembre de 2025

Índice

1. Introducción **4**

2. Acoplado **5**
2.1. Diagrama de arquitectura 6
2.2. Explicación de los templates de CloudFormation 7
2.2.1. Repositorios de imágenes 7
2.2.1.1. ecr.yml - Repositorio ECR para Crumblr 7
2.2.1.2. Crumb-ecr-gui.yaml - Repositorio ECR para Crumblr Frontend 8
2.2.2. Bases de datos 10
2.2.2.1. db-postgres.yml - Base de datos PostgreSQL para Crumblr (Amazon RDS) 10
2.2.3. ECS Fargate 13
2.2.3.1. main.yml - Despliegue del backend 13
2.2.3.2. frontEC2Cluster.yml - Despliegue del Frontend 17
2.3. Código 21
2.3.1. Código backend 21
2.3.1.1. db.py 21
2.3.1.2. factory.py 21
2.3.1.3. postgres-db.py 22
2.3.1.4. crumb.py 24
2.3.1.5. main.py 24
2.3.2. Dockerfile 27

3. FrontEnd **28**
3.1. Código 28
3.1.1. app.js 28
3.1.2. index.html 30
3.1.3. style.css 31
3.2. dockerfile 32

4. Desacoplado **33**
4.1. Diagrama de arquitectura 34
4.2. Explicación de los templates de CloudFormation 34
4.2.1. ECR 34
4.2.1.1. BUILDSLLECR.yml - Repositorios ECR para las Lambdas de Crumblr 34
4.2.1.2. Crumblr-ecr-gui - Repositorio ECR para el Frontend de Crumblr 37
4.2.2. Bases de datos 37
4.2.2.1. db-postgres 37
4.2.3. Lanzamineto de Aplicación 37
4.2.3.1. main.yml - Despliegue del Backend de Crumblr 37
4.2.3.2. frontEC2Cluster.yml - Despliegue del Frontend 39
4.3. Código 40
4.3.1. Handlers 40
4.3.2. Shared 46

4.3.2.1. crumb.py 46
4.3.2.2. shared/*.py 46
4.3.2.3. shared/crumb-service 46
4.3.3. dockerfile 47

5. Presupuestos **49**
5.1. Costo acoplado 49
5.2. Costo desacoplado 49
5.3. Costo Frontend 49
5.4. Resumen de costos totales 50

6. Conclusiones **51**

7. Uso de la IA **51**

8. github **51**

9. Arreglos tras revisión **52**
9.1. Cambios en arquitectura 52

1. Introducción

En este informe se detallan las actividades realizadas, centradas en el despliegue de infraestructura y servicios en AWS mediante **CloudFormation**. Se describen los trabajos tanto de manera desacoplada como el trabajo en curso de integración/acoplamiento.

2. Acoplado

He editado las plantillas YAML suministradas por el profesor para generar un nuevo conjunto de plantillas que desplieguen los componentes del proyecto *Crumblr*, separando frontend y backend.

En el proyecto de github se encuentra el progreso realizado a lo largo del desarrollo de la practica así como las utilidades *shell scripts* creados. Sin embargo, en esta sección se describen las modificaciones y la estructura de las nuevas plantillas CloudFormation. Los componentes principales que he modificado/creado son los siguientes:

- **Repositorios de imágenes:** Dos plantillas YAML para la creación de los repositorios en Amazon ECR, uno para el frontend y otro para el backend.
- **Backend:** Una plantilla YAML que despliega un **ECS cluster** con la **task definition** correspondiente al backend, incluyendo la configuración de API Gateway para exponer los endpoints de la aplicación.
- **Frontend:** Una plantilla YAML que despliega un **ECS cluster** para el frontend, configurando el **Application Load Balancer** y el security group necesario.

Estas configuraciones permiten que cada componente funcione de manera autónoma y puedan probarse independientemente.

2.1. Diagrama de arquitectura

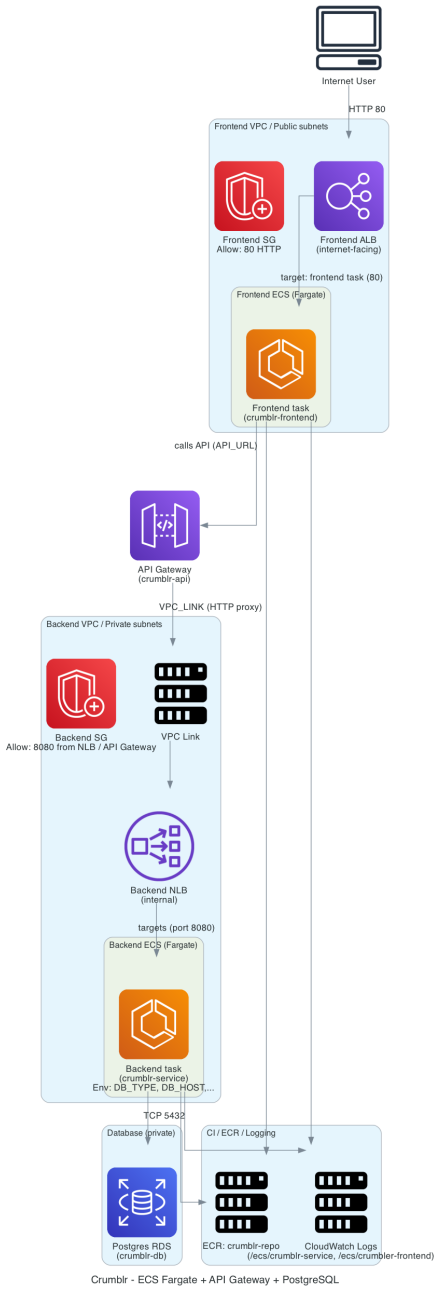

Figura 1: Diagrama de la arquitectura de Crumblr

2.2. Explicación de los templates de CloudFormation

2.2.1. Repositorios de imágenes

2.2.1.1. ecr.yml - Repositorio ECR para Crumblr Esta plantilla CloudFormation define la infraestructura necesaria para almacenar las imágenes Docker de la aplicación Crumblr en Amazon Elastic Container Registry (ECR).

Listing 1: Configuración del repositorio ECR

```
AWSTemplateFormatVersion: "2010-09-09"
Description: "Crumblr ECR Repository"

Parameters:
  RepositoryName:
    Type: String
    Default: cn/crumblr-repo
    Description: ECR repository name

Resources:
  ECRRepository:
    Type: AWS::ECR::Repository
    Properties:
      RepositoryName: !Ref RepositoryName
      ImageScanningConfiguration:
        ScanOnPush: false
      LifecyclePolicy:
        LifecyclePolicyText: |
          {
            "rules": [
              {
                "rulePriority": 1,
                "description": "Mantain only last 2 images",
                "selection": {
                  "tagStatus": "any",
                  "countType": "imageCountMoreThan",
                  "countNumber": 2
                },
                "action": {
                  "type": "expire"
                }
              }
            ]
          }

Outputs:
  RepositoryUri:
    Description: "ECR repository URI"
    Value: !GetAtt ECRRepository.RepositoryUri
    Export:
      Name: Crumblr-ECR-URI

  RepositoryName:
```

```
    Description: "ECR repository name"
    Value: !Ref RepositoryName
    Export:
      Name: Crumblr-ECR-Name

  RepositoryArn:
    Description: "ECR repository ARN"
    Value: !GetAtt ECRRepository.Arn
    Export:
      Name: Crumblr-ECR-ARN
```

Componentes principales:

- **Parameters - RepositoryName:** Define un parámetro configurable para el nombre del repositorio, con valor por defecto `cn/crumblr-repo`.
- **ECRRepository:** Crea el repositorio ECR donde se almacenarán las imágenes Docker. Utiliza el nombre especificado en los parámetros mediante la función intrínseca `!Ref`.
- **ImageScanningConfiguration:** Tiene desactivado el escaneo automático de vulnerabilidades (`ScanOnPush: false`). En un entorno de producción, se recomienda activar esta funcionalidad para detectar posibles problemas de seguridad en las imágenes, pero por comodidad y para acelerar la entrega de esta práctica decidí dejarlo desactivado.
- **LifecyclePolicy:** Implementa una política de gestión del ciclo de vida que mantiene únicamente las 2 imágenes más recientes en el repositorio. Esto se mantiene ya que así se reduce el costo
- **Outputs:** Exporta tres valores esenciales que pueden ser referenciados por otras plantillas CloudFormation:
  - **RepositoryUri:** La URI completa del repositorio, necesaria para hacer push de imágenes Docker.
  - **RepositoryName:** El nombre del repositorio, útil para scripts de automatización.
  - **RepositoryArn:** El ARN del repositorio, requerido para configurar permisos IAM y políticas de acceso.

2.2.1.2. Crumb-ecr-gui.yaml - Repositorio ECR para Crumblr Frontend Esta plantilla CloudFormation define el repositorio de Amazon Elastic Container Registry (ECR) destinado a almacenar las imágenes Docker correspondientes al *frontend* de la aplicación Crumblr.

