Infraestructura para la Computación de Altas Prestaciones
Proyecto Final: AquaSenseCloud

Francisco Javier Mercader Martínez
Javier Moreno Pagán
Pablo Meseguer Domenech

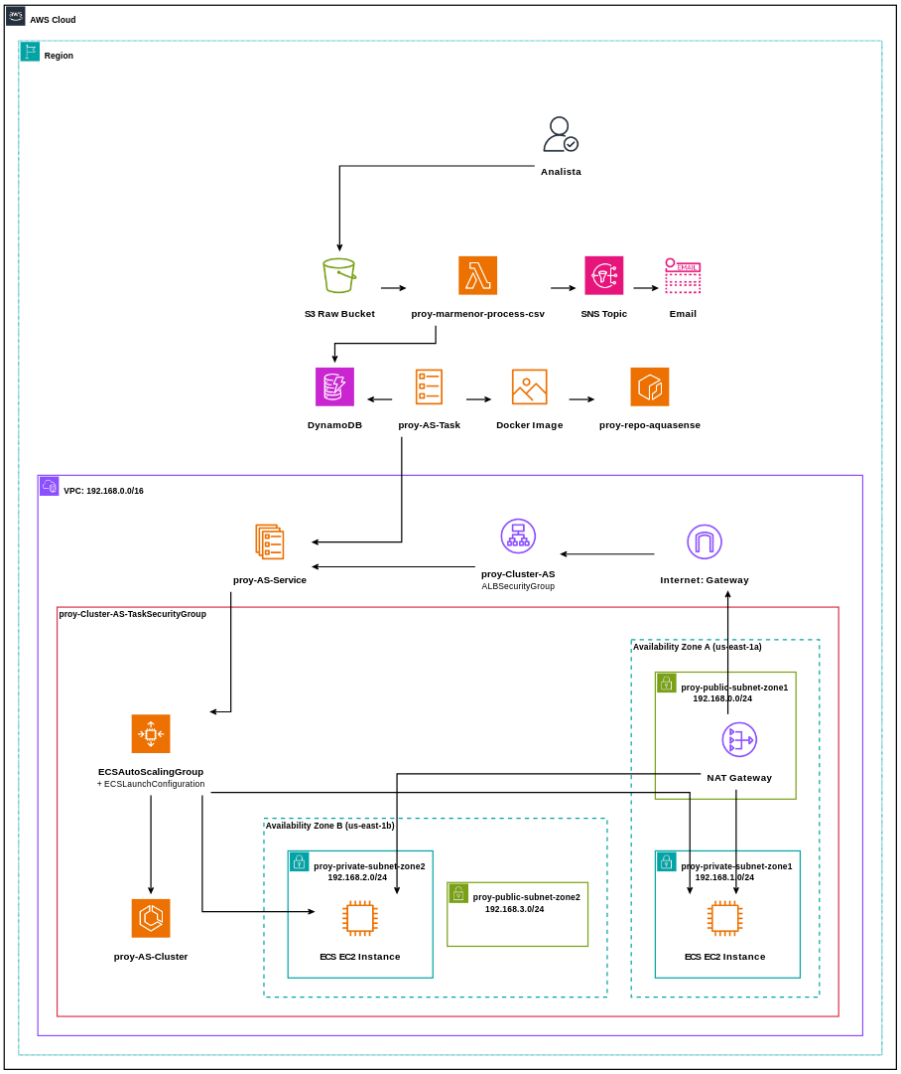
Curso: 2024/2025

Índice

1 Cronograma

Fecha	Tarea	Responsable	Duración
11 de noviembre	**Inicio y Diseño de Arquitectura:** Definición de requisitos, análisis del dataset CSV y diseño de la arquitectura completa en AWS y endpoints de la API.	Todo el Equipo	1 día
12-13 de noviembre	**Infraestructura de Red Base:** Creación de VPC, subnets, Security Groups, bucket S3 para datos crudos y tablas en DynamoDB.	Javier Moreno	2 días
14-15 de noviembre	**Lógica de Negocio y Pipeline:** Desarrollo de funciones Lambda para procesamiento de CSV y cálculos estadísticos (maxdiff, temp mensual).	Pablo Meseguer	2 días
16-17 de noviembre	**Configuración API Base:** Configuración del API Gateway y desarrollo de Lambdas para endpoints REST básicos (/sd, /temp).	Fco. Jav. Mercader	2 días
18-19 de noviembre	**Seguridad y Automatización:** Configuración de roles IAM, políticas de seguridad y creación de plantillas CloudFormation base (IaC).	Javier Moreno	2 días
20-21 de noviembre	**Sistema de Alarmas:** Configuración de SNS Topic, integración de alertas por desviación de datos y pruebas de envío de emails.	Pablo Meseguer	2 días
22-24 de noviembre	**Integración y Pruebas:** Integración de todas las Lambdas con DynamoDB, manejo de errores en API y creación de suite de pruebas en Postman.	Fco. Jav. Mercader	3 días
25-26 de noviembre	**Dockerización:** Creación de Dockerfile optimizado para la API, construcción de imagen y subida al repositorio ECR.	Pablo Meseguer	2 días
27-28 de noviembre	**Infraestructura Avanzada (ECS sobre EC2):** Configuración de Application Load Balancer (ALB), Target Groups y Listener, creación del clúster ECS (`Launch Type: EC2`), Task Definition EC2 y ECS Service.	Javier Moreno	2 días
29 de nov - 1 de dic	**Escalabilidad (capacidad EC2):** Configuración de ECSLaunchConfiguration y ECSAutoScalingGroup (`MinSize=2, MaxSize=5, DesiredCapacity=3`) como capacidad del clúster ECS, y validación de funcionamiento bajo carga.	Fco. Jav. Mercader	3 días
2 de diciembre	**Documentación Técnica:** Redacción de la memoria técnica, diagramas finales de arquitectura y justificación de decisiones.	Javier Moreno	1 día
3 de diciembre	**Manual de Replicación:** Elaboración de guía paso a paso para replicar el despliegue y scripts de limpieza de recursos.	Pablo Meseguer	1 día
4 de diciembre	**Pruebas Finales y Despliegue:** Despliegue final limpio en cuentas AWS Academy y validación "End-to-End" de todo el flujo.	Todo el Equipo	1 día
5 de diciembre	**Entrega Final:** Recopilación de entregables, preparación de material de defensa y envío del proyecto.	Todo el Equipo	1 día

2 Diagrama Arquitectura de la solución



3 Listado de Recursos y Servicios con su funcionalidad

1. Ingesta y almacenamiento

- **Amazon S3 (AWS::S3::Bucket):** La arquitectura utiliza dos buckets con propósitos diferenciados:
  - **Bucket de Datos (proy-marmenor-csv-raw):** Almacena los archivos CSV de datos brutos con las medias y desviaciones típicas de temperaturas. Actúa como el punto de entrada del pipeline, donde la carga de un nuevo archivo desencadena automáticamente la ejecución de la función Lambda.
  - **Bucket de Código (proy-marmenor-codebucket):** Repositorio de soporte utilizado para almacenar el código fuente comprimido (`.zip`) de la función Lambda y otros artefactos necesarios para el despliegue automatizado de la infraestructura mediante CloudFormation.

2. Pipeline de datos

- **AWS Lambda (AWS::Lambda::Function):** Ejecuta funciones que procesan los archivos almacenados en S3, transformando dichos datos y cargándolos en una tabla DynamoDB. Esos datos almacenados en la tabla ya están preparados para ser consultados.
- **Amazon SNS (AWS::SNS::Topic):** Se utiliza para enviar notificaciones cuando la desviación estándar semanal de las temperaturas supera el umbral de 0.5. Esta modificación se publica en el tema SNS, informando a los suscriptores (en este caso los analistas) sobre la situación crítica en el monitoreo de temperaturas.

3. Almacenamiento de datos procesados/transformados

- **DynamoDB (AWS::DynamoDB::Table):** Almacena todos los datos transformados y procesados, listos para ser consultados, proporcionando así un acceso rápido y escalable a los datos.

4. Recursos para el desarrollo y pruebas iniciales

- **Instancia EC2 (AWS::EC2::Instance):** Utilizada para desarrollar, probar y ajustar el contenedor con la aplicación AquaSense. En esta instancia se configuró el entorno de Docker y se realizaron pruebas locales antes de enviar la imagen al repositorio de ECR.

5. Infraestructura de red y conectividad

- **VPC (AWS::EC2::VPC):** Proporciona un entorno de red aislado donde se despliegan los recursos de la infraestructura.
- **Subnets (AWS::EC2::Subnet):** Se crean 2 subredes públicas y 2 subredes privadas distribuidas en distintas Availability Zones. Las subredes públicas alojan los componentes de entrada/salida a Internet (p. ej., NAT Gateway) y las privadas alojan los recursos de cómputo internos (ECS sobre EC2 y tareas en modo awsvpc).
- **Internet Gateway (AWS::EC2::InternetGateway):** Conecta la VPC con Internet, permitiendo la exposición del ALB.
- **NAT Gateway (AWS::EC2::NatGateway):** Proporciona salida a Internet a los recursos ubicados en subredes privadas sin exponerlos directamente. Se ubica en una subred pública y se utiliza como ruta por defecto (0.0.0.0/0) desde la tabla de rutas privada.
- **Route Table (AWS::EC2::RouteTable):** Define las rutas de tráfico dentro de la VPC. Incluye una tabla pública con salida al Internet Gateway y una tabla privada con salida al NAT Gateway, asociadas a sus respectivas subredes.

6. Almacenamiento de imágenes

- **ECR (Amazon Elastic Container Registry):** Repositorio para almacenar y versionar la imagen del contenedor de AquaSense después de desarrollarla en EC2. Este recurso nos facilita la integración con ECS para el despliegue en producción de nuestra aplicación.

7. Recursos para el despliegue de la aplicación

- **ECSCluster (AWS::ECS::Cluster):** Agrupa, administra y organiza todas las tareas y servicios relacionados con la ejecución del contenedor de la aplicación. Actúa como punto central de administración para la ejecución de tareas, asegurando que estas puedan desplegarse de manera distribuida y eficiente.
- **ALB (AWS::ElasticLoadBalancingV2::LoadBalancer):** Balanceador de carga que gestiona/balancea el tráfico de los analistas hacia las tareas ECS. Redirige las solicitudes desde el puerto 80 (tráfico HTTP) al puerto 5000 del contenedor, además, mejora la disponibilidad y la tolerancia a fallos de nuestra aplicación.
- **ALBListener (AWS::ElasticLoadBalancingV2::Listener):** Configura reglas en el balanceador de carga para redirigir las solicitudes hacia el grupo de destino, escucha el tráfico de los analistas a través del puerto 80.
- **ECSTargetGroup (AWS::ElasticLoadBalancingV2::TargetGroup):** Asocia las tareas ECS con el balanceador de carga y configura verificaciones de estado mediante el endpoint `/health` que ha ido configurado en nuestra aplicación. Este permite conocer el estado en que se encuentra las tareas para que, en caso de que alguan falle, se marca la tarea como no saludable y eso informa al ECS Service de que hay que lanzar nuevas tareas para mantener su número deseado.