Listing 2: Configuración del repositorio ECR para Crumblr Frontend

```
AWSTemplateFormatVersion: "2010-09-09"
Description: "Crumblr ECR Repository"

Parameters:
  RepositoryName:
```

```
    Type: String
    Default: cn/crumblr-gui
    Description: ECR repository name

Resources:
  ECRRepository:
    Type: AWS::ECR::Repository
    Properties:
      RepositoryName: !Ref RepositoryName
      ImageScanningConfiguration:
        ScanOnPush: false
      LifecyclePolicy:
        LifecyclePolicyText: |
          {
            "rules": [
              {
                "rulePriority": 1,
                "description": "Mantain only last 2 images",
                "selection": {
                  "tagStatus": "any",
                  "countType": "imageCountMoreThan",
                  "countNumber": 2
                },
                "action": {
                  "type": "expire"
                }
              }
            ]
          }

Outputs:
  RepositoryUri:
    Description: "ECR repository URI"
    Value: !GetAtt ECRRepository.RepositoryUri
    Export:
      Name: Crumblr-Front-ECR-URI

  RepositoryName:
    Description: "ECR repository name"
    Value: !Ref RepositoryName
    Export:
      Name: Crumblr-Front-ECR-Name

  RepositoryArn:
    Description: "ECR repository ARN"
    Value: !GetAtt ECRRepository.Arn
    Export:
      Name: Crumblr-Front-ECR-ARN
```

Componentes principales:

- **Parameters - RepositoryName:** Define el nombre del repositorio ECR con valor

por defecto `cn/crumblr-gui`, utilizado para almacenar las imágenes Docker del *frontend* de la aplicación.

- **ECRRepository:** Crea el repositorio ECR que contendrá las imágenes del *frontend*. Utiliza el parámetro definido anteriormente mediante la función `!Ref`.
- **ImageScanningConfiguration:** El escaneo automático de vulnerabilidades está deshabilitado (`ScanOnPush: false`). De forma similar al repositorio del backend, esta decisión se toma para agilizar los despliegues en el entorno de pruebas, aunque se recomienda activarlo en entornos de producción.
- **LifecyclePolicy:** Aplica una política que mantiene únicamente las dos imágenes más recientes, eliminando versiones antiguas para optimizar el uso del almacenamiento y reducir costos.
- **Outputs:** Exporta tres valores clave que permiten referenciar este repositorio desde otras plantillas o configuraciones:
  - **RepositoryUri:** URI del repositorio ECR, necesaria para realizar el *push* de imágenes Docker del *frontend*.
  - **RepositoryName:** Nombre del repositorio
  - **RepositoryArn:** ARN del repositorio

2.2.2. Bases de datos

2.2.2.1. db-postgres.yml - Base de datos PostgreSQL para Crumblr (Amazon RDS) Esta plantilla de CloudFormation crea una instancia de base de datos PostgreSQL administrada en Amazon RDS. Define parámetros de configuración, subredes privadas, grupos de seguridad y exporta el endpoint de conexión para su uso en otras pilas.

Listing 3: RDS PostgreSQL para Crumblr - Base de datos persistente

```
AWSTemplateFormatVersion: "2010-09-09"
Description: "RDS PostgreSQL para Crumblr (almacena los crumbs)"

Parameters:
  VpcId:
    Type: AWS::EC2::VPC::Id
    Description: VPC para la base de datos

  SubnetIds:
    Type: List<AWS::EC2::Subnet::Id>
    Description: Subnets (minimo 2)

  DBName:
    Type: String
    Default: crumblr_db
    Description: Nombre de la base de datos

  DBUser:
    Type: String
    Default: postgres
```

```
    Description: Usuario maestro

  DBPassword:
    Type: String
    NoEcho: true
    MinLength: 8
    Description: Contrasena del usuario maestro

Resources:
  DBSecurityGroup:
    Type: AWS::EC2::SecurityGroup
    Properties:
      GroupDescription: "Security group para RDS PostgreSQL (Crumblr)"
      VpcId: !Ref VpcId
      SecurityGroupIngress:
        - IpProtocol: tcp
          FromPort: 5432
          ToPort: 5432
          CidrIp: 0.0.0.0/0  # Abierto a todos; idealmente restringirlo al bac

  DBSubnetGroup:
    Type: AWS::RDS::DBSubnetGroup
    Properties:
      DBSubnetGroupName: crumblr-db-subnet-group
      DBSubnetGroupDescription: "Subnet group para la base de datos PostgreSQL
      SubnetIds: !Ref SubnetIds

  PostgresDBInstance:
    Type: AWS::RDS::DBInstance
    Properties:
      DBInstanceIdentifier: crumblr-postgres-db
      Engine: postgres
      EngineVersion: "17.6"
      DBInstanceClass: db.t3.micro
      AllocatedStorage: 20
      StorageType: gp2
      DBName: !Ref DBName
      MasterUsername: !Ref DBUser
      MasterUserPassword: !Ref DBPassword
      VPCSecurityGroups:
        - !Ref DBSecurityGroup
      DBSubnetGroupName: !Ref DBSubnetGroup
      PubliclyAccessible: true   # Solo si el backend no esta en la misma VPC
      BackupRetentionPeriod: 0
      DeletionProtection: false
      MultiAZ: false
      StorageEncrypted: false

Outputs:
  DBEndpoint:
    Description: "Endpoint del servidor PostgreSQL de Crumblr"
```

```
    Value: !GetAtt PostgresDBInstance.Endpoint.Address
    Export:
      Name: Crumblr-DB-Endpoint

  DBPort:
    Description: "Puerto del servidor PostgreSQL"
    Value: !GetAtt PostgresDBInstance.Endpoint.Port
    Export:
      Name: Crumblr-DB-Port

  DBName:
    Description: "Nombre de la base de datos"
    Value: !Ref DBName
    Export:
      Name: Crumblr-DB-Name

  DBUser:
    Description: "Usuario maestro de la base de datos"
    Value: !Ref DBUser
    Export:
      Name: Crumblr-DB-User
```

Componentes principales:

- **Parameters:** Permiten configurar la plantilla con la VPC, subredes, nombre de base de datos y credenciales. El parámetro `DBPassword` usa `NoEcho` para ocultar la contraseña en la consola de AWS.
- **DBSecurityGroup:** Define un grupo de seguridad que habilita el puerto 5432 para conexiones PostgreSQL. El rango `0.0.0.0/0` permite acceso público, pero se recomienda limitarlo al rango del backend ECS o a una IP específica.
- **DBSubnetGroup:** Agrupa las subredes donde se desplegará la base de datos, asegurando redundancia y aislamiento dentro de la VPC.
- **PostgresDBInstance:**
  - Crea la instancia RDS usando el motor `postgres` versión 17.6.
  - Define una clase `db.t3.micro`, ideal para entornos de desarrollo o pruebas.
  - Almacena 20 GB en volumen SSD (`gp2`), con backups deshabilitados y sin Multi-AZ.
  - `PubliclyAccessible: true` permite conexión externa, útil si el backend no reside en la misma red privada.
- **Outputs:** Exporta los valores del endpoint, puerto, nombre de base y usuario, de modo que otras pilas (por ejemplo, la del backend ECS) puedan importarlos mediante referencias cruzadas de CloudFormation.

2.2.3. ECS Fargate

2.2.3.1. main.yml - Despliegue del backend Esta plantilla de CloudFormation implementa la infraestructura central de la aplicación Crumblr, incluyendo el servicio backend desplegado sobre Amazon ECS Fargate, un balanceador de carga interno (NLB), una API REST administrada por Amazon API Gateway y una base de datos configurable (PostgreSQL o DynamoDB).

Listing 4: Infraestructura principal de Crumblr — ECS + API Gateway + DB

```
AWSTemplateFormatVersion: "2010-09-09"
Description: "Crumblr - ECS Fargate + API Gateway + PostgreSQL (or DynamoDB)"

Parameters:
  ImageName:
    Type: String
    Description: ECR Image name (from Crumblr ECR stack)
    Default: crumblr-repo:latest

  VpcId:
    Type: AWS::EC2::VPC::Id
    Description: Target VPC

  SubnetIds:
    Type: List<AWS::EC2::Subnet::Id>
    Description: At least 2 subnets in different AZs

  DBType:
    Type: String
    Default: postgres
    AllowedValues:
      - postgres
      - dynamodb

  DBHost:
    Type: String
    Default: ""
    Description: "DB Host (only used for PostgreSQL)"

  DBName:
    Type: String
    Default: ""
    Description: "DB Name (only used for PostgreSQL)"

  DBUser:
    Type: String
    Default: ""
    Description: "DB User (only used for PostgreSQL)"

  DBPass:
    Type: String
    NoEcho: true
    Default: ""
```

```
    Description: "DB Password (only used for PostgreSQL)"

  DBDynamoName:
    Type: String
    Default: "crumbs"
    Description: "DynamoDB table name"

# --- Networking ---
Resources:
  ECSSecurityGroup:
    Type: AWS::EC2::SecurityGroup
    Properties:
      GroupDescription: Security group for Crumblr ECS tasks
      VpcId: !Ref VpcId
      SecurityGroupIngress:
        - IpProtocol: tcp
          FromPort: 8080
          ToPort: 8080
          CidrIp: 0.0.0.0/0

# --- Load Balancer ---
  NLB:
    Type: AWS::ElasticLoadBalancingV2::LoadBalancer
    Properties:
      Name: crumblr-nlb
      Type: network
      Scheme: internal
      Subnets: !Ref SubnetIds

  TargetGroup:
    Type: AWS::ElasticLoadBalancingV2::TargetGroup
    Properties:
      Name: crumblr-tg
      Port: 8080
      Protocol: TCP
      VpcId: !Ref VpcId
      TargetType: ip
      HealthCheckProtocol: HTTP
      HealthCheckPath: /health

  Listener:
    Type: AWS::ElasticLoadBalancingV2::Listener
    Properties:
      DefaultActions:
        - Type: forward
          TargetGroupArn: !Ref TargetGroup
      LoadBalancerArn: !Ref NLB
      Port: 8080
      Protocol: TCP

# --- ECS ---
```

```
ECSCluster:
  Type: AWS::ECS::Cluster
  Properties:
    ClusterName: crumblr-cluster
CrumblrLogGroup:
  Type: AWS::Logs::LogGroup
  Properties:
    LogGroupName: /ecs/crumblr-service
    RetentionInDays: 14

TaskDefinition:
  Type: AWS::ECS::TaskDefinition
  Properties:
    Family: crumblr-task
    NetworkMode: awsvpc
    RequiresCompatibilities: [FARGATE]
    Cpu: 256
    Memory: 512
    ExecutionRoleArn: !Sub "arn:aws:iam::${AWS::AccountId}:role/LabRole"
    TaskRoleArn: !Sub "arn:aws:iam::${AWS::AccountId}:role/LabRole"
    ContainerDefinitions:
      - Name: crumblr-container
        Image: !Sub "${AWS::AccountId}.dkr.ecr.${AWS::Region}.amazonaws.com/
        PortMappings:
          - ContainerPort: 8080
        LogConfiguration:
          LogDriver: awslogs
          Options:
            awslogs-group: /ecs/crumblr-service
            awslogs-region: !Ref AWS::Region
            awslogs-stream-prefix: crumblr
        Environment:
          - Name: DB_TYPE
            Value: !Ref DBType
          - Name: DB_HOST
            Value: !Ref DBHost
          - Name: DB_NAME
            Value: !Ref DBName
          - Name: DB_USER
            Value: !Ref DBUser
          - Name: DB_PASS
            Value: !Ref DBPass
          - Name: DB_DYNAMONAME
            Value: !Ref DBDynamoName

ECSService:
  Type: AWS::ECS::Service
  DependsOn: Listener
  Properties:
    Cluster: !Ref ECSCluster
    ServiceName: crumblr-service
```

```
      TaskDefinition: !Ref TaskDefinition
      DesiredCount: 1
      LaunchType: FARGATE
      NetworkConfiguration:
        AwsvpcConfiguration:
          AssignPublicIp: ENABLED
          Subnets: !Ref SubnetIds
          SecurityGroups: [!Ref ECSSecurityGroup]
      LoadBalancers:
        - ContainerName: crumblr-container
          ContainerPort: 8080
          TargetGroupArn: !Ref TargetGroup

# --- API Gateway ---
  VPCLink:
    Type: AWS::ApiGateway::VpcLink
    Properties:
      Name: crumblr-vpc-link
      TargetArns: [!Ref NLB]

  RestAPI:
    Type: AWS::ApiGateway::RestApi
    Properties:
      Name: crumblr-api
      Description: "REST API for Crumblr CRUD operations"

# (Se omiten metodos por brevedad)

# --- Outputs ---
Outputs:
  APIEndpoint:
    Description: "Crumblr API URL"
    Value: !Sub "https://${RestAPI}.execute-api.${AWS::Region}.amazonaws.com/p

  APIKeyId:
    Description: "API Key ID"
    Value: !Ref APIKey
```

Componentes principales:

- **Parameters:** Permiten configurar la plantilla con valores dinámicos, como la imagen del contenedor, la VPC de destino y el tipo de base de datos (PostgreSQL o DynamoDB). Las credenciales de base de datos se definen con la opción `NoEcho` para no mostrarse en la consola.
- **Networking:** Crea un grupo de seguridad (`ECSSecurityGroup`) que habilita tráfico TCP en el puerto 8080, necesario para las tareas ECS y el balanceador.
- **Load Balancer (NLB):** Implementa un balanceador de carga interno de tipo `network` con su correspondiente `TargetGroup` y `Listener`, encaminando el tráfico hacia las tareas ECS.
- **ECS Fargate:**

- `ECSCluster`: Define el clúster Fargate donde correrán los servicios.
- `TaskDefinition`: Especifica los recursos (CPU, memoria) y la imagen Docker obtenida del repositorio ECR.
- Variables de entorno (`DB_TYPE`, `DB_HOST`, etc.) permiten que el contenedor se conecte dinámicamente al motor de base de datos configurado.
- `ECSService`: Lanza una instancia del contenedor y la asocia al balanceador mediante `TargetGroupArn`.

- **API Gateway:**
  - Define un `RestAPI` con recursos `/crumbs` y `/crumbs/{id}`, conectados al backend mediante un `VpcLink`.
  - Configura los métodos HTTP (GET, POST, PUT, DELETE) con integración tipo `HTTP_PROXY` apuntando al NLB.
  - Implementa opciones CORS para habilitar solicitudes desde el frontend.
- **Deployment y seguridad:**
  - El `ApiGateway::Deployment` y el `Stage` crean el entorno `prod`.
  - Se genera una `APIKey` y un `UsagePlan` asociado, limitando el acceso a consumidores autorizados.
- **Outputs:** Exporta la URL pública del API Gateway (`APIEndpoint`) y el identificador de la API Key, ambos útiles para integraciones externas o pruebas.

2.2.3.2. frontEC2Cluster.yml - Despliegue del Frontend Esta plantilla CloudFormation despliega la infraestructura necesaria para el frontend de Crumblr sobre ECS Fargate con un Application Load Balancer (ALB), conectándose al backend mediante URL y API Key.

Listing 5: Crumblr Frontend — ECS Fargate + ALB

```
AWSTemplateFormatVersion: "2010-09-09"
Description: "Crumblr Frontend on ECS Fargate + ALB"

Parameters:
  VpcId:
    Type: AWS::EC2::VPC::Id
  SubnetIds:
    Type: List<AWS::EC2::Subnet::Id>
  ImageName:
    Type: String
    Description: "ECR image name (e.g., crumblr-frontend-repo:latest)"
  APIURL:
    Type: String
    Description: The full API URL (e.g., https://...)
  APIKEY:
    Type: String
    Description: The API key
```

```
Resources:
  FrontendSecurityGroup:
    Type: AWS::EC2::SecurityGroup
    Properties:
      GroupDescription: "Allow HTTP for Crumblr Frontend"
      VpcId: !Ref VpcId
      SecurityGroupIngress:
        - IpProtocol: tcp
          FromPort: 80
          ToPort: 80
          CidrIp: 0.0.0.0/0
      SecurityGroupEgress:
        - IpProtocol: -1
          CidrIp: 0.0.0.0/0

  FrontendCluster:
    Type: AWS::ECS::Cluster
    Properties:
      ClusterName: crumblr-cluster-front

  FrontendALB:
    Type: AWS::ElasticLoadBalancingV2::LoadBalancer
    Properties:
      Name: crumblr-frontend-alb
      Scheme: internet-facing
      Subnets: !Ref SubnetIds
      SecurityGroups:
        - !Ref FrontendSecurityGroup
      Type: application

  FrontendTargetGroup:
    Type: AWS::ElasticLoadBalancingV2::TargetGroup
    Properties:
      VpcId: !Ref VpcId
      Protocol: HTTP
      Port: 80
      TargetType: ip
      HealthCheckPath: /
      Matcher:
        HttpCode: 200

  FrontendListener:
    Type: AWS::ElasticLoadBalancingV2::Listener
    Properties:
      LoadBalancerArn: !Ref FrontendALB
      Port: 80
      Protocol: HTTP
      DefaultActions:
        - Type: forward
          TargetGroupArn: !Ref FrontendTargetGroup
```

```
FrontendLogGroup:
  Type: AWS::Logs::LogGroup
  Properties:
    LogGroupName: /ecs/crumbler-frontend
    RetentionInDays: 14

FrontendTaskDefinition:
  Type: AWS::ECS::TaskDefinition
  Properties:
    Family: crumblr-frontend-task
    NetworkMode: awsvpc
    RequiresCompatibilities: [FARGATE]
    Cpu: 256
    Memory: 512
    ExecutionRoleArn: !Sub arn:aws:iam::${AWS::AccountId}:role/LabRole
    ContainerDefinitions:
      - Name: crumblr-frontend
        Image: !Sub "${AWS::AccountId}.dkr.ecr.${AWS::Region}.amazonaws.com/
        PortMappings:
          - ContainerPort: 80
        Environment:
          - Name: API_URL
            Value: !Ref APIURL
          - Name: API_KEY
            Value: !Ref APIKEY
        LogConfiguration:
          LogDriver: awslogs
          Options:
            awslogs-group: /ecs/crumbler-frontend
            awslogs-region: !Ref AWS::Region
            awslogs-stream-prefix: crumblr-frontend

FrontendService:
  Type: AWS::ECS::Service
  DependsOn:
    - FrontendListener
  Properties:
    Cluster: !Ref FrontendCluster
    ServiceName: crumblr-frontend-service
    TaskDefinition: !Ref FrontendTaskDefinition
    DesiredCount: 1
    LaunchType: FARGATE
    NetworkConfiguration:
      AwsvpcConfiguration:
        AssignPublicIp: ENABLED
        Subnets: !Ref SubnetIds
        SecurityGroups:
          - !Ref FrontendSecurityGroup
    LoadBalancers:
      - ContainerName: crumblr-frontend
        ContainerPort: 80
```

```
            TargetGroupArn: !Ref FrontendTargetGroup

Outputs:
  FrontendURL:
    Description: "Crumblr Frontend URL"
    Value: !GetAtt FrontendALB.DNSName
```

Componentes principales:

- **Parameters:** Permiten configurar dinámicamente la plantilla con:
  - `VpcId` y `SubnetIds` para definir la red donde se desplegará el frontend.
  - `ImageName` para indicar la imagen del frontend en ECR.
  - `APIURL` y `APIKEY` para que el contenedor pueda conectarse al backend.
- **FrontendSecurityGroup:** Grupo de seguridad que permite tráfico HTTP entrante en el puerto 80 y todo el tráfico saliente. Esto asegura que el frontend sea accesible públicamente y pueda comunicarse con otros servicios.
- **FrontendCluster:** Clúster ECS donde se ejecutarán las tareas Fargate del frontend.
- **FrontendALB, FrontendTargetGroup y FrontendListener:**
  - `FrontendALB` es el Application Load Balancer público que distribuye el tráfico a las tareas ECS.
  - `FrontendTargetGroup` define los objetivos (IP de contenedores) y la verificación de salud.
  - `FrontendListener` escucha en el puerto 80 y redirige el tráfico al Target Group.
- **FrontendLogGroup:** Grupo de logs de CloudWatch donde se almacenarán los registros del contenedor, con retención de 14 días.
- **FrontendTaskDefinition:** Define los recursos de cada tarea Fargate, la imagen Docker, variables de entorno (`API_URL`, `API_KEY`) y la configuración de logs.
- **FrontendService:** Lanza la tarea Fargate y la asocia al ALB mediante el Target Group, asegurando alta disponibilidad y asignación de IP pública para acceso externo.
- **Outputs:** Exporta la URL del frontend (`FrontendURL`) para que pueda ser usada en documentación o integraciones posteriores.