8. Definición y gestión de tareas

- **ECSTaskDefinition (AWS::ECS::TaskDefinition):** Describe la definición de tarea para ejecutar el contenedor en ECS con compatibilidad EC2 (RequiresCompatibilities: EC2) y modo de red awsvpc. Especifica la imagen del contenedor en ECR, los recursos asignados (CPU y memoria), el mapeo de puertos (ContainerPort: 8080) y los roles (ExecutionRoleArn y TaskRoleArn). Es la plantilla base a partir de la cual ECS lanza las tareas.
- **ECSService (AWS::ECS::Service):** Administra la ejecución continua del número deseado de tareas (DesiredCount: 3) a partir de la Task Definition anterior, usando LaunchType: EC2. Integra el servicio con el ALB mediante el Target Group, y ejecuta las tareas en subredes privadas (AssignPublicIp: DISABLED) con los Security Groups importados del stack de red. La disponibilidad real del servicio depende de la capacidad aportada por el Auto Scaling Group de instancias EC2.

9. Escalado automático

- **ECSLaunchConfiguration (AWS::AutoScaling::LaunchConfiguration):** Define la configuración de las instancias EC2 que aportan capacidad al clúster ECS (AMI optimizada de ECS, tipo de instancia, Security Groups e Instance Profile). Incluye UserData para registrar automáticamente la instancia en el clúster ECS (`ECS_CLUSTER=...`).
- **ECSAutoScalingGroup (AWS::AutoScaling::AutoScalingGroup):** Proporciona la capacidad de cómputo al clúster ECS mediante un grupo de instancias EC2 desplegadas en subredes privadas. Se configura con MinSize=2, MaxSize=5 y DesiredCapacity=3. En este proyecto el escalado se realiza a nivel de instancias (capacidad del clúster), no a nivel de tareas mediante Application Auto Scaling.

10. Seguridad

- **ALB Security Group (AWS::EC2::SecurityGroup):** Permite el acceso al balanceador de carga desde Internet (a través del puerto 80-HTTP), asegurando que el tráfico pueda alcanzar la aplicación.

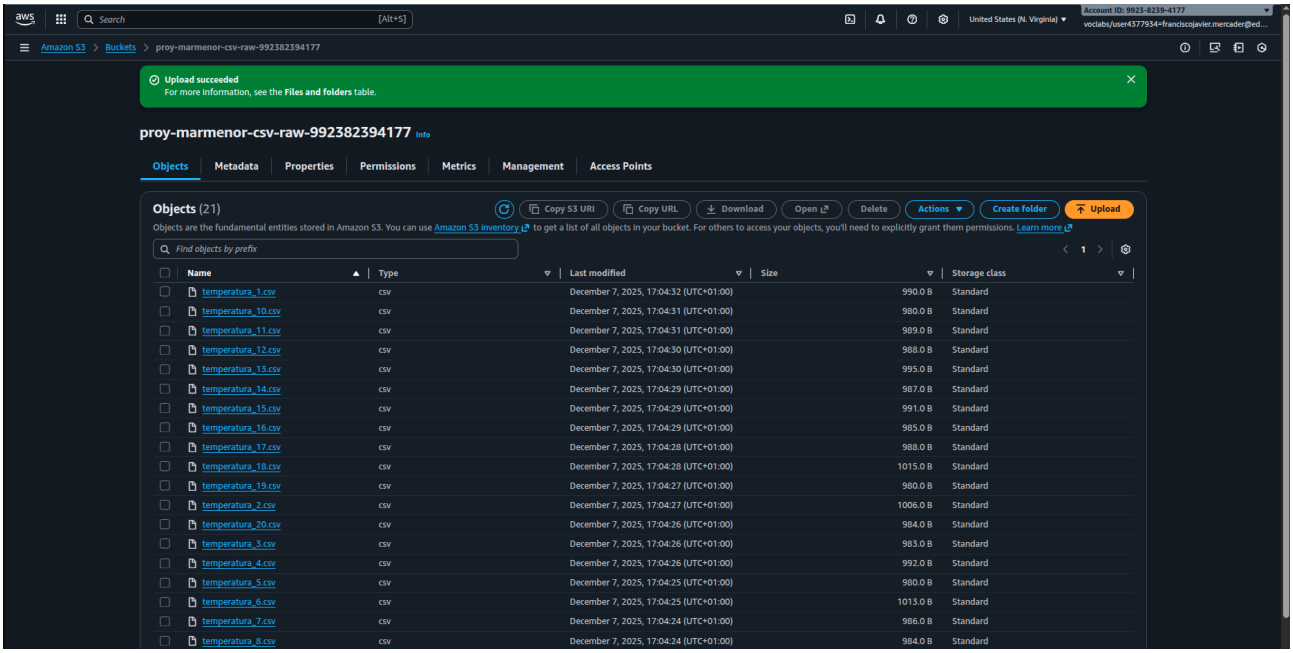
- **Task Security Group (AWS::EC2::SecurityGroup):** Permite el tráfico del ALB a las tareas de ECS (a través del puerto 5000), asegurando que las solicitudes lleguen al contenedor de la aplicación de manera distribuida y segura.

11. Gestión e implementación

- **CloudFormation:** Automatiza el despliegue de la infraestructura mediante plantillas asegurando la coherencia y la reproducibilidad en la implementación.
- **AWS CLI:** Utilizada para el desarrollo de la imagen y autenticación de recursos a la hora de subir la imagen del contenedor de nuestra aplicación a AWS ECR.
- **Docker:** Facilita el desarrollo y la contenedorización de la aplicación, asegurando un entorno de ejecución consistente.
- **IAM Roles (AWS::IAM::Role):** Proporcionan los permisos necesarios para que las tareas de ECS accedan a DynamoDB y otros recursos de AWS. Permite que los distintos recursos de la arquitectura desplegada se puedan comunicar entre sí con seguridad.

4 Ejemplos de demostración de correcta funcionalidad

- **Función Lambda:**



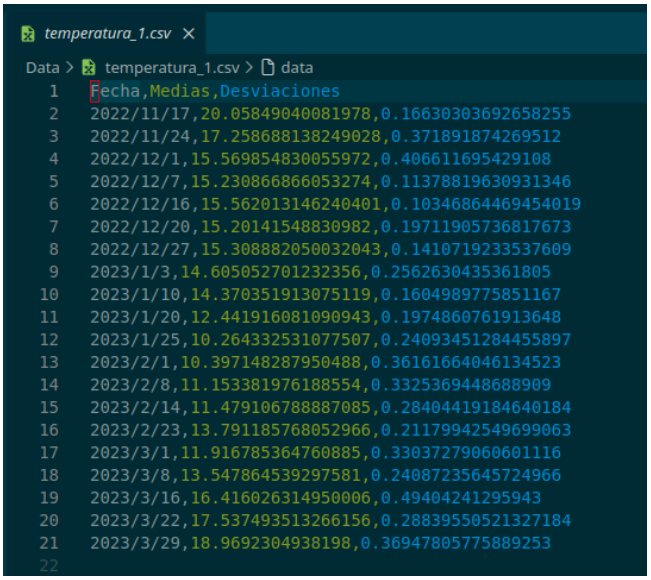


Figure 1: Contenido del fichero: "`temperatura_1.csv`"

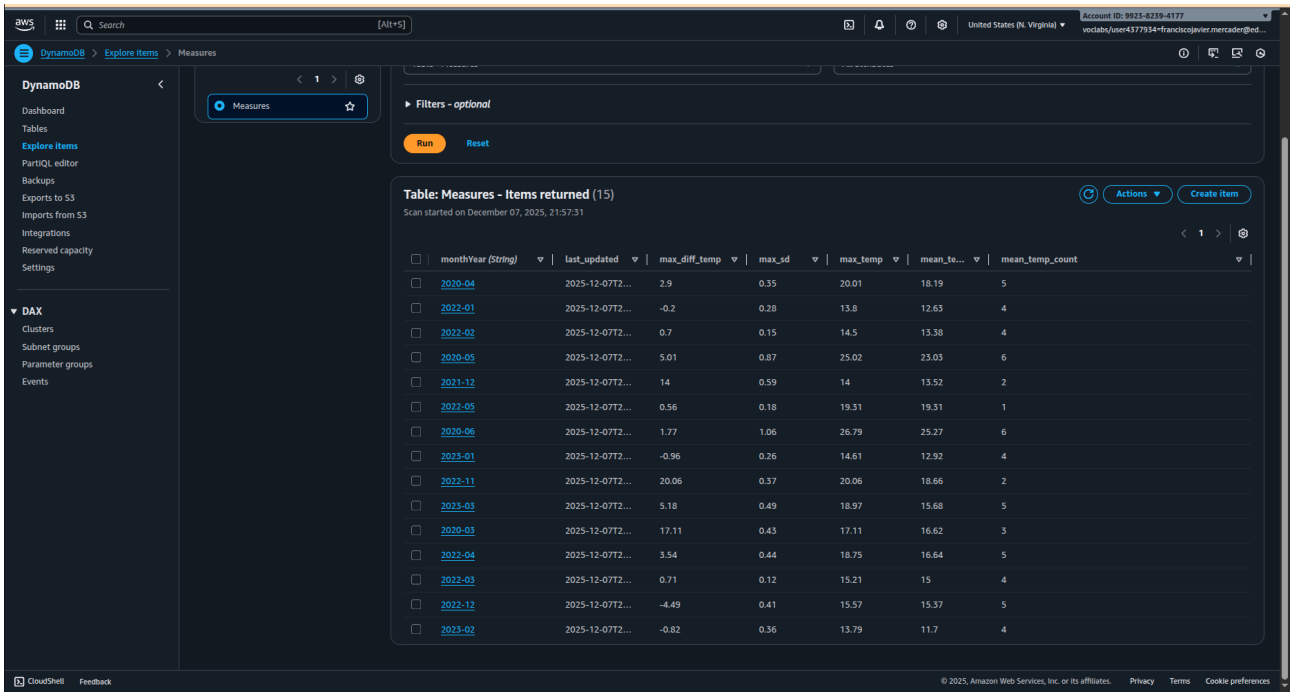


Figure 2: Almacenamiento de datos transformados en DynamoDB

Nota: Observar que la función lambda trata con la posibilidad de ingesta de datos sin importar los siguientes factores: su orden temporal, su repetición y datos mensuales separados en distintos ficheros.

Comentar también que los datos transformados se almacenan en la tabla de DynamoDB correctamente si se realiza la ingesta de manera controlada y con ficheros de un tamaño medio (2-3 Kbytes) como los utilizados en las pruebas realizadas.