2.3. Código

2.3.1. Código backend

El código del backend decidí mantenerlo con flask por comodidad y porque gran parte de la arquitectura del código ya estaba hecha. Los principales cambios que realicé se basan en diverger de la implementación original con tickets para adaptarla a Crumbs.

2.3.1.1. db.py . Define la clase abstracta `Database` que establece los métodos que deben implementar todas las clases de acceso a datos de Crumblr. Se mantiene la misma estructura que en la implementación original con tickets, adaptando los nombres y tipos de datos a los de Crumbs.

Listing 6: Interfaz de base de datos de Crumblr

```
from abc import ABC, abstractmethod
from typing import List, Optional
from models.crumb import Crumb

class Database(ABC):

    @abstractmethod
    def initialize(self):
        pass

    @abstractmethod
    def create_crumb(self, crumb: Crumb) -> Crumb:
        pass

    @abstractmethod
    def get_crumb(self, crumb_id: str) -> Optional[Crumb]:
        pass

    @abstractmethod
    def get_all_crumbs(self) -> List[Crumb]:
        pass

    @abstractmethod
    def update_crumb(self, crumb_id: str, crumb: Crumb) -> Optional
       [Crumb]:
        pass

    @abstractmethod
    def delete_crumb(self, crumb_id: str) -> bool:
        pass
```

2.3.1.2. factory.py Implementa la clase `DatabaseFactory`, que permite crear instancias de distintas implementaciones de `Database` de forma dinámica.

Listing 7: Fábrica de bases de datos de Crumblr

```
import os
from typing import Dict, Type
```

```
from .db import Database
from .postgres_db import PostgresDatabase


class DatabaseFactory:

    _databases: Dict[str, Type[Database]] = {
        'postgres': PostgresDatabase
    }

    @classmethod
    def create(cls, db_type: str = None) -> Database:
        if db_type is None:
            db_type = os.getenv('DB_TYPE', 'postgres')

        db_type = db_type.lower()

        database_class = cls._databases.get(db_type)

        if database_class is None:
            available = ', '.join(cls._databases.keys())
            raise ValueError(
                f"DB_TYPE '{db_type}' no valido. "
                f"Opciones disponibles: {available}"
            )
        return database_class()

    @classmethod
    def get_available_databases(cls) -> list:
        return list(cls._databases.keys())
```

2.3.1.3. postgres-db.py . Implementa la clase `PostgresDatabase`, que hereda de `Database` y gestiona la persistencia de los *crumbs* en PostgreSQL.

Listing 8: Implementación PostgreSQL de la base de datos Crumblr

```
import psycopg2
import psycopg2.extras
import json
from typing import List, Optional
from .db import Database
from models.crumb import Crumb
import os

class PostgresDatabase(Database):

    def __init__(self):
        self.connection = psycopg2.connect(
            host=os.getenv('DB_HOST'),
            user=os.getenv('DB_USER'),
            password=os.getenv('DB_PASS'),
            database=os.getenv('DB_NAME')
```

```
        )
        self.connection.autocommit = True
        self.initialize()

    def initialize(self):
        with self.connection.cursor() as cursor:
            cursor.execute("""
                CREATE TABLE IF NOT EXISTS crumbs (
                    crumb_id VARCHAR(36) PRIMARY KEY,
                    content TEXT NOT NULL,
                    image_url TEXT,
                    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
                );
            """)

    def create_crumb(self, crumb: Crumb) -> Crumb:
        with self.connection.cursor() as cursor:
            sql = """
                INSERT INTO crumbs (crumb_id, content, image_url,
                   created_at)
                VALUES (%s, %s, %s, %s)
            """
            cursor.execute(sql, (crumb.crumb_id, crumb.content,
               crumb.image_url, crumb.created_at))
        return crumb


    def get_crumb(self, crumb_id: str) -> Optional[Crumb]:
        with self.connection.cursor(cursor_factory=psycopg2.extras.
           RealDictCursor) as cursor:
            sql = "SELECT * FROM crumbs WHERE crumb_id = %s"
            cursor.execute(sql, (crumb_id,))
            result = cursor.fetchone()
            if result:
                result = dict(result)
                result['created_at'] = (
                    result['created_at'].isoformat() if result['
                       created_at'] else None
                )
                return Crumb(**result)
        return None


    def get_all_crumbs(self):
        with self.connection.cursor(cursor_factory=psycopg2.extras.
           RealDictCursor) as cursor:
            sql = "SELECT * FROM crumbs ORDER BY created_at DESC"
            cursor.execute(sql)
            results = cursor.fetchall()
            return [Crumb(**row) for row in results]
```

```
    def update_crumb(self, crumb_id: str, crumb: Crumb) -> Optional
        [Crumb]:
            with self.connection.cursor() as cursor:
                sql = """
                    UPDATE crumbs
                    SET content = %s,
                        image_url = %s,
                        created_at = created_at
                    WHERE crumb_id = %s
                """
                cursor.execute(sql, (crumb.content, crumb.image_url,
                    crumb_id))
                if cursor.rowcount > 0:
                    return self.get_crumb(crumb_id)
            return None

    def delete_crumb(self, crumb_id: str) -> bool:
            with self.connection.cursor() as cursor:
                sql = "DELETE FROM crumbs WHERE crumb_id = %s"
                cursor.execute(sql, (crumb_id,))
                return cursor.rowcount > 0
```

2.3.1.4. crumb.py Define la clase `Crumb`, que representa un "crumb" en la aplicación Crumblr.

Listing 9: Modelo de Crumb en la aplicación Crumblr

```
from pydantic import BaseModel, Field, field_validator
from typing import Optional, List, Literal
from datetime import datetime
import uuid

class Crumb:
    def __init__(self, crumb_id=None, content='', image_url=None,
        created_at=None):
            self.crumb_id = crumb_id or str(uuid.uuid4())
            self.content = content
            self.image_url = image_url
            self.created_at = created_at or datetime.utcnow()
```

2.3.1.5. main.py Implementa la aplicación Flask de Crumblr, con rutas para CRUD de *crumbs*, manejo de errores y CORS.

Listing 10: Aplicación Flask para Crumblr — CRUD y Healthcheck

```
from flask import Flask, request, jsonify
from pydantic import ValidationError
import psycopg2
from models.crumb import Crumb
from db.factory import DatabaseFactory
```

```
app = Flask(__name__)

# Inicializar base de datos
try:
    db = DatabaseFactory.create()
except ValueError as e:
    raise RuntimeError(f"Error initializing DB: {e}") from e


# Middleware CORS
@app.after_request
def add_cors_headers(response):
    response.headers['Access-Control-Allow-Origin'] = '*'
    response.headers['Access-Control-Allow-Headers'] = 'Content-
       Type'
    response.headers['Access-Control-Allow-Methods'] = 'GET,POST,
       PUT,DELETE,OPTIONS'
    return response


# Crear un nuevo crumb
@app.route('/crumbs', methods=['POST'])
def create_crumb():
    try:
        data = request.get_json()
        crumb = Crumb(**data)
        created = db.create_crumb(crumb)
        return jsonify(created.__dict__), 201
    except ValidationError as e:
        return jsonify({'error': 'Validation error', 'details': e.
           errors()}), 400
    except psycopg2.IntegrityError as e:
        return jsonify({'error': 'Database integrity error', '
           details': str(e)}), 409
    except psycopg2.OperationalError as e:
        return jsonify({'error': 'Database connection error', '
           details': str(e)}), 503
    except psycopg2.Error as e:
        return jsonify({'error': 'Database error', 'details': str(e
           )}), 500


# Obtener un crumb por ID
@app.route('/crumbs/<crumb_id>', methods=['GET'])
def get_crumb(crumb_id):
    try:
        crumb = db.get_crumb(crumb_id)
        if crumb:
            return jsonify(crumb.__dict__), 200
        return jsonify({'error': 'Crumb not found'}), 404
    except psycopg2.OperationalError as e:
```

```
        return jsonify({'error': 'Database connection error', '
            details': str(e)}), 503
    except psycopg2.Error as e:
        return jsonify({'error': 'Database error', 'details': str(e
            )}), 500


# Obtener todos los crumbs
@app.route('/crumbs', methods=['GET'])
def get_all_crumbs():
    try:
        crumbs = db.get_all_crumbs()
        return jsonify([c.__dict__ for c in crumbs]), 200
    except psycopg2.OperationalError as e:
        return jsonify({'error': 'Database connection error', '
            details': str(e)}), 503
    except psycopg2.Error as e:
        return jsonify({'error': 'Database error', 'details': str(e
            )}), 500


# Actualizar un crumb
@app.route('/crumbs/<crumb_id>', methods=['PUT'])
def update_crumb(crumb_id):
    try:
        data = request.get_json()
        data.pop('crumb_id', None)
        data.pop('created_at', None)
        crumb = Crumb(**data)
        updated = db.update_crumb(crumb_id, crumb)
        if updated:
            return jsonify(updated.__dict__), 200
        return jsonify({'error': 'Crumb not found'}), 404
    except ValidationError as e:
        return jsonify({'error': 'Validation error', 'details': e.
            errors()}), 400
    except psycopg2.IntegrityError as e:
        return jsonify({'error': 'Database integrity error', '
            details': str(e)}), 409
    except psycopg2.OperationalError as e:
        return jsonify({'error': 'Database connection error', '
            details': str(e)}), 503
    except psycopg2.Error as e:
        return jsonify({'error': 'Database error', 'details': str(e
            )}), 500


# Eliminar un crumb
@app.route('/crumbs/<crumb_id>', methods=['DELETE'])
def delete_crumb(crumb_id):
    try:
```

```
        if db.delete_crumb(crumb_id):
            return '', 204
        return jsonify({'error': 'Crumb not found'}), 404
    except psycopg2.OperationalError as e:
        return jsonify({'error': 'Database connection error', '
            details': str(e)}), 503
    except psycopg2.Error as e:
        return jsonify({'error': 'Database error', 'details': str(e
            )}), 500


# Healthcheck
@app.route('/health', methods=['GET'])
def health():
    return jsonify({'status': 'healthy'}), 200


if __name__ == '__main__':
    app.run(host='0.0.0.0', port=8080)
```

2.3.2. Dockerfile

El dockerfile se mantiene idéntico al original con tickets.

Listing 11: Dockerfile para el backend de Crumblr

```
FROM python:3.11-slim

WORKDIR /app

COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

COPY Crumblr-Back/ .

EXPOSE 8080

CMD ["python", "main.py"]
```

3. FrontEnd

El frontend de crumblr lo cree con ayuda de chatgpt en html y js con despliegue alpine en docker. Cabe destacar que el frontend es el mismo para la versión acoplada y desacoplada ya que me ahorraba tiempo y no podía crear más lambdas.

El código del frontend es el siguiente:

3.1. Código

3.1.1. app.js

Código JavaScript de la aplicación frontend que se comunica con la API de Crumblr para CRUD de crumbs.

```
const API_URL =
   "https://v4vbkmeftj.execute-api.us-east-1.amazonaws.com/prod/crumbs";
const API_KEY = "x3SV9yBfwuWJ3M3NyT3E541ykzkEWdN9jrMj40ah";

async function fetchCrumbs() {
  const res = await fetch(API_URL, {
    headers: { "x-api-key": API_KEY }
  });

  if (!res.ok) {
    alert("Error al obtener los crumbs");
    return;
  }

  const data = await res.json();
  const list = document.getElementById("crumbs-list");
  list.innerHTML = "";

  data.forEach(crumb => {
    const li = document.createElement("li");
    li.innerHTML = `
      <div id="crumb-${crumb.crumb_id}">
        <p class="content">${crumb.content}</p>
        ${crumb.image_url ? `<img src="${crumb.image_url}"
           alt="Crumb image">` : ""}
        <small>${new
           Date(crumb.created_at).toLocaleString()}</small>
      </div>
      <button onclick="editCrumb('${crumb.crumb_id}',
         '${crumb.content.replace(/'/g, "\\'")}',
         '${crumb.image_url || ""}')"></button>
      <button onclick="deleteCrumb('${crumb.crumb_id}')"></button>
    `;
    list.appendChild(li);
  });
}

async function createCrumb(e) {
```

```
34    e.preventDefault();
35    const content = document.getElementById("content").value.trim();
36    const image_url =
          document.getElementById("image_url").value.trim();
37
38    if (!content) return alert("El contenido no puede estar vacío");
39
40    const res = await fetch(API_URL, {
41      method: "POST",
42      headers: {
43        "Content-Type": "application/json",
44        "x-api-key": API_KEY
45      },
46      body: JSON.stringify({ content, image_url })
47    });
48
49    if (res.ok) {
50      document.getElementById("create-form").reset();
51      fetchCrumbs();
52    } else {
53      alert("Error al crear el crumb");
54    }
55  }
56
57  async function deleteCrumb(id) {
58    const res = await fetch(`${API_URL}/${id}`, {
59      method: "DELETE",
60      headers: { "x-api-key": API_KEY }
61    });
62
63    if (res.ok) fetchCrumbs();
64    else alert("Error al eliminar el crumb");
65  }
66
67  function editCrumb(id, currentContent, currentImage) {
68    const newContent = prompt("Editá el contenido del crumb:",
          currentContent);
69    if (newContent === null) return;
70
71    const newImage = prompt("Editá la URL de imagen (o dejá
          vacío):", currentImage);
72    updateCrumb(id, newContent, newImage);
73  }
74
75  async function updateCrumb(id, content, image_url) {
76    const res = await fetch(`${API_URL}/${id}`, {
77      method: "PUT",
78      headers: {
79        "Content-Type": "application/json",
80        "x-api-key": API_KEY
81      },
```

```
    body: JSON.stringify({ content, image_url })
  });

  if (res.ok) {
    fetchCrumbs();
  } else {
    alert("Error al actualizar el crumb");
  }
}

document.getElementById("create-form").addEventListener("submit",
    createCrumb);
fetchCrumbs();
```

Listing 12: Frontend JS para Crumbs

Un detalle importante es que a la hora de lanzar este código es necesario cambiar la URL de la API y la API KEY por las correspondientes a la implementación del backend. A pesar de haber implementado apiurl y apikey en el yaml del frontend, no conseguí cargarlos correctamente y decidí dejarlos hardcodeados para llegar a la entrega.

3.1.2. index.html

Listing 13: Interfaz web de Crumblr

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <title>Crumblr</title>
  <link rel="stylesheet" href="style.css">
</head>
<body>
  <h1>Crumblr</h1>

  <section id="create">
    <h2>Nuevo Crumb</h2>
    <form id="create-form">
      <textarea id="content" placeholder="Que estas pensando?"
          required></textarea>
      <input type="text" id="image_url" placeholder="URL de imagen
          (opcional)">
      <button type="submit">Publicar</button>
    </form>
  </section>

  <section id="list">
    <h2>Crumbs recientes</h2>
    <ul id="crumbs-list"></ul>
  </section>

  <script src="app.js"></script>
```

```
</body>
</html>
```

3.1.3. style.css

Listing 14: Estilos CSS para Crumblr

```
    body {
  font-family: system-ui, sans-serif;
  background: #fafafa;
  color: #333;
  padding: 2rem;
  max-width: 600px;
  margin: auto;
}

h1 {
  text-align: center;
  color: #222;
}

form {
  display: flex;
  flex-direction: column;
  gap: 0.5rem;
  margin-bottom: 2rem;
}

textarea, input, button {
  padding: 0.75rem;
  border-radius: 10px;
  border: 1px solid #ccc;
  font-size: 1rem;
}

button {
  background: #ffb703;
  color: #222;
  font-weight: bold;
  border: none;
  cursor: pointer;
}

button:hover {
  background: #fb8500;
}

ul {
  list-style: none;
  padding: 0;
}
```

```
li {
  background: white;
  border-radius: 10px;
  margin-bottom: 1rem;
  padding: 1rem;
  box-shadow: 0 2px 5px rgba(0,0,0,0.1);
  display: flex;
  justify-content: space-between;
  align-items: center;
}

li img {
  max-width: 100%;
  border-radius: 10px;
  margin-top: 0.5rem;
}

small {
  color: #888;
  display: block;
  margin-top: 0.25rem;
  font-size: 0.8rem;
}
```

3.2. dockerfile

De nuevo, intento mantenerlo simple con alpine.

Listing 15: Dockerfile para Frontend