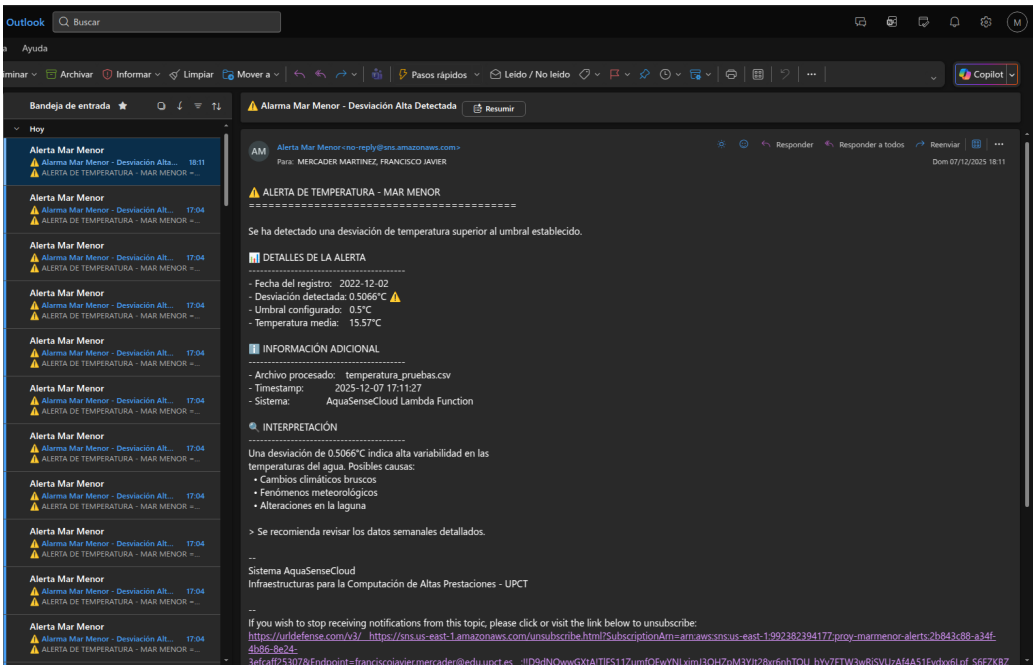
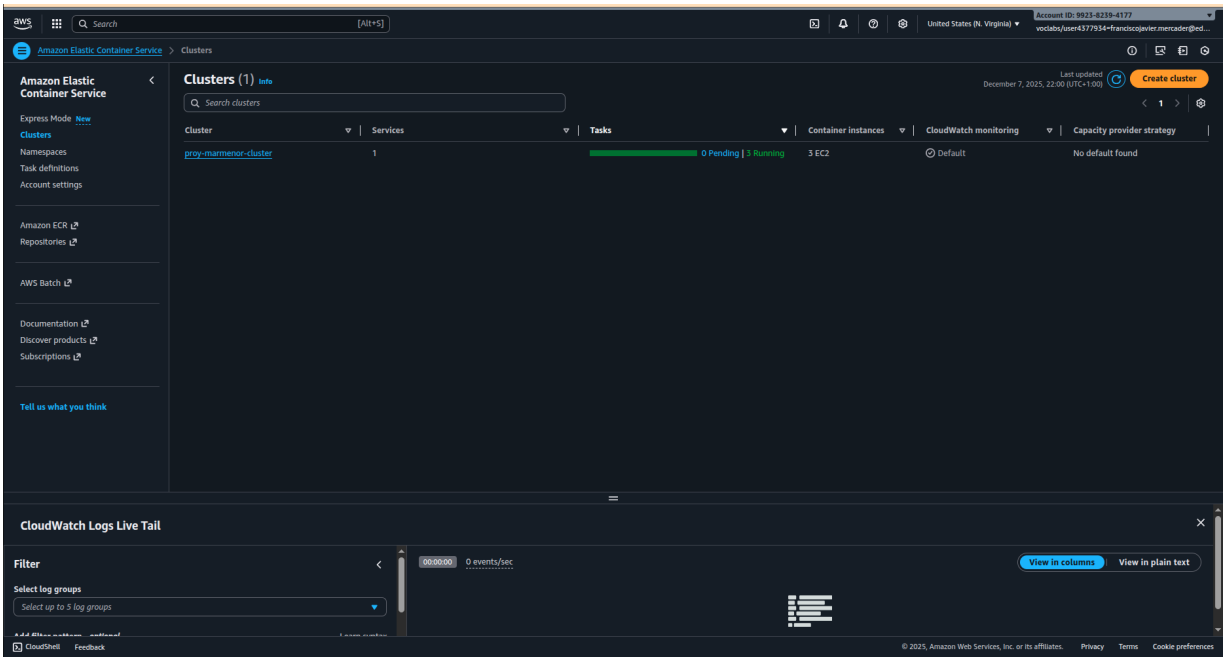
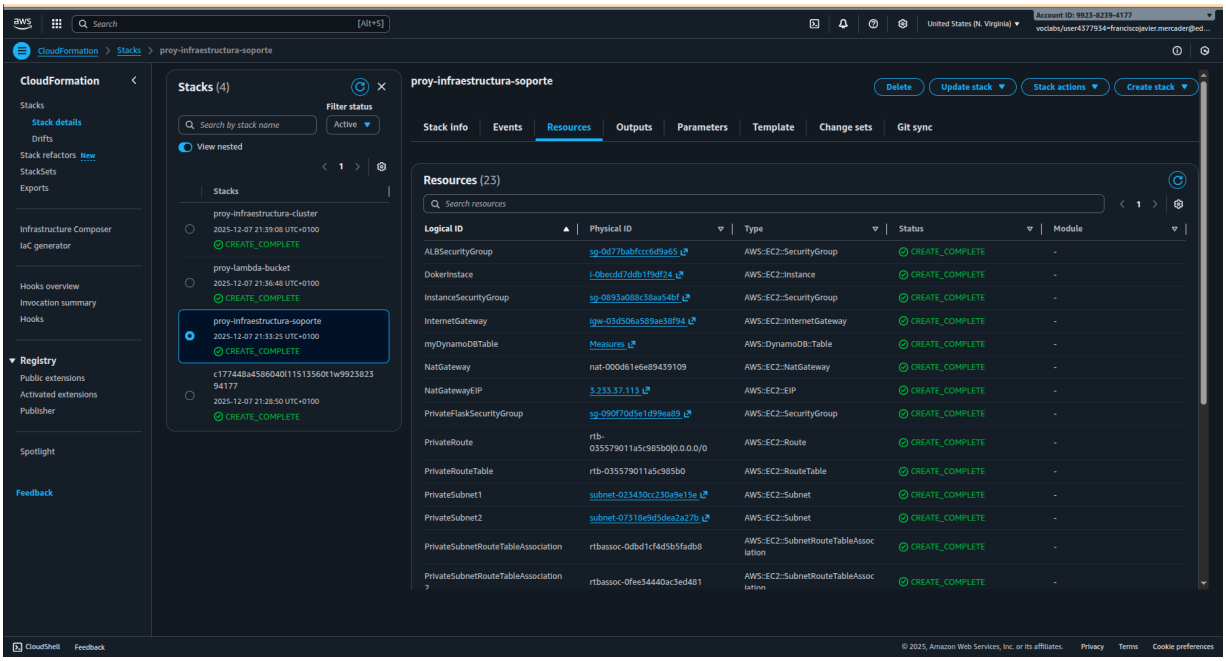


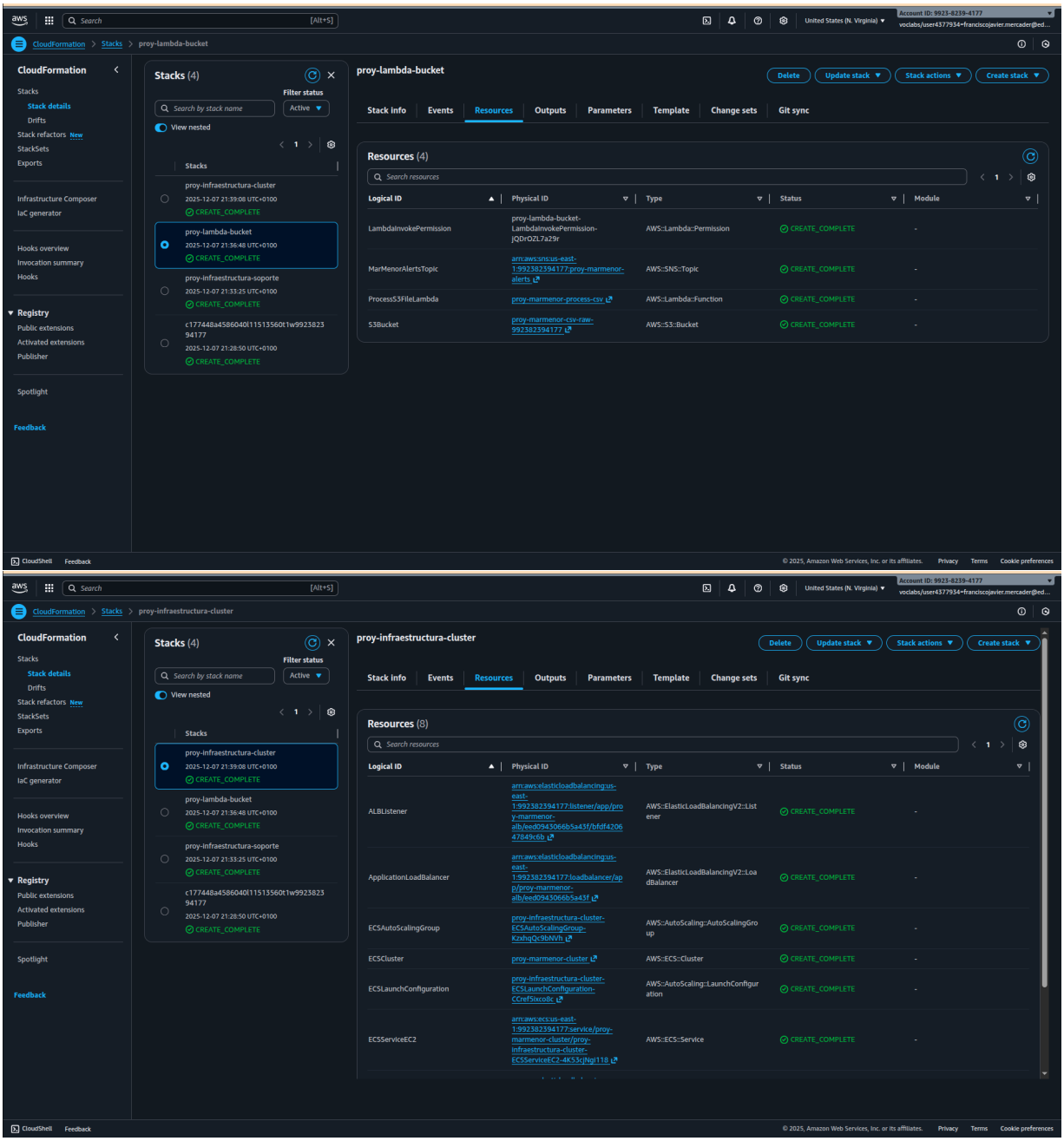
Figure 3: Envío de correo a los analistas mediante un tópico definido mediante AWS SNS

- **ECR:**



- **Automatización del aprovisionamiento de la infraestructura mediante CloudFormation:**