```
FROM nginx:stable-alpine

RUN rm -rf /usr/share/nginx/html/*

COPY . /usr/share/nginx/html

COPY nginx.conf /etc/nginx/conf.d/default.conf

EXPOSE 80

CMD ["nginx", "-g", "daemon off;"]
```

4. Desacoplado

Modelo de integración de los servicios de backend de forma desacoplada. Para hacer esto cree un nuevo *main.yml* que define una serie de lambdas a las cuales se lanza mediante eventos generados por la *API Gateway*.

El frontend se mantiene exacto al definido en: 2.2.3.2

Las tareas que se realizan son las siguientes:

- Lanzamiento de los repositorios ECR para las lambdas.
- Subida de las imágenes docker de las lambdas a los repositorios ECR (codigo modificado en **??**).
- Despliegue de la base de datos RDS PostgreSQL.
- Configuración de las Lambdas y sus respectivos permisos.
- Definición de la API Gateway para enrutar las peticiones HTTP a las Lambdas correspondientes.
- Ajustes de CORS en API Gateway para permitir peticiones desde el dominio del frontend.
- Lanzamiento del frontend.

4.1. Diagrama de arquitectura

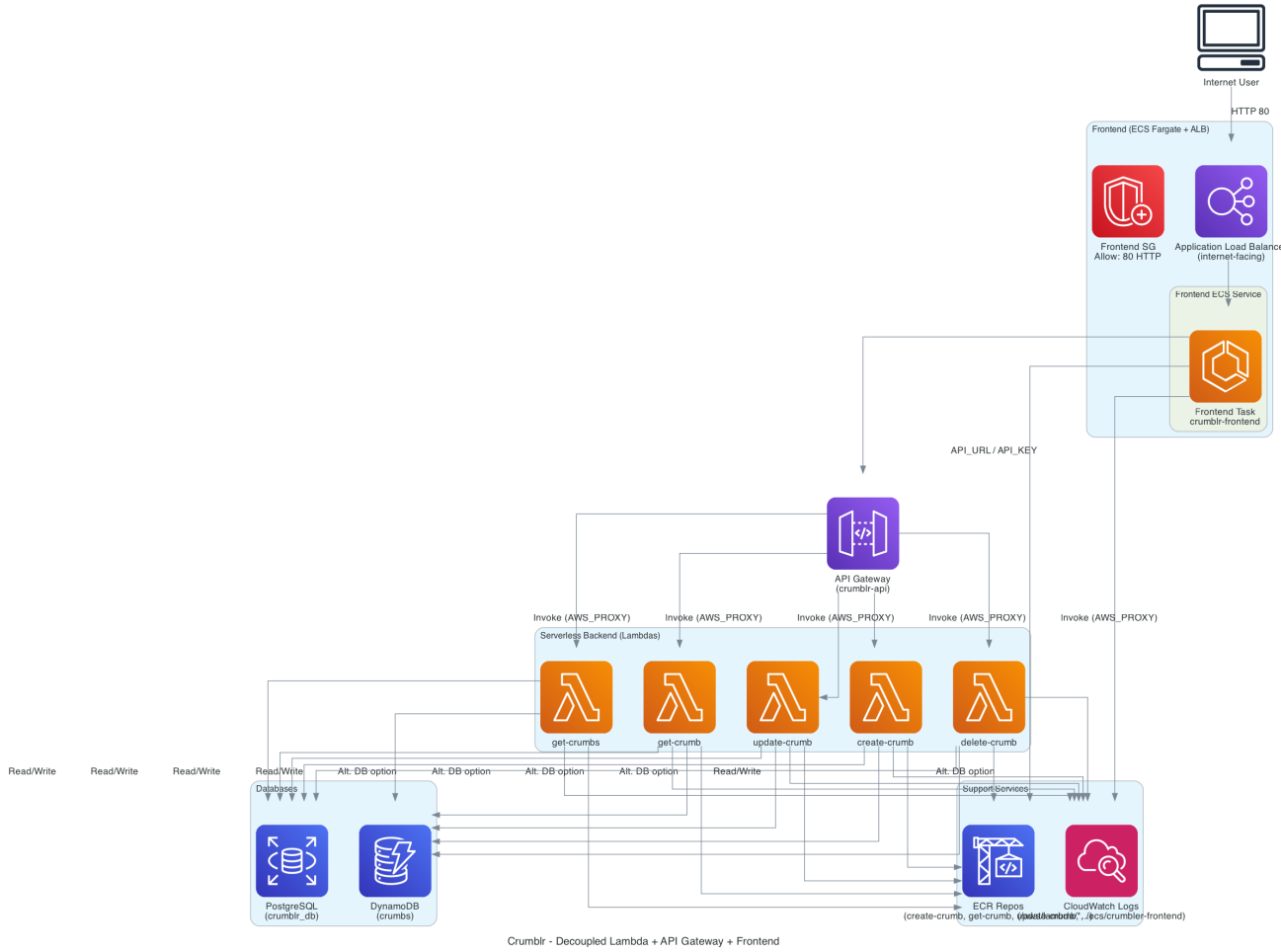

Figura 2: Diagrama de la arquitectura de Crumblr

4.2. Explicación de los templates de CloudFormation

4.2.1. ECR

4.2.1.1. BUILDSLLECR.yml - Repositorios ECR para las Lambdas de Crumblr

Debido al aumento del número de ECR y la complejidad de sincronización del código, se eliminó la mayor parte de opciones de personalización del YAML para evitar problemas. Este YAML define los repositorios ECR que serán utilizados por las funciones Lambda de Crumblr, cada uno con su política de ciclo de vida para mantener solo las últimas dos imágenes.

Listing 16: Repositorios ECR para las Lambdas de Crumblr

```
AWSTemplateFormatVersion: "2010-09-09"
```

```
Description: "Crumblr Lambda Repositories"

Resources:
  ECRRepository1:
    Type: AWS::ECR::Repository
    Properties:
      RepositoryName: get-crumbs
      ImageScanningConfiguration:
        ScanOnPush: false
      LifecyclePolicy:
        LifecyclePolicyText: |
          {
            "rules": [
              {
                "rulePriority": 1,
                "description": "Mantain only last 2 images",
                "selection": {
                  "tagStatus": "any",
                  "countType": "imageCountMoreThan",
                  "countNumber": 2
                },
                "action": { "type": "expire" }
              }
            ]
          }
  ECRRepository2:
    Type: AWS::ECR::Repository
    Properties:
      RepositoryName: get-crumb
      ImageScanningConfiguration:
        ScanOnPush: false
      LifecyclePolicy:
        LifecyclePolicyText: |
          {
            "rules": [
              {
                "rulePriority": 1,
                "description": "Mantain only last 2 images",
                "selection": {
                  "tagStatus": "any",
                  "countType": "imageCountMoreThan",
                  "countNumber": 2
                },
                "action": { "type": "expire" }
              }
            ]
          }
  ECRRepository3:
    Type: AWS::ECR::Repository
    Properties:
      RepositoryName: delete-crumb
```

```
    ImageScanningConfiguration:
      ScanOnPush: false
    LifecyclePolicy:
      LifecyclePolicyText: |
        {
          "rules": [
            {
              "rulePriority": 1,
              "description": "Mantain only last 2 images",
              "selection": {
                "tagStatus": "any",
                "countType": "imageCountMoreThan",
                "countNumber": 2
              },
              "action": { "type": "expire" }
            }
          ]
        }
ECRRepository4:
  Type: AWS::ECR::Repository
  Properties:
    RepositoryName: update-crumb
    ImageScanningConfiguration:
      ScanOnPush: false
    LifecyclePolicy:
      LifecyclePolicyText: |
        {
          "rules": [
            {
              "rulePriority": 1,
              "description": "Mantain only last 2 images",
              "selection": {
                "tagStatus": "any",
                "countType": "imageCountMoreThan",
                "countNumber": 2
              },
              "action": { "type": "expire" }
            }
          ]
        }
ECRRepository5:
  Type: AWS::ECR::Repository
  Properties:
    RepositoryName: create-crumb
    ImageScanningConfiguration:
      ScanOnPush: false
    LifecyclePolicy:
      LifecyclePolicyText: |
        {
          "rules": [
            {
```

```
            "rulePriority": 1,
            "description": "Mantain only last 2 images",
            "selection": {
              "tagStatus": "any",
              "countType": "imageCountMoreThan",
              "countNumber": 2
            },
            "action": { "type": "expire" }
          }
        ]
      }
```

4.2.1.2. Crumblr-ecr-gui - Repositorio ECR para el Frontend de Crumblr

Ya descrito en: 2.2.1.2.

4.2.2. Bases de datos

4.2.2.1. db-postgres Se mantiene la misma que en el acoplado, explicado en la sección: 2.2.2.1.

4.2.3. Lanzamineto de Aplicación

4.2.3.1. main.yml - Despliegue del Backend de Crumblr Esta plantilla define la infraestructura de Crumblr en AWS, incluyendo funciones Lambda, API Gateway, PostgreSQL, CORS y CloudWatch Logs.

Listing 17: Infraestructura de Crumblr: Lambdas, API Gateway y bases de datos

```
AWSTemplateFormatVersion: "2010-09-09"
Description: "Crumblr -> AWS Lambda + API Gateway + PostgreSQL/
   DynamoDB with CORS and CloudWatch Logs"

Parameters:
  VpcId:
    Type: AWS::EC2::VPC::Id
    Description: Target VPC
  SubnetIds:
    Type: List<AWS::EC2::Subnet::Id>
    Description: At least 2 subnets in different AZs
  DBType:
    Type: String
    Default: postgres
    AllowedValues:
      - postgres
      - dynamodb
  DBHost:
    Type: String
    Default: ""
    Description: "DB Host (only used for PostgreSQL)"
  DBName:
    Type: String
    Default: "crumblr_db"
```

```
  DBUser:
    Type: String
    Default: "postgres"
  DBPass:
    Type: String
    NoEcho: true
    Default: "Nicololo"
  DBDynamoName:
    Type: String
    Default: "crumbs"

Resources:
  # Funciones Lambda
  CreateCrumbLambda:
    Type: AWS::Lambda::Function
    Properties:
      FunctionName: create-crumb
      PackageType: Image
      Code:
        ImageUri: !Sub "${AWS::AccountId}.dkr.ecr.${AWS::Region}.
           amazonaws.com/create-crumb:latest"
      Environment:
        Variables:
          DB_TYPE: !Ref DBType
          DB_HOST: !Ref DBHost
          DB_NAME: !Ref DBName
          DB_USER: !Ref DBUser
          DB_PASS: !Ref DBPass
          DB_DYNAMONAME: !Ref DBDynamoName
  GetCrumbLambda: {}
  GetCrumbsLambda: {}
  UpdateCrumbLambda: {}
  DeleteCrumbLambda: {}

  # Logs en CloudWatch
  CreateCrumbLogGroup: {}
  GetCrumbLogGroup: {}
  GetCrumbsLogGroup: {}
  UpdateCrumbLogGroup: {}
  DeleteCrumbLogGroup: {}

  # API Gateway
  RestAPI:
    Type: AWS::ApiGateway::RestApi
    Properties:
      Name: crumblr-api

  CrumbsResource:
    Type: AWS::ApiGateway::Resource
  CrumbResource:
    Type: AWS::ApiGateway::Resource
```

```
  # Métodos API y permisos Lambda
  PostCrumbs: {}
  GetCrumbs: {}
  GetCrumb: {}
  PutCrumb: {}
  DeleteCrumb: {}
  OptionsCrumbs: {}
  OptionsCrumb: {}