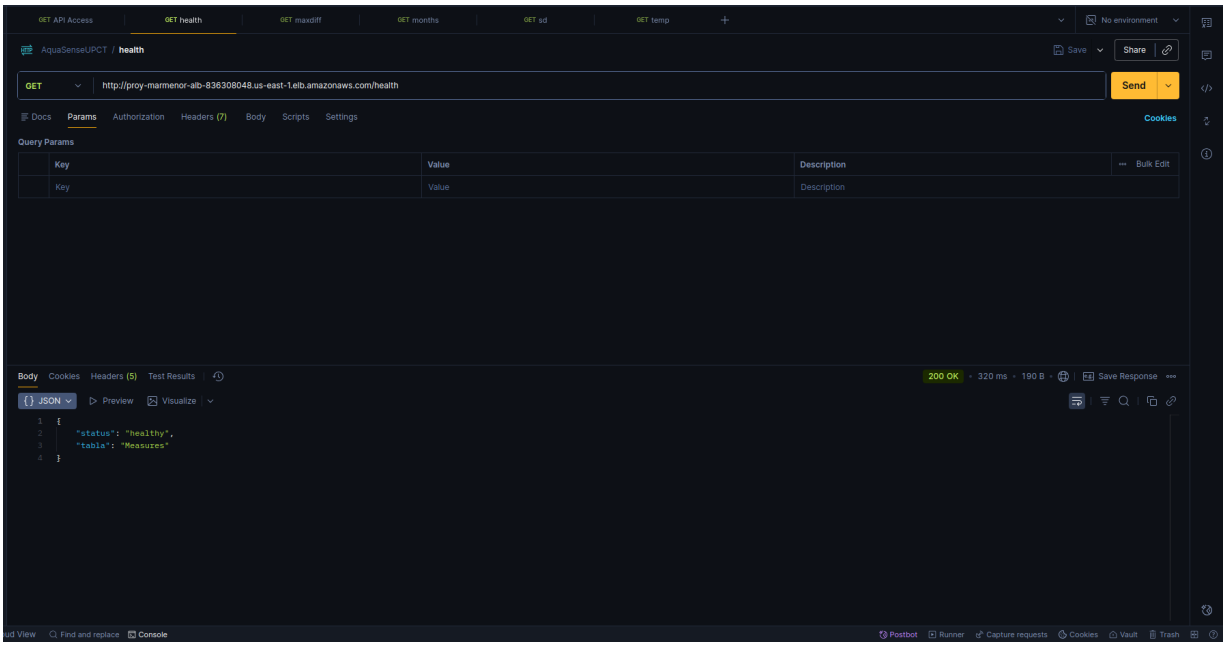




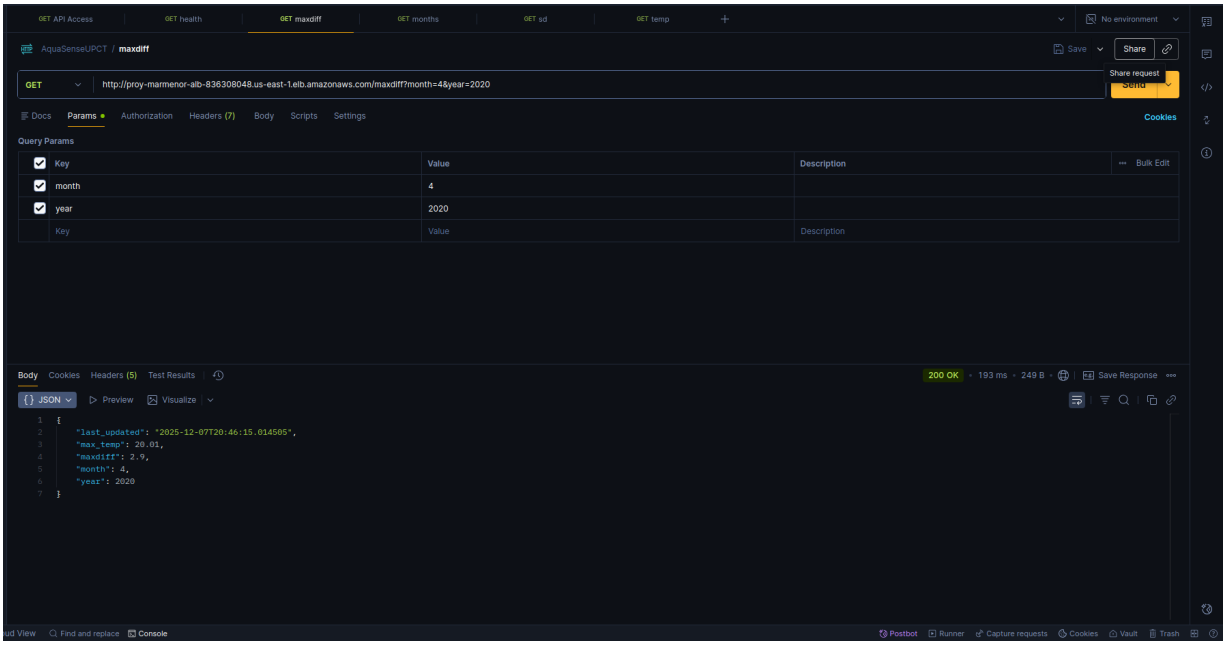
- **Funcionamiento de la aplicación:**

  Distintas consultas que pueden realizar los analistas a través del servidor web desplegado por las tareas. En la propia URL definen el tipo de dato a consultar con su mes y año específico, además del correcto funcionamiento de la aplicación con el endpoint **`/health`**.

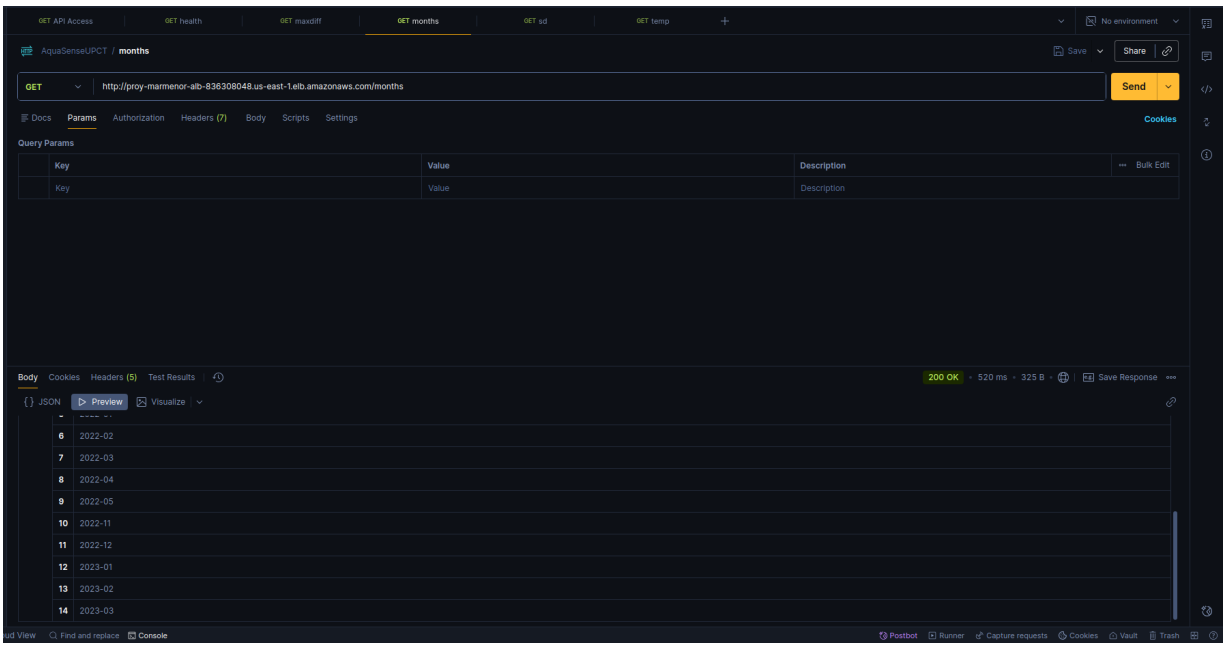
`health/`



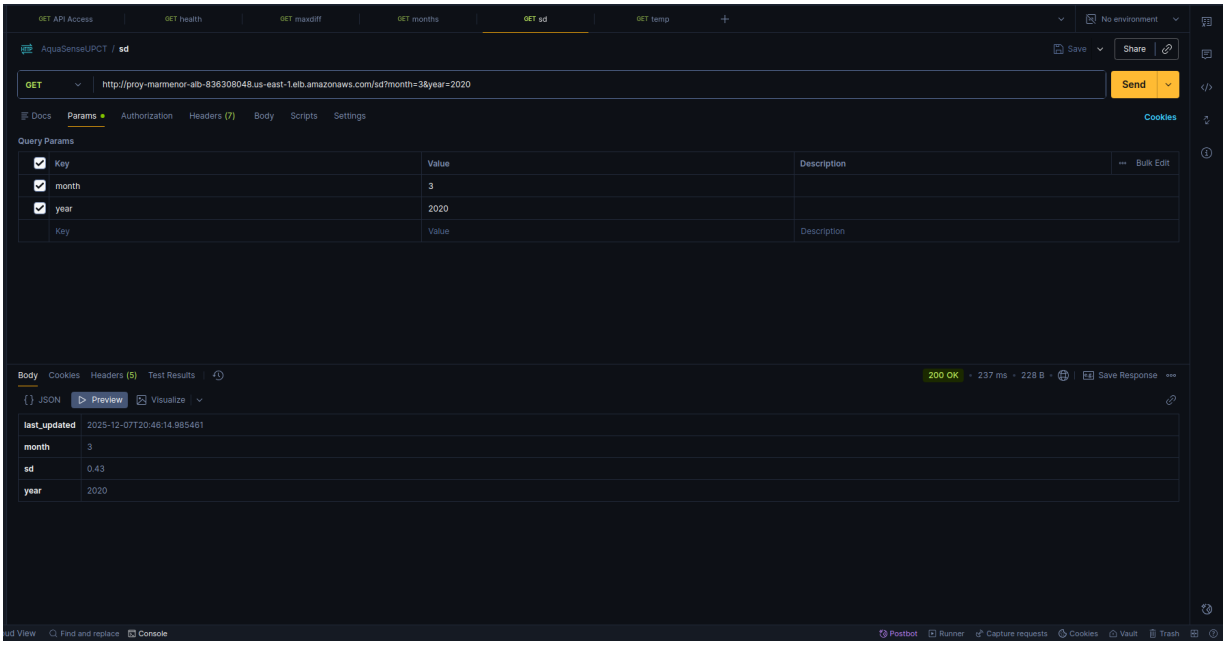
`maxdiff/`



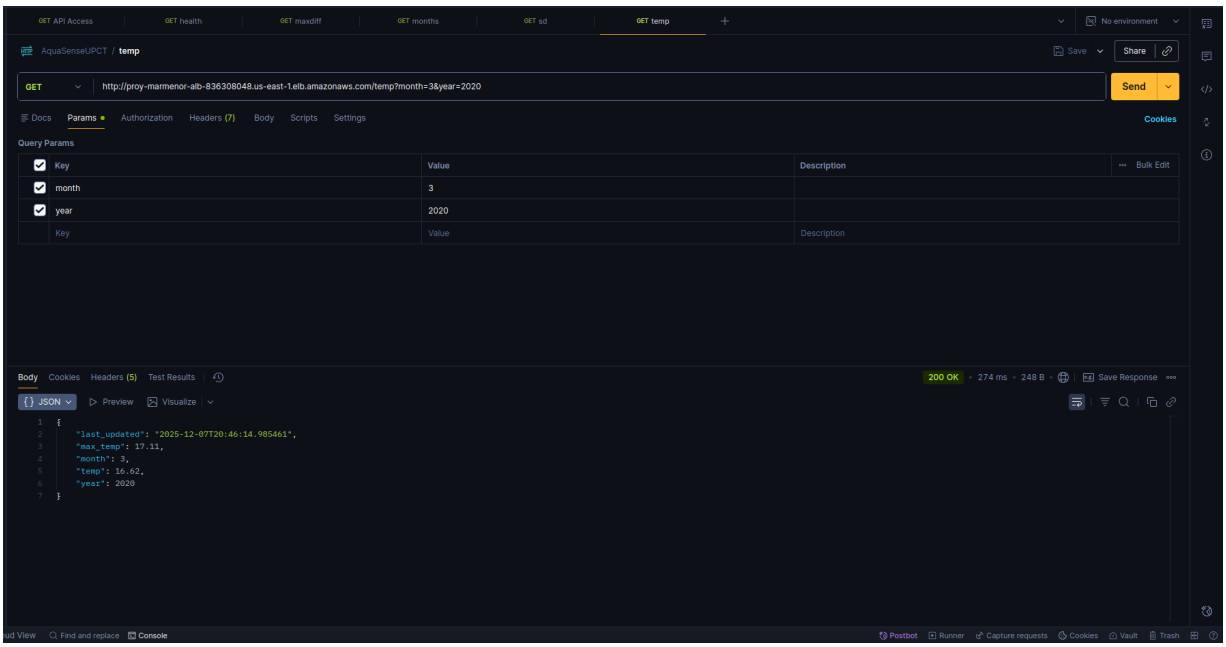
`months/`


GET API Access
GET health
GET maxdiff
GET months
GET sd
GET temp
No environment
AquaSenseUPCT / months
Save
Share
GET
http://proy-marmenor-alb-836308048.us-east-1.elb.amazonaws.com/months
Send
Docs
Params
Authorization
Headers (7)
Body
Scripts
Settings
Cookies
Query Params
Key
Value
Description
Bulk Edit
Body
Cookies
Headers (5)
Test Results
200 OK
520 ms
325 B
Save Response
JSON
Preview
Visualize
6 2022-02
7 2022-03
8 2022-04
9 2022-05
10 2022-11
11 2022-12
12 2023-01
13 2023-02
14 2023-03
Find and replace
Console
Postbot
Runner
Capture requests
Cookies
Vault
Trash

sd/


GET API Access
GET health
GET maxdiff
GET months
GET sd
GET temp
No environment
AquaSenseUPCT / sd
Save
Share
GET
http://proy-marmenor-alb-836308048.us-east-1.elb.amazonaws.com/sd?month=3&year=2020
Send
Docs
Params
Authorization
Headers (7)
Body
Scripts
Settings
Cookies
Query Params
Key
Value
Description
Bulk Edit
month
3
year
2020
Body
Cookies
Headers (5)
Test Results
200 OK
237 ms
228 B
Save Response
JSON
Preview
Visualize
last_updated 2025-12-07T20:46:14.985461
month 3
sd 0.43
year 2020
Find and replace
Console
Postbot
Runner
Capture requests
Cookies
Vault
Trash

`temp/`



5 Pasos a seguir

5.1 Despliegue de la infraestructura

- Paso 0 (Previo): Crear la clave (*Key Pair*) "proy-marmenor"

  Acciones :

  1. Ir a la consola de AWS → EC2 → Network & Security → Key Pairs.
  2. Crear una nueva Key Pair con nombre: proy-marmenor.
  3. Descargar el fichero (formato .pem) y guardarlo de forma segura.

     Nota: debe crearse en la misma región donde se despliega la infraestructura (us-east-1). Si la clave no existe, la pila de CloudFormation que crea la instancia EC2 fallará.

- Paso 1: Infraestructura Base

  1. Acceder a la consola de CloudFormation.
  2. Crear ua nueva pila con **`proy-infraestructura-soporte.yaml`**
  3. Esperar cmpletado (crea: VPC, Subnets, S3, DynamoDB, EC2, SNS)
  4. Verificar que la instancia EC2 haya subido la imagen a ECR.

- Paso 2: Pipeline de Datos Lambda

  1) Crear nueva pila con **`proy-lambda_bucket.yaml`**
  2) Configurar parámetros:
     - **Email**: Correo electrónico para notificaciones SNS, se debe confirmar la suscripción para recibir los mensajes de alerta.
     - **NetworkStackName:** **`proy-infraestructura-soporte`**
  3) Esperar completado (crea: Lambda Bucket S3 raw, Topic SNS)

- Paso 3: Clúster ECS y Aplicación

  1. Crear nueva pila con **`proy-infraestructura-cluster.yaml`**

2. Configurar parámetro:
    - **NetworkStackName:** `proy-infraestructura-soporte`
3. Esperar completado (crea: ECS Cluster, ALB, Target Group, Service)
4. Anota la URL del ALB de los Outputs.

5.2 Carga de Datos y Verificación

5.2.1 Ingesta de Datos

1. Acceder al bucket S3: `proy-marmenor-csv-raw-[ACCOUNT-ID]`
2. Subir archivos CSV de temperatura (formato requerido):

```
Fecha,Medias,Desviaciones
2017/03/22,16.784072875976562,0.28715428709983826
```

3. Verificar ejecución automática de Lambda en CloudWatch Logs.
4. Confirmar datos en DynamoDB (table `Measures`).

5.2.2 Prueba de Endpoints

Desde un navegador o herramienta como Postman/curl:

```
# Health Check
http://[ALB-DNS]/health

# Temperatura media
http://[ALB-DNS]/temp?month=3&year=2017

# Desviación estándar
http://[ALB-DNS]/sd?month=3&year=2017

# Diferencia máxima
http://[ALB-DNS]/maxdiff?month=4&year=2017

# Meses disponibles
http://[ALB-DNS]/months
```

5.3 Eliminación de la Infraestructura

Orden de eliminación (crítico para evitar errores):

1. Vaciar buckets S3:
    - `proy-marmenor-csv-raw-[ACCOUNT-ID]`
    - `proy-marmenor-codebucket-[ACCOUNT-ID]`
2. Eliminar imagen de ECR: `aquasense-api`
3. Eliminar pilas CloudFormation en orden inverso:
    1. `proy-infraestructura-cluster`

2. `proy-lambda_bucket`
3. `proy-infraestructura-soporte`

4. Verificar eliminación de tabla DynamoDB
5. Verificar eliminación de suscripciones SNS

Nota: Esperar 2-3 minutos entre eliminaciones para evitar dependencias.

6 Anexos

6.1 Anexo A - Código Fuente Completo

6.1.1 A.1 Función Lambda de Procesamiento

Archivo: `funcion_lambda.py`

- **Propósito**: Procesa archivos CSV y actualiza DynamoDB
- **Características**:
  - Procesamiento de múltiples archivos simultáneos
  - Detección y sobrescritura de duplicados
  - Ajuste automático de mes (días $\leq 3 \rightarrow$ mes anterior)
  - Envío de alertas SNS (desviación $> 0.5°C$)
  - Cálculo de métricas mensuales agregadas
- **Triggers**: S3 ObjectCreated en `*.csv`
- **Runtime**: Python 3.11
- **Timeout**: 60 segundos
- **Memoria**: 256 MB

```
"""
Función Lambda para AquaSenseCloud
Pipeline de procesamiento de datos del Mar Menor

Descripción:
    Esta función Lambda se ejecuta automáticamente cuando se sube un archivo CSV
    al bucket S3 configurado. Procesa TODOS los archivos CSV del bucket,
    detecta fechas duplicadas y sobrescribe con el último valor encontrado.

Funcionalidades:
    - Lectura de TODOS los archivos CSV del bucket (no solo el trigger)
    - Detección y sobrescritura de fechas duplicadas (última gana)
    - Parsing flexible de múltiples formatos de fecha
    - Cálculo de métricas mensuales (temperatura media, desviación máxima, diferencias)
    - Detección de alarmas (desviación > 0.5ºC)
    - Actualización de DynamoDB con datos procesados
    - Envió de notificaciones SNS
    - Ajuste de mes: si el día es <= 3, se asigna al mes anterior
```

```
Triggers:
    - S3 ObjectCreated:* en bucket proy-marmenor-data-raw-*
    - Filtro: archivos *.csv

Variables de Entorno Requeridas:
    - DYNAMODB_TABLE: Nombre de la tabla DynamoDB
    - SNS_TOPIC_ARN: ARN del topic SNS para alarmas
    - DESVIATION_THRESHOLD: Umbral de desviación (default: 0.5)
"""

import json
import boto3
import csv
import os
import urllib.parse
from datetime import datetime, timedelta
from decimal import Decimal
from collections import defaultdict

# ============================================================================
# CONFIGURACIÓN Y CLIENTES AWS
# ============================================================================

s3_client = boto3.client("s3")
dynamodb = boto3.resource("dynamodb")
sns_client = boto3.client("sns")

# Variables de entorno
DYNAMODB_TABLE = os.environ["DYNAMODB_TABLE"]
SNS_TOPIC_ARN = os.environ["SNS_TOPIC_ARN"]
DEVIATION_THRESHOLD = Decimal(str(os.environ.get("DEVIATION_THRESHOLD", "0.5")))

# Tabla DynamoDB
table = dynamodb.Table(DYNAMODB_TABLE)


# ============================================================================
# FUNCIONES AUXILIARES
# ============================================================================

def round_decimal(value):
    """Redondea un valor Decimal a 4 decimales."""
    return value.quantize(Decimal("0.0001"))

def adjust_month_for_date(fecha_dt):
    """
    Ajusta el mes de una fecha según el día.
    Si el día es <= 3, se asigna al mes anterior.

    Args:
        fecha_dt: datetime object
```

```
    Returns:
        str: Mes en formato 'YYYY-MM' ajustado
    """
    if fecha_dt.day <= 3:
        # Mover al mes anterior
        adjusted_date = fecha_dt.replace(day=1) - timedelta(days=1)
        return adjusted_date.strftime('%Y-%m')
    else:
        return fecha_dt.strftime('%Y-%m')

def send_alert(fecha_str, desviacion, temp_media, filename):
    """Envía una alerta por SNS."""
    try:
        fecha_dt = datetime.strptime(fecha_str, '%Y/%m/%d')
        fecha_formatted = fecha_dt.strftime("%Y-%m-%d")

        subject = "  Alarma Mar Menor - Desviación Alta Detectada"
        message = f"""
 ALERTA DE TEMPERATURA - MAR MENOR
==========================================
Se ha detectado una desviación superior al umbral.

 DETALLES
------------------------------------------
- Fecha:         {fecha_formatted}
- Desviación:    {float(desviacion):.4f}°C
- Umbral:        {float(DEVIATION_THRESHOLD):.2f}°C
- Temp Media:    {float(temp_media):.2f}°C
- Archivo:       {filename}

--
Sistema AquaSenseCloud
        """.strip()

        sns_client.publish(
            TopicArn=SNS_TOPIC_ARN,
            Subject=subject,
            Message=message
        )
    except Exception as e:
        print(f"Error enviando alerta SNS: {str(e)}")


def get_all_csv_files(bucket):
    """Obtiene lista de todos los archivos CSV del bucket ordenados alfabéticamente."""
    try:
        response = s3_client.list_objects_v2(Bucket=bucket)

        if 'Contents' not in response:
            return []

        # Filtrar solo CSVs y ordenar alfabéticamente
```

```
        csv_files = sorted([
            obj['Key'] for obj in response['Contents']
            if obj['Key'].lower().endswith('.csv')
        ])

        return csv_files

    except Exception as e:
        print(f"Error listing bucket contents: {e}")
        return []


def process_csv_file(bucket, key, local_dir):
    """
    Descarga y procesa un archivo CSV.
    Retorna diccionario de fechas: {'2023-01-15': {'temp': 20, 'sd': 0.5, 'source': 'file
        .csv', 'adjusted_month': '2023-01'}}
    """
    local_filename = os.path.join(local_dir, os.path.basename(key))
    daily_data = {}

    try:
        s3_client.download_file(bucket, key, local_filename)

        with open(local_filename, encoding='utf-8') as csvfile:
            reader = csv.DictReader(csvfile, delimiter=',')

            for row in reader:
                # Parseo de fecha
                try:
                    fecha = datetime.strptime(row['Fecha'], '%Y/%m/%d')
                except ValueError:
                    try:
                        fecha = datetime.strptime(row['Fecha'], '%Y-%m-%d')
                    except ValueError:
                        print(f"Warning: Invalid date format in {key}: {row['Fecha']}")
                        continue

                fecha_str = fecha.strftime('%Y-%m-%d')
                temp_media = round_decimal(Decimal(row['Medias']))
                desviacion = round_decimal(Decimal(row['Desviaciones']))

                # Ajustar mes según el día (si día <= 3, va al mes anterior)
                adjusted_month = adjust_month_for_date(fecha)

                # Guardar (sobrescribe si ya existe en este archivo)
                daily_data[fecha_str] = {
                    'temp': temp_media,
                    'sd': desviacion,
                    'source': key,
                    'adjusted_month': adjusted_month  # Mes ajustado
                }
```

```
        return daily_data

    except Exception as e:
        print(f"Error processing file {key}: {e}")
        return {}

    finally:
        if os.path.exists(local_filename):
            os.remove(local_filename)


# ================================================================================
# HANDLER PRINCIPAL
# ================================================================================

def lambda_handler(event, context):

    bucket = event['Records'][0]['s3']['bucket']['name']
    trigger_key = urllib.parse.unquote_plus(event['Records'][0]['s3']['object']['key'])

    local_dir = "/tmp"

    try:
        # ============================================================
        # PASO 1: Obtener TODOS los archivos CSV del bucket
        # ============================================================
        all_csv_files = get_all_csv_files(bucket)

        if not all_csv_files:
            return {
                "statusCode": 200,
                "body": json.dumps({
                    "message": "No CSV files found in bucket",
                    "trigger_file": trigger_key
                })
            }

        # ============================================================
        # PASO 2: Procesar todos los archivos y fusionar datos
        # ============================================================
        merged_daily_data = {}  # {'2023-01-15': {'temp': 20, 'sd': 0.5, 'source': 'file.
            csv', 'adjusted_month': '2023-01'}}
        alerts_sent = 0
        total_rows = 0
        files_processed = 0
        month_adjustments = 0  # Contador de fechas ajustadas

        for csv_key in all_csv_files:
            file_data = process_csv_file(bucket, csv_key, local_dir)

            if file_data:
```

```
            files_processed += 1
            total_rows += len(file_data)

            # Fusionar: última aparición sobrescribe
            for fecha, data in file_data.items():

                # Verificar si se ajustó el mes
                fecha_dt = datetime.strptime(fecha, '%Y-%m-%d')
                original_month = fecha_dt.strftime('%Y-%m')
                if data['adjusted_month'] != original_month:
                    month_adjustments += 1

                merged_daily_data[fecha] = data

                # Detectar alertas
                if data['sd'] > DEVIATION_THRESHOLD:
                    send_alert(fecha.replace('-', '/'), data['sd'], data['temp'],
                        csv_key)
                    alerts_sent += 1

    duplicates_found = total_rows - len(merged_daily_data)

    # ============================================================
    # PASO 3: Agrupar por mes AJUSTADO y calcular métricas
    # ============================================================
    monthly_data = defaultdict(dict)  # {'2023-01': {'2023-01-05': {...}, ...}}

    for fecha_str, data in merged_daily_data.items():
        # Usar el mes ajustado en lugar del mes natural
        mes = data['adjusted_month']
        monthly_data[mes][fecha_str] = data

    # ============================================================
    # PASO 4: Actualizar DynamoDB
    # ============================================================
    updated_months = 0

    for mes, dates_dict in sorted(monthly_data.items()):
        try:
            # Calcular métricas del mes
            all_temps = [d['temp'] for d in dates_dict.values()]
            all_sds = [d['sd'] for d in dates_dict.values()]

            max_temp = max(all_temps)
            max_sd = max(all_sds)
            mean_temp = round_decimal(sum(all_temps) / len(all_temps))
            count = len(all_temps)

            # Calcular diferencia con mes anterior
            current_month_dt = datetime.strptime(mes, '%Y-%m')
            previous_month_dt = current_month_dt.replace(day=1) - timedelta(days=1)
            previous_month = previous_month_dt.strftime('%Y-%m')
```

```
            # Buscar mes anterior en datos procesados o DB
            if previous_month in monthly_data:
                prev_temps = [d['temp'] for d in monthly_data[previous_month].values
                    ()]
                prev_max = max(prev_temps)
            else:
                # Buscar en DynamoDB
                previous_item_resp = table.get_item(Key={'monthYear': previous_month
                    })
                previous_item = previous_item_resp.get('Item', {})
                prev_max = Decimal(str(previous_item.get('max_temp', 0)))

            max_diff_temp = round_decimal(max_temp - prev_max)

            # Guardar en DynamoDB
            table.put_item(
                Item={
                    'monthYear': mes,
                    'max_temp': max_temp,
                    'max_sd': max_sd,
                    'mean_temp': mean_temp,
                    'max_diff_temp': max_diff_temp,
                    'mean_temp_count': count,
                    'last_updated': datetime.now().isoformat()
                }
            )

            updated_months += 1

        except Exception as e:
            print(f"Error updating month {mes}: {e}")
            raise

    return {
        "statusCode": 200,
        "body": json.dumps({
            "message": "Processing completed successfully",
            "trigger_file": trigger_key,
            "files_processed": files_processed,
            "total_rows": total_rows,
            "unique_dates": len(merged_daily_data),
            "duplicates_overwritten": duplicates_found,
            "month_adjustments": month_adjustments,
            "months_updated": updated_months,
            "alerts_sent": alerts_sent
        })
    }

except Exception as e:
    import traceback
    traceback.print_exc()
```

```
        raise Exception(f"Error processing bucket {bucket}: {str(e)}")
```