  # Deployment y Stage
  APIDeployment: {}
  APIStage: {}
  APIKey: {}
  UsagePlan: {}
  UsagePlanKey: {}

Outputs:
  APIEndpoint:
    Value: !Sub "https://${RestAPI}.execute-api.${AWS::Region}.
       amazonaws.com/prod"
  APIKeyId: !Ref APIKey
  GetAPIKeyCommand: !Sub "aws apigateway get-api-key --api-key ${
     APIKey} --include-value --query 'value' --output text"
```

Explicación general:

- La plantilla define **parámetros** para la VPC, subnets y detalles de la base de datos (solo probado e implementado para PostgreSQL).
- Se crean **funciones Lambda** para cada operación CRUD: crear, obtener (uno o todos), actualizar y eliminar *crumbs*.
- Cada Lambda tiene su propio **grupo de logs** en CloudWatch.
- Se configura un **API Gateway REST** con rutas para /crumbs y /crumbs/id, incluyendo métodos HTTP y soporte para CORS.
- Se genera una **API Key** con un plan de uso asociado para controlar el acceso.
- Los **outputs** permiten obtener la URL de la API y la id de la clave de acceso.

4.2.3.2. frontEC2Cluster.yml - Despliegue del Frontend

Ya descrito en: 2.2.3.2.

4.3. Código

Los cambios que se realizaron al código residen principalmente en la implementación de un handler específico para cada operación. Sin embargo, se reutiliza todo lo que se encuentra en shared (modelo e inicialización) ya que no podía lanzar otra lambda por restricciones del laboratorio y no puedo saber cuál será la primera operación que realizará el backend.

4.3.1. Handlers

Listing 18: Handler para crear un crumb

```
    import json
import logging
from datetime import datetime
from shared.services.crumb_service import CrumbService

logger = logging.getLogger()
logger.setLevel(logging.INFO)

service = CrumbService()

def handler(event, context):
    logger.info(f"Received event: {json.dumps(event)}")
    try:
        data = json.loads(event.get('body', '{}'))
        crumb = service.create_crumb(data)

        # Convertir el objeto Crumb a dict serializable
        crumb_dict = {
            'crumb_id': crumb.crumb_id,
            'content': crumb.content,
            'image_url': crumb.image_url,
            'created_at': crumb.created_at.isoformat() if
               isinstance(crumb.created_at, datetime) else crumb.
               created_at
        }

        response = {
            "statusCode": 201,
            "headers": {
                "Access-Control-Allow-Origin": "*",
                "Access-Control-Allow-Headers": "Content-Type,x-api
                   -key",
                "Access-Control-Allow-Methods": "GET,POST,PUT,
                   DELETE,OPTIONS"
            },
            "body": json.dumps(crumb_dict)
        }
        logger.info(f"Response: {response}")
        return response
    except Exception as e:
```

```
        logger.exception("Error creating crumb")
        return {
            "statusCode": 500,
            "headers": {
                "Access-Control-Allow-Origin": "*",
                "Access-Control-Allow-Headers": "Content-Type,x-api
                    -key",
                "Access-Control-Allow-Methods": "GET,POST,PUT,
                    DELETE,OPTIONS"
            },
            "body": json.dumps({"error": str(e)})
        }
```

Listing 19: Handler para obtener todos los crumbs

```
import json
import logging
from datetime import datetime
from shared.services.crumb_service import CrumbService

logger = logging.getLogger()
logger.setLevel(logging.INFO)

service = CrumbService()

def handler(event, context):
    logger.info(f"Received event: {json.dumps(event)}")
    try:
        crumbs = service.get_all_crumbs()

        # Convertir lista de crumbs a lista de dicts serializables
        crumbs_list = []
        for crumb in crumbs:
            crumbs_list.append({
                'crumb_id': crumb.crumb_id,
                'content': crumb.content,
                'image_url': crumb.image_url,
                'created_at': crumb.created_at.isoformat() if
                    isinstance(crumb.created_at, datetime) else
                    crumb.created_at
            })

        response = {
            "statusCode": 200,
            "headers": {
                "Access-Control-Allow-Origin": "*",
                "Access-Control-Allow-Headers": "Content-Type,x-api
                    -key",
                "Access-Control-Allow-Methods": "GET,POST,PUT,
                    DELETE,OPTIONS"
```

```
            },
            "body": json.dumps(crumbs_list)
        }
        return response
    except Exception as e:
        logger.exception("Error getting crumbs")
        return {
            "statusCode": 500,
            "headers": {
                "Access-Control-Allow-Origin": "*",
                "Access-Control-Allow-Headers": "Content-Type,x-api
                    -key",
                "Access-Control-Allow-Methods": "GET,POST,PUT,
                    DELETE,OPTIONS"
            },
            "body": json.dumps({"error": str(e)})
        }
```

Listing 20: Handler para eliminar un crumb

```
import json
import logging
from shared.services.crumb_service import CrumbService

# Configurar logging
logger = logging.getLogger()
logger.setLevel(logging.INFO)

service = CrumbService()

def handler(event, context):
    logger.info(f"Received event: {json.dumps(event)}")
    try:
        crumb_id = event['pathParameters']['id']
        service.delete_crumb(crumb_id)
        response = {
            "statusCode": 200,
            "headers": {
                "Access-Control-Allow-Origin": "*",
                "Access-Control-Allow-Headers": "Content-Type,x-api
                    -key",
                "Access-Control-Allow-Methods": "GET,POST,PUT,
                    DELETE,OPTIONS"
            },
            "body": json.dumps({"message": "Crumb deleted
                successfully"})
        }
        logger.info(f"Response: {response}")
        return response
    except ValueError as e:
```

```
        logger.warning(f"Crumb not found: {e}")
        return {
            "statusCode": 404,
            "headers": {
                "Access-Control-Allow-Origin": "*",
                "Access-Control-Allow-Headers": "Content-Type,x-api
                    -key",
                "Access-Control-Allow-Methods": "GET,POST,PUT,
                    DELETE,OPTIONS"
            },
            "body": json.dumps({"error": str(e)})
        }
    except Exception as e:
        logger.exception("Error deleting crumb")
        return {
            "statusCode": 500,
            "headers": {
                "Access-Control-Allow-Origin": "*",
                "Access-Control-Allow-Headers": "Content-Type,x-api
                    -key",
                "Access-Control-Allow-Methods": "GET,POST,PUT,
                    DELETE,OPTIONS"
            },
            "body": json.dumps({"error": str(e)})
        }
```

Listing 21: Handler para obtener un crumb por ID

```
import json
import logging
from datetime import datetime
from shared.services.crumb_service import CrumbService

logger = logging.getLogger()
logger.setLevel(logging.INFO)

service = CrumbService()

def handler(event, context):
    logger.info(f"Received event: {json.dumps(event)}")
    try:
        crumb_id = event['pathParameters']['id']
        crumb = service.get_crumb(crumb_id)

        crumb_dict = {
            'crumb_id': crumb.crumb_id,
            'content': crumb.content,
            'image_url': crumb.image_url,
            'created_at': crumb.created_at.isoformat() if
                isinstance(crumb.created_at, datetime) else crumb.
```

```
                created_at
            }

            response = {
                "statusCode": 200,
                "headers": {
                    "Access-Control-Allow-Origin": "*",
                    "Access-Control-Allow-Headers": "Content-Type,x-api
                        -key",
                    "Access-Control-Allow-Methods": "GET,POST,PUT,
                        DELETE,OPTIONS"
                },
                "body": json.dumps(crumb_dict)
            }
            return response
        except ValueError as e:
            return {
                "statusCode": 404,
                "headers": {
                    "Access-Control-Allow-Origin": "*",
                    "Access-Control-Allow-Headers": "Content-Type,x-api
                        -key",
                    "Access-Control-Allow-Methods": "GET,POST,PUT,
                        DELETE,OPTIONS"
                },
                "body": json.dumps({"error": str(e)})
            }
        except Exception as e:
            logger.exception("Error getting crumb")
            return {
                "statusCode": 500,
                "headers": {
                    "Access-Control-Allow-Origin": "*",
                    "Access-Control-Allow-Headers": "Content-Type,x-api
                        -key",
                    "Access-Control-Allow-Methods": "GET,POST,PUT,
                        DELETE,OPTIONS"
                },
                "body": json.dumps({"error": str(e)})
            }
```

Listing 22: Handler para actualizar un crumb

```
import json
import logging
from datetime import datetime
from shared.services.crumb_service import CrumbService

# Configurar logging
logger = logging.getLogger()
```

```
logger.setLevel(logging.INFO)

service = CrumbService()

def handler(event, context):
    logger.info(f"Received event: {json.dumps(event)}")
    try:
        crumb_id = event['pathParameters']['id']
        data = json.loads(event['body'])
        updated = service.update_crumb(crumb_id, data)

        # Convertir el objeto Crumb a dict serializable
        crumb_dict = {
            'crumb_id': updated.crumb_id,
            'content': updated.content,
            'image_url': updated.image_url,
            'created_at': updated.created_at.isoformat() if
               isinstance(updated.created_at, datetime) else
               updated.created_at
        }

        response = {
            "statusCode": 200,
            "headers": {
                "Access-Control-Allow-Origin": "*",
                "Access-Control-Allow-Headers": "Content-Type,x-api
                   -key",
                "Access-Control-Allow-Methods": "GET,POST,PUT,
                   DELETE,OPTIONS"
            },
            "body": json.dumps(crumb_dict)
        }
        logger.info(f"Response: {response}")
        return response
    except ValueError as e:
        logger.warning(f"Crumb not found: {crumb_id}")
        return {
            "statusCode": 404,
            "headers": {
                "Access-Control-Allow-Origin": "*",
                "Access-Control-Allow-Headers": "Content-Type,x-api
                   -key",
                "Access-Control-Allow-Methods": "GET,POST,PUT,
                   DELETE,OPTIONS"
            },
            "body": json.dumps({"error": str(e)})
        }
    except Exception as e:
        logger.exception("Error updating crumb")
        return {
            "statusCode": 500,
```

```
            "headers": {
                "Access-Control-Allow-Origin": "*",
                "Access-Control-Allow-Headers": "Content-Type,x-api
                    -key",
                "Access-Control-Allow-Methods": "GET,POST,PUT,
                    DELETE,OPTIONS"
            },
            "body": json.dumps({"error": str(e)})
        }
```

Cada uno de estos handlers se despliega como una función Lambda independiente y realiza la acción de su nombre.

4.3.2. Shared

Código necesario compartido entre los distintos handlers, incluyendo el modelo de datos y la inicialización del servicio de acceso a datos.

4.3.2.1. crumb.py El mismo que en la sección 2.3.1.4.

4.3.2.2. shared/*.py los mismos que en las secciones 2.3.1.1, 2.3.1.2 y 2.3.1.3.

4.3.2.3. shared/crumb-service Implementa la lógica de negocio para los crumbs, utilizando el patrón de diseño Service.

Listing 23: Servicio de Crumb de Crumblr

```
    from shared.db.factory import DatabaseFactory
from shared.models.crumb import Crumb

class CrumbService:
    def __init__(self):
        self.db = DatabaseFactory.create()

    def create_crumb(self, data: dict) -> Crumb:
        crumb = Crumb(**data)
        return self.db.create_crumb(crumb)

    def get_crumb(self, crumb_id: str) -> Crumb:
        crumb = self.db.get_crumb(crumb_id)
        if not crumb:
            raise ValueError("Crumb not found")
        return crumb

    def get_all_crumbs(self):
        return self.db.get_all_crumbs()

    def update_crumb(self, crumb_id: str, data: dict) -> Crumb:
        data.pop('crumb_id', None)
        data.pop('created_at', None)
        crumb = Crumb(**data)
```

```
        updated = self.db.update_crumb(crumb_id, crumb)
        if not updated:
            raise ValueError("Crumb not found")
        return updated

    def delete_crumb(self, crumb_id: str) -> bool:
        deleted = self.db.delete_crumb(crumb_id)
        if not deleted:
            raise ValueError("Crumb not found")
        return deleted
```

Soy consciente de que se trata de un código monolítico y que podría haberlo separado para que cada handler tuviera su propio servicio, pero por limitaciones de tiempo y para evitar redundancias he optado por esta solución ya que el build lo realizaba copiando la carpeta de shared en las diferentes lambdas con un mismo dockerfile para todas las lambdas.

4.3.3. dockerfile

Cada servicio lambda se construye utilizando el siguiente Dockerfile, que instala las dependencias necesarias y copia el código fuente.

Listing 24: Dockerfile para las funciones Lambda de Crumblr

```
# Base image oficial de AWS Lambda para Python 3.11
FROM public.ecr.aws/lambda/python:3.11

# Copiar código de la Lambda
COPY app.py ${LAMBDA_TASK_ROOT}/

# Copiar código compartido
COPY ./shared ${LAMBDA_TASK_ROOT}/shared

# Crear __init__.py en todas las carpetas para que Python las
   reconozca como módulos
RUN touch ${LAMBDA_TASK_ROOT}/shared/__init__.py && \
    touch ${LAMBDA_TASK_ROOT}/shared/db/__init__.py && \
    touch ${LAMBDA_TASK_ROOT}/shared/models/__init__.py && \
    touch ${LAMBDA_TASK_ROOT}/shared/services/__init__.py

# Instalar dependencias
RUN pip install --no-cache-dir pydantic psycopg2-binary

# Comando de arranque para Lambda
CMD ["app.handler"]
```

- **Imagen base:** Se utiliza `public.ecr.aws/lambda/python:3.11`, que ya incluye el runtime de Python 3.11 preparado para AWS Lambda.
- **Copiado del código:** Se copia el archivo principal `app.py` al directorio de trabajo de Lambda (`$LAMBDA_TASK_ROOT`) y, además, se copia el código compartido ubicado en `./shared` para poder ser utilizado por todas las funciones.

- **Módulos Python:** Se crean los archivos `__init__.py` en cada subcarpeta del código compartido (`shared`, `db`, `models`, `services`) para que Python reconozca estos directorios como paquetes y se pueda importar su contenido correctamente.
- **Instalación de dependencias:** Se instalan las librerías necesarias con `pip`, en este caso `pydantic` para validación de modelos y `psycopg2-binary` para conexión con PostgreSQL, usando la opción `-no-cache-dir` para no guardar la cache de instalación y mantener la imagen ligera.
- **Comando de arranque:** Finalmente, se define el `CMD` de la imagen, que indica a Lambda cuál es el manejador principal de la función, en este caso `app.handler`.

5. Presupuestos

Los presupuestos para la implementación de la arquitectura desacoplada de Crumblr se dividen en varias categorías:

- **Costos de frontend:**
- **Costos de backend acoplado:**
- **Costos de backend desacoplado:**

5.1. Costo acoplado

Cuadro 1: Precios estimados de AWS para los servicios usados en Crumblr (us-east-1)

Servicio	**Parámetro**	**Precio**	**Notas**
ECR	1 GB/mes	$0.10/GB-mes	No incluye transferencias fuera de región.
Fargate	1 vCPU + 2 GB/h	$0.04/vCPU-h + $0.0044/GB-h	Task 0.25 vCPU y 0.5 GB: $0.011/h.
RDS PostgreSQL	db.t3.micro	$0.0410/h (db.m5.xlarge)	Depende de instancia, almacenamiento, Multi-AZ y backups.

Costo mensual estimado: 37.54 USD
Costo anual estimado: 450.48 USD

5.2. Costo desacoplado

Cuadro 2: Precios estimados de AWS para Crumblr (us-east-1)

Servicio	**Parámetro**	**Precio**	**Notas**
Lambda	512 MB, 1M invoc./mes	0.12 USD	Basado en uso típico.
API Gateway REST	1M llamadas/mes	3.50 USD	Precio base por millón de llamadas.
ECR	1 GB	0.10 USD	Solo almacenamiento.
RDS PostgreSQL	db.t3.micro, 20 GB	29 USD	On-Demand + almacenamiento.

Costo mensual estimado: 32.72 USD
Costo anual estimado: 392.64 USD

5.3. Costo Frontend

Costo mensual estimado: 18.61 USD
Costo anual estimado: 223.32 USD

Cuadro 3: Precios estimados de AWS para el frontend de Crumblr (us-east-1)

Servicio	**Parámetro**	**Precio**	**Notas**
Fargate	0.25 vCPU + 0.5 GB, 1 h	0.011 USD	Task única, por hora.
ALB	1 ALB + 1M LCU-mes	18 USD	LCU = Load Balancer Capacity Units.
ECR	1 GB	0.10 USD	Solo almacenamiento.
CloudWatch Logs	14 días retención	0.50 USD	Logs de contenedor.

5.4. Resumen de costos totales

Cuadro 4: Resumen de costos anuales por arquitectura

Arquitectura	**Costo Mensual**	**Costo Anual**
Backend Acoplado	37.54 USD	450.48 USD
Backend Desacoplado	32.72 USD	392.64 USD
Frontend	18.61 USD	223.32 USD
Total Acoplado + Frontend	**56.15 USD**	**673.80 USD**
Total Desacoplado + Frontend	**51.33 USD**	**615.96 USD**

6. Conclusiones

Para la aplicación Crumblr, la arquitectura desacoplada basada en AWS Lambda y API Gateway ofrece varias ventajas significativas sobre una arquitectura acoplada tradicional utilizando ECS Fargate. En primer lugar, el modelo desacoplado permite una mayor escalabilidad y flexibilidad, ya que cada función Lambda puede escalar de forma independiente según la demanda, lo que resulta en un uso más eficiente de los recursos y potencialmente menores costos operativos. Además, la gestión de la infraestructura se simplifica considerablemente, ya que AWS se encarga del aprovisionamiento y mantenimiento de las funciones Lambda, reduciendo la carga administrativa. Sin embargo, esta arquitectura también presenta desafíos, como la complejidad añadida en la gestión de múltiples funciones y la necesidad de diseñar cuidadosamente las interacciones entre ellas para evitar latencias innecesarias. Además, las limitaciones inherentes a Lambda, como el tiempo máximo de ejecución y el tamaño del paquete de despliegue, pueden restringir ciertos casos de uso. En resumen, la elección entre una arquitectura desacoplada y una acoplada dependerá de las necesidades específicas del proyecto, considerando factores como la escalabilidad, la complejidad operativa y los costos asociados.

7. Uso de la IA

El uso de herramientas de inteligencia artificial, como ChatGPT, ha sido fundamental en la elaboración de este informe y en el desarrollo del código asociado. Estas herramientas han facilitado la generación de fragmentos de código, la redacción de secciones del informe y la resolución de dudas técnicas, permitiendo un enfoque más eficiente y productivo en el trabajo realizado. Sin embargo, es importante destacar que la infraestructura y la lógica del proyecto han sido diseñadas por mí.

8. github

El código fuente completo del proyecto está disponible en el siguiente repositorio de GitHub: `https://github.com/NicolasReyAlonso/CN_Practica_4.git`.

9. Arreglos tras revisión

Tras ser evaluado, decidí cambiar el acceso a la base de datos y al backend a través de security groups.

9.1. Cambios en arquitectura

- Integración entera en un solo Yaml tanto para el acoplado como para el desacoplado. (A escepción del ECR).
- El security group de la base de datos ahora permite acceso solo desde el security group del backend (ECS Fargate o Lambdas).
- Arreglos en el frontend para que funcione sin necesidad de cambiar código.