6.1.2 A.2 API REST - Servidor Web

Archivo: `aquasense.py`

- **Framework**: Flask 3.0.0
- **Endpoints implementados**: 6
- **Base de datos**: DynamoDB (tabla plana)
- **Puerto**: 8080
- **Health check**: `/health` (para ALB)

```
"""
AquaSense API REST - Servidor Web para consultar datos del Mar Menor

Autor: Equipo AquaSenseCloud

Descripción:
    API REST que proporciona acceso a las estadísticas de temperatura
    del Mar Menor almacenadas en DynamoDB con estructura de tabla plana.

    La tabla DynamoDB utiliza una estructura simplificada donde cada registro
    mensual contiene todos los datos agregados (temperatura media, máxima
    desviación, diferencia con mes anterior) en una única fila.

Endpoints:
    GET /            - Información general de la API
    GET /health      - Health check del servidor
    GET /maxdiff     - Diferencia de temperatura máxima mensual vs mes anterior
    GET /sd          - Máxima desviación estándar mensual
    GET /temp        - Temperatura media mensual
    GET /months      - Lista de todos los meses con datos disponibles

Estructura de Tabla DynamoDB (Flat Table):
    Partition Key: monthYear (String, formato "YYYY-MM")
    Atributos:
        - max_temp: Temperatura máxima del mes (Decimal)
        - max_sd: Máxima desviación estándar del mes (Decimal)
        - mean_temp: Temperatura media del mes (Decimal)
        - max_diff_temp: Diferencia con temperatura máxima mes anterior (Decimal)
        - mean_temp_count: Número de registros procesados (Number)
        - last_updated: Timestamp de última actualización (String ISO)

Requisitos:
    - Flask 3.0.0
    - boto3 1.34.0
    - Python 3.11+

Variables de Entorno:
    - PORT: Puerto del servidor (default: 8080)
```

```
    - DYNAMODB_TABLE: Nombre de la tabla DynamoDB (default: "proy-MarMenorData")
    - AWS_REGION: Región AWS (default: "us-east-1")
"""

from flask import Flask, request, jsonify
import boto3
import os
from decimal import Decimal
import logging

# ======================================
# CONFIGURACIÓN
# ======================================

# Configuración de logging para trazabilidad
logging.basicConfig(
    level=logging.INFO, format="%(asctime)s - %(name)s - %(levelname)s - %(message)s"
)
logger = logging.getLogger(__name__)

# Inicialización de Flask
app = Flask(__name__)
app.config["JSON_SORT_KEYS"] = False  # Mantener orden de claves en respuestas JSON

# Cliente DynamoDB
dynamodb = boto3.resource(
    "dynamodb", region_name=os.environ.get("AWS_REGION", "us-east-1")
)
table_name = os.environ.get("DYNAMODB_TABLE", "proy-MarMenorData")
table = dynamodb.Table(table_name)

logger.info(f"AquaSenseCloud API iniciada")
logger.info(f"DynamoDB Table: {table_name}")

# ======================================
# FUNCIONES AUXILIARES
# ======================================

def decimal_to_float(obj):
    """
    Convierte objetos Decimal de DynamoDB a float para serialización JSON.

    DynamoDB devuelve números como objetos Decimal, que no son serializables
    directamente a JSON. Esta función convierte recursivamente todos los
    Decimal a float en diccionarios, listas y valores individuales.

    Args:
        obj: Objeto que puede contener Decimals (dict, list, Decimal, o cualquier tipo)

    Returns:
        Objeto con todos los Decimals convertidos a float
```

```
    Ejemplo:
        >>> decimal_to_float({"temp": Decimal("17.5"), "count": Decimal("3")})
        {"temp": 17.5, "count": 3.0}
    """
    if isinstance(obj, Decimal):
        return float(obj)
    elif isinstance(obj, dict):
        return {k: decimal_to_float(v) for k, v in obj.items()}
    elif isinstance(obj, list):
        return [decimal_to_float(item) for item in obj]
    return obj

def validate_parameters(request):
    """
    Valida los parámetros month y year de la petición HTTP.

    Verifica que:
    - Los parámetros existan
    - Sean números enteros válidos
    - El mes esté en el rango [1-12]
    - El año esté en el rango [2000-2100]

    Args:
        request: Objeto request de Flask con los query parameters

    Returns:
        tuple: (month, year, error_response)
            - Si es válido: (int, int, None)
            - Si hay error: (None, None, (json_response, status_code))

    Ejemplo:
        month, year, error = validate_parameters(request)
        if error:
            return error
        # Continuar con month y year validados
    """
    try:
        month_str = request.args.get("month")
        year_str = request.args.get("year")

        # Verificar que ambos parámetros existan
        if not month_str or not year_str:
            return (
                None,
                None,
                (jsonify({
                    "error": "Parámetros faltantes",
                    "message": 'Los parámetros "month" y "year" son obligatorios',
                    "ejemplo": "/temp?month=3&year=2017"
                }), 400)
            )
```

```
        # Convertir a enteros
        month = int(month_str)
        year = int(year_str)

        # Validar rangos
        if not (1 <= month <= 12) or not (2000 <= year <= 2100):
            return (
                None,
                None,
                (jsonify({
                    "error": "Parámetros fuera de rango",
                    "message": "El mes debe estar entre 1-12 y el año entre 2000-2100"
                }), 400)
            )

        return month, year, None

    except ValueError:
        return (
            None,
            None,
            (jsonify({
                "error": "Formato inválido",
                "message": "Los parámetros deben ser números enteros"
            }), 400)
        )

# ======================================
# ENDPOINTS
# ======================================

@app.route("/", methods=["GET"])
def index():
    """
    Endpoint raíz - Información general de la API.

    Proporciona información sobre los endpoints disponibles y cómo usarlos.
    Útil para descubrimiento de API y documentación inicial.

    Returns:
        JSON con información de la API y lista de endpoints (200 OK)

    Ejemplo:
        GET /

        Response:
        {
            "servicio": "AquaSenseCloud API",
            "version": "2.0 (Flat Table)",
            "endpoints": {...}
        }
    """
```

```
    return jsonify({
        "servicio": "AquaSenseCloud API",
        "version": "2.0 (Flat Table)",
        "descripcion": "API REST para consultar datos de temperatura del Mar Menor",
        "endpoints": {
            "/health": "Health check del servidor",
            "/maxdiff": "Diferencia temperatura máxima mensual vs mes anterior",
            "/sd": "Máxima desviación estándar mensual",
            "/temp": "Temperatura media mensual",
            "/months": "Lista de meses con datos disponibles"
        },
        "uso": {
            "maxdiff": "GET /maxdiff?month=3&year=2017",
            "sd": "GET /sd?month=3&year=2017",
            "temp": "GET /temp?month=3&year=2017",
            "months": "GET /months"
        },
        "proyecto": "Infraestructura para la Computación de Altas Prestaciones - UPCT"
    }), 200

@app.route("/health", methods=["GET"])
def health_check():
    """
    Health check endpoint para monitoreo y load balancer.

    Verifica que:
    - El servicio Flask esté funcionando
    - La conexión con DynamoDB esté operativa

    Este endpoint es usado por el Application Load Balancer para determinar
    si la instancia debe recibir tráfico.

    Returns:
        JSON con estado "healthy" (200 OK) si todo funciona correctamente
        JSON con estado "unhealthy" (503 Service Unavailable) si hay errores

    Ejemplo:
        GET /health

        Response (exitoso):
        {
            "status": "healthy",
            "tabla": "proy-MarMenorData"
        }
    """
    try:
        # Verificar conectividad con DynamoDB
        table.table_status

        return jsonify({
            "status": "healthy",
            "tabla": table_name
```

```
        }), 200

    except Exception as e:
        logger.error(f"Health check failed: {str(e)}")
        return jsonify({
            "status": "unhealthy",
            "error": str(e)
        }), 503

@app.route("/maxdiff", methods=["GET"])
def get_maxdiff():
    """
    Endpoint: /maxdiff?month=M&year=Y

    Retorna la diferencia de temperatura máxima del mes especificado
    respecto a la temperatura máxima del mes anterior.

    Requisito del proyecto:
        "Diferencia de la máxima temperatura del mes respecto a la máxima
        temperatura del mes anterior"

    Tabla DynamoDB (estructura plana):
        Key: {"monthYear": "YYYY-MM"}
        Atributo: max_diff_temp (Decimal)

    Query Parameters:
        month (int): Mes a consultar (1-12)
        year (int): Año a consultar (2000-2100)

    Returns:
        JSON con la diferencia de temperatura (200 OK)
        JSON con error (404 Not Found si no hay datos)
        JSON con error (400 Bad Request si parámetros inválidos)
        JSON con error (500 Internal Server Error si falla la consulta)

    Ejemplo:
        GET /maxdiff?month=4&year=2017

        Response:
        {
            "month": 4,
            "year": 2017,
            "maxdiff": 2.14,
            "max_temp": 19.47,
            "last_updated": "2024-12-07T10:30:00Z"
        }
    """
    try:
        # Validar parámetros de entrada
        month, year, error = validate_parameters(request)
        if error:
            return error
```

```
        # Construir clave de consulta en formato YYYY-MM
        month_year = f"{year}-{month:02d}"

        logger.info(f"Consultando maxdiff para: {month_year}")

        # Consultar DynamoDB (tabla plana - solo Partition Key)
        response = table.get_item(Key={"monthYear": month_year})

        # Verificar si existe el registro
        if "Item" not in response:
            logger.warning(f"No hay datos para {month_year}")
            return jsonify({
                "error": "Datos no encontrados",
                "message": f"No hay datos disponibles para {month}/{year}"
            }), 404

        # Convertir Decimals a float para JSON
        item = decimal_to_float(response["Item"])

        # Preparar respuesta con los datos del atributo max_diff_temp
        result = {
            "month": month,
            "year": year,
            "maxdiff": item.get("max_diff_temp"),  # Diferencia con mes anterior
            "max_temp": item.get("max_temp"),       # Temperatura máxima del mes
            "last_updated": item.get("last_updated")
        }

        logger.info(f"Maxdiff {month_year}: {result['maxdiff']}°C")

        return jsonify(result), 200

    except Exception as e:
        logger.error(f"Error en /maxdiff: {str(e)}")
        return jsonify({
            "error": "Internal Server Error",
            "message": str(e)
        }), 500

@app.route("/sd", methods=["GET"])
def get_sd():
    """
    Endpoint: /sd?month=M&year=Y

    Retorna la máxima desviación estándar de temperatura del mes especificado.

    Requisito del proyecto:
        "Máxima desviación del conjunto de datos obtenidos durante el mes"

    La desviación estándar indica la variabilidad de las temperaturas durante
    el mes. Una desviación alta sugiere cambios bruscos de temperatura.
```

```
    Tabla DynamoDB (estructura plana):
        Key: {"monthYear": "YYYY-MM"}
        Atributo: max_sd (Decimal)

    Query Parameters:
        month (int): Mes a consultar (1-12)
        year (int): Año a consultar (2000-2100)

    Returns:
        JSON con la desviación estándar (200 OK)
        JSON con error (404 Not Found si no hay datos)
        JSON con error (400 Bad Request si parámetros inválidos)
        JSON con error (500 Internal Server Error si falla la consulta)

    Ejemplo:
        GET /sd?month=3&year=2017

        Response:
        {
            "month": 3,
            "year": 2017,
            "sd": 0.6254,
            "last_updated": "2024-12-07T10:30:00Z"
        }
    """
    try:
        # Validar parámetros de entrada
        month, year, error = validate_parameters(request)
        if error:
            return error

        # Construir clave de consulta
        month_year = f"{year}-{month:02d}"

        logger.info(f"Consultando sd para: {month_year}")

        # Consultar DynamoDB (tabla plana)
        response = table.get_item(Key={"monthYear": month_year})

        # Verificar si existe el registro
        if "Item" not in response:
            logger.warning(f"No hay datos para {month_year}")
            return jsonify({
                "error": "Datos no encontrados",
                "message": f"No hay datos disponibles para {month}/{year}"
            }), 404

        # Convertir Decimals a float
        item = decimal_to_float(response["Item"])

        # Preparar respuesta con el atributo max_sd
```

```
        result = {
            "month": month,
            "year": year,
            "sd": item.get("max_sd"),  # Máxima desviación estándar del mes
            "last_updated": item.get("last_updated")
        }

        logger.info(f"SD {month_year}: {result['sd']}")

        return jsonify(result), 200

    except Exception as e:
        logger.error(f"Error en /sd: {str(e)}")
        return jsonify({
            "error": "Internal Server Error",
            "message": str(e)
        }), 500

@app.route("/temp", methods=["GET"])
def get_temp():
    """
    Endpoint: /temp?month=M&year=Y

    Retorna la temperatura media del mes especificado.

    Requisito del proyecto:
        "Temperatura media de los datos del mes"

    La temperatura media se calcula como el promedio ponderado de todas
    las mediciones semanales del mes, combinando datos de múltiples archivos
    CSV si es necesario.

    Tabla DynamoDB (estructura plana):
        Key: {"monthYear": "YYYY-MM"}
        Atributo: mean_temp (Decimal) - temperatura media calculada

    Query Parameters:
        month (int): Mes a consultar (1-12)
        year (int): Año a consultar (2000-2100)

    Returns:
        JSON con la temperatura media (200 OK)
        JSON con error (404 Not Found si no hay datos)
        JSON con error (400 Bad Request si parámetros inválidos)
        JSON con error (500 Internal Server Error si falla la consulta)

    Ejemplo:
        GET /temp?month=3&year=2017

        Response:
        {
            "month": 3,
```

```
            "year": 2017,
            "temp": 17.06,
            "max_temp": 17.33,
            "last_updated": "2024-12-07T10:30:00Z"
        }
    """
    try:
        # Validar parámetros de entrada
        month, year, error = validate_parameters(request)
        if error:
            return error

        # Construir clave de consulta
        month_year = f"{year}-{month:02d}"

        logger.info(f"Consultando temp para: {month_year}")

        # Consultar DynamoDB (tabla plana)
        response = table.get_item(Key={"monthYear": month_year})

        # Verificar si existe el registro
        if "Item" not in response:
            logger.warning(f"No hay datos para {month_year}")
            return jsonify({
                "error": "Datos no encontrados",
                "message": f"No hay datos disponibles para {month}/{year}"
            }), 404

        # Convertir Decimals a float
        item = decimal_to_float(response["Item"])

        # Preparar respuesta con el atributo mean_temp
        result = {
            "month": month,
            "year": year,
            "temp": item.get("mean_temp"),    # Temperatura media del mes
            "max_temp": item.get("max_temp"), # Temperatura máxima del mes
            "last_updated": item.get("last_updated")
        }

        logger.info(f"Temp {month_year}: {result['temp']}°C")

        return jsonify(result), 200

    except Exception as e:
        logger.error(f"Error en /temp: {str(e)}")
        return jsonify({
            "error": "Internal Server Error",
            "message": str(e)
        }), 500

@app.route("/months", methods=["GET"])
```

```python
def get_available_months():
    """
    Endpoint: /months

    Lista todos los meses disponibles en la base de datos.

    Útil para que los analistas conozcan qué períodos tienen datos disponibles
    antes de hacer consultas específicas. Con la estructura de tabla plana,
    cada registro en DynamoDB representa un mes completo con todas sus métricas.

    Tabla DynamoDB (estructura plana):
        Cada fila tiene monthYear como Partition Key único
        No hay Sort Key, por lo que cada mes es una fila independiente

    Returns:
        JSON con lista de meses (200 OK)
        JSON con error (500 Internal Server Error si falla el escaneo)

    Nota:
        Usa table.scan() que es costoso en tablas grandes, pero es aceptable
        para este dataset (máximo ~100 meses). Para datasets mayores, considerar
        usar un índice GSI o una tabla auxiliar de índice.

    Ejemplo:
        GET /months

        Response:
        {
            "months": ["2017-03", "2017-04", "2017-05", ...],
            "count": 45
        }
    """
    try:
        logger.info("Listando meses disponibles")

        # Escanear tabla completa (solo proyectando monthYear para eficiencia)
        response = table.scan(ProjectionExpression="monthYear")
        items = response.get("Items", [])

        # Extraer lista de meses únicos y ordenar
        # Como monthYear es la Partition Key, cada valor es único automáticamente
        months_list = sorted([item["monthYear"] for item in items])

        logger.info(f"Total meses disponibles: {len(months_list)}")

        return jsonify({
            "months": months_list,
            "count": len(months_list)
        }), 200

    except Exception as e:
        logger.error(f"Error en /months: {str(e)}")
```

```
        return jsonify({
            "error": "Internal Server Error",
            "message": str(e)
        }), 500

# ======================================
# PUNTO DE ENTRADA
# ======================================

if __name__ == "__main__":
    port = int(os.environ.get("PORT", 8080))
    app.run(host="0.0.0.0", port=port)
```

6.2 Anexo B - Plantillas de Infraestructura como Código

6.2.1 B.1 Infraestructura Base

Archivo: `proy-infraestructura-soporte.yaml`

Recursos creados:

- VPC (192.168.0.0/16)
- 2 Subnets públicas + 2 privadas
- Internet Gateway + NAT Gateway
- Security Groups (ALB, Instancias)
- Tabla DynamoDB: `Measures`
- Bucket S3: `proy-marmenor-codebucket`
- Instancia EC2 para construcción Docker

```
AWSTemplateFormatVersion: 2010-09-09
Description: >-
  Application Template: Crea la infraestructura básica (DB, S3, VPC, SNS)


######################
# Parameters section
######################

Parameters:
  # AMI
  AmazonLinuxAMIID:
    Type: AWS::SSM::Parameter::Value<AWS::EC2::Image::Id>
    Default: /aws/service/ami-amazon-linux-latest/amzn2-ami-hvm-x86_64-gp2


######################
# Resources section
######################

Resources:
```

```
################
# DynamoDB TABLE
################

myDynamoDBTable:
  Type: AWS::DynamoDB::Table
  Properties:
    AttributeDefinitions:
      - AttributeName: monthYear
        AttributeType: S
    KeySchema:
      - AttributeName: monthYear
        KeyType: HASH
    ProvisionedThroughput:
      ReadCapacityUnits: 20 #!Ref ReadCapacityUnits
      WriteCapacityUnits: 20
    PointInTimeRecoverySpecification:
      PointInTimeRecoveryEnabled: true
    TableName: Measures

#########
# Bucket
#########
S3Bucket:
  Type: AWS::S3::Bucket
  Properties:
    BucketName: !Sub "proy-marmenor-codebucket-${AWS::AccountId}"


#####################
#VPC
####################
VPC:
  Type: AWS::EC2::VPC
  Properties:
    EnableDnsSupport: true
    EnableDnsHostnames: true
    CidrBlock: 192.168.0.0/16
    Tags:
      - Key: Name
        Value: proy-marmenor-vpc

## Internet Gateway
InternetGateway:
  Type: AWS::EC2::InternetGateway

VPCGatewayAttachment:
  Type: AWS::EC2::VPCGatewayAttachment
  Properties:
    VpcId: !Ref VPC
    InternetGatewayId: !Ref InternetGateway
```

```
# NAT Gateway y Elastic IP
NatGatewayEIP:
  Type: AWS::EC2::EIP
  DependsOn: VPCGatewayAttachment
  Properties:
    Domain: vpc
    Tags:
      - Key: Name
        Value: proy-Nat-eip

NatGateway:
  Type: AWS::EC2::NatGateway
  Properties:
    AllocationId: !GetAtt NatGatewayEIP.AllocationId
    SubnetId: !Ref PublicSubnet1
    Tags:
      - Key: Name
        Value: !Sub proy-nat-gateway

## Public Route Table
PublicRouteTable:
  Type: AWS::EC2::RouteTable
  Properties:
    VpcId: !Ref VPC
    Tags:
      - Key: Name
        Value: proy-Public-rt

PublicRoute:
  Type: AWS::EC2::Route
  DependsOn: VPCGatewayAttachment
  Properties:
    RouteTableId: !Ref PublicRouteTable
    DestinationCidrBlock: 0.0.0.0/0
    GatewayId: !Ref InternetGateway

PrivateRouteTable:
    Type: AWS::EC2::RouteTable
    Properties:
      VpcId: !Ref VPC
      Tags:
        - Key: Name
          Value: proy-Private-rt

PrivateRoute:
  Type: AWS::EC2::Route
  Properties:
    RouteTableId: !Ref PrivateRouteTable
    DestinationCidrBlock: 0.0.0.0/0
    NatGatewayId: !Ref NatGateway

## Public Subnet
```

```
PublicSubnet1:
  Type: AWS::EC2::Subnet
  Properties:
    VpcId: !Ref VPC
    CidrBlock: 192.168.0.0/24
    AvailabilityZone: !Select
      - 0
      - !GetAZs
        Ref: AWS::Region
    Tags:
      - Key: Name
        Value: proy-PublicSubnet1

PublicSubnetRouteTableAssociation:
  Type: AWS::EC2::SubnetRouteTableAssociation
  Properties:
    SubnetId: !Ref PublicSubnet1
    RouteTableId: !Ref PublicRouteTable

# segunda public subnet
PublicSubnet2:
  Type: AWS::EC2::Subnet
  Properties:
    VpcId: !Ref VPC
    CidrBlock: 192.168.3.0/24
    AvailabilityZone: !Select
      - 1
      - !GetAZs
        Ref: AWS::Region
    Tags:
      - Key: Name
        Value: proy-PublicSubnet2

PublicSubnetRouteTableAssociation2:
  Type: AWS::EC2::SubnetRouteTableAssociation
  Properties:
    SubnetId: !Ref PublicSubnet2
    RouteTableId: !Ref PublicRouteTable

# privates subnets:
PrivateSubnet1:
  Type: AWS::EC2::Subnet
  Properties:
    VpcId: !Ref VPC
    CidrBlock: 192.168.1.0/24
    AvailabilityZone: !Select
      - 0
      - !GetAZs
        Ref: AWS::Region
    Tags:
      - Key: Name
        Value: proy-PrivateSubnet1
```

```
PrivateSubnetRouteTableAssociation:
  Type: AWS::EC2::SubnetRouteTableAssociation
  Properties:
    SubnetId: !Ref PrivateSubnet1
    RouteTableId: !Ref PrivateRouteTable

PrivateSubnet2:
  Type: AWS::EC2::Subnet
  Properties:
    VpcId: !Ref VPC
    CidrBlock: 192.168.2.0/24
    AvailabilityZone: !Select
      - 1
      - !GetAZs
        Ref: AWS::Region
    Tags:
      - Key: Name
        Value: proy-PrivateSubnet2

PrivateSubnetRouteTableAssociation2:
  Type: AWS::EC2::SubnetRouteTableAssociation
  Properties:
    SubnetId: !Ref PrivateSubnet2
    RouteTableId: !Ref PrivateRouteTable

## SecurityGroup
ALBSecurityGroup:
  Type: AWS::EC2::SecurityGroup
  Properties:
    GroupDescription: Grupo de seguridad para el balanceador de carga
    GroupName: proy-alb-sg
    VpcId: !Ref VPC
    SecurityGroupIngress: # reglas de entrada
      - IpProtocol: tcp
        FromPort: 80
        ToPort: 80
        CidrIp: 0.0.0.0/0
        Description: Allow HTTP
    SecurityGroupEgress: # trafico de salida
      - IpProtocol: -1
        CidrIp: 0.0.0.0/0
        Description: Allow all outbound traffic
    Tags:
      - Key: Name
        Value: proy-alb-sg

PrivateFlaskSecurityGroup:
  Type: AWS::EC2::SecurityGroup
  Properties:
    GroupDescription: Grupo de seguridad para las instancias
    GroupName: proy-instances-sg
```

```
    VpcId: !Ref VPC
    SecurityGroupIngress: # reglas de entrada
      - IpProtocol: tcp
        FromPort: 8080
        ToPort: 8080
        SourceSecurityGroupId: !Ref ALBSecurityGroup
        Description: Trafico entre instancias y el balanceador
    SecurityGroupEgress: # trafico de salida
      - IpProtocol: -1
        CidrIp: 0.0.0.0/0
        Description: Allow all outbound traffic
    Tags:
      - Key: Name
        Value: proy-instances-sg

  # Grupo de seguridad temporal para la DockerInstance (SSH y testeo local)
InstanceSecurityGroup:
  Type: AWS::EC2::SecurityGroup
  Properties:
    GroupName: proy-instanceSecurityGroup
    GroupDescription: Acceso SSH y testeo Flask
    VpcId: !Ref VPC
    SecurityGroupIngress:
      - IpProtocol: tcp
        FromPort: 22
        ToPort: 22
        CidrIp: 0.0.0.0/0
      - IpProtocol: tcp
        FromPort: 8080
        ToPort: 8080
        CidrIp: 0.0.0.0/0


## Instances:
DokerInstace:
  Type: AWS::EC2::Instance
  DependsOn: S3Bucket
  Properties:
    InstanceType: t3.micro
    ImageId: !Ref AmazonLinuxAMIID
    KeyName: proy-marmenor
    IamInstanceProfile: "LabInstanceProfile"
    NetworkInterfaces:
      - GroupSet:
          - !Ref InstanceSecurityGroup
        AssociatePublicIpAddress: true
        DeviceIndex: 0
        DeleteOnTermination: true
        SubnetId: !Ref PublicSubnet1
    UserData:
      Fn::Base64: !Sub |
        #!/bin/bash
```

```
        # Exportar las variables de CloudFormation al entorno de la EC2
        export DYNAMODB_TABLE_NAME="${myDynamoDBTable}"
        export AWS_REGION="${AWS::Region}"
        # Instalar AWS CLI y descargar el script
        yum install -y aws-cli
        wget -O /tmp/docker_setup.sh https://raw.githubusercontent.com/franjavi-upct-es/Proyecto-ICAP
            -AquaSense/main/Plantillas%20CloudFormation%20IaC/docker_setup.sh

        # Hacer el script ejecutable y ejecutarlo
        chmod +x /tmp/docker_setup.sh
        /tmp/docker_setup.sh
    InstanceInitiatedShutdownBehavior: terminate
    Tags:
      - Key: Name
        Value: proy-DockerInstance



######################
# Outputs section
######################
Outputs:

  VPCId:
    Value: !Ref VPC
    Export:
      Name: !Sub "${AWS::StackName}-VPCId"

  PublicSubnet1Id:
    Value: !Ref PublicSubnet1
    Export:
      Name: !Sub "${AWS::StackName}-PublicSubnet1Id"

  PublicSubnet2Id:
    Value: !Ref PublicSubnet2
    Export:
      Name: !Sub "${AWS::StackName}-PublicSubnet2Id"

  PrivateSubnet1Id:
    Value: !Ref PrivateSubnet1
    Export:
      Name: !Sub "${AWS::StackName}-PrivateSubnet1Id"

  PrivateSubnet2Id:
    Value: !Ref PrivateSubnet2
    Export:
      Name: !Sub "${AWS::StackName}-PrivateSubnet2Id"

  PrivateFlaskSecurityGroupId:
    Value: !Ref PrivateFlaskSecurityGroup
    Export:
      Name: !Sub "${AWS::StackName}-PrivateFlaskSG"
```

```
  ALBSecurityGroupId:
    Value: !Ref ALBSecurityGroup
    Export:
      Name: !Sub "${AWS::StackName}-ALBSecurityGroup"

  DynamoDBTableName:
    Value: !Ref myDynamoDBTable
    Export:
      Name: !Sub "${AWS::StackName}-DynamoDBTable"
```

6.2.2 B.2 Pipeline Lambda

Archivo: `proy-lambda_bucket.yaml`

Recursos creados:

- Función Lambda: `proy-marmenor-process-csv`
- Bucket S3: `proy-marmenor-csv-raw`
- Topic SNS: `proy-marmenor-alerts`
- Permisos de invocación S3 → Lambda

```
AWSTemplateFormatVersion: 2010-09-09
Description: >-
  Application Template: Crea la lambda y el bucket s3.

Parameters:
  Email:
    Type: String
    ConstraintDescription: email para el sns
    Default: javier.moreno@edu.upct.es

  NetworkStackName:
    Description: Nombre de la pila de infraestructura base
    Type: String
    Default: proy-infraestructura-soporte




Resources:
 ####################
  # Lambda Function
  ##################
  ProcessS3FileLambda:
    Type: AWS::Lambda::Function
    Properties:
      FunctionName: proy-marmenor-process-csv
      Runtime: python3.11
      Handler: funcion_lambda.lambda_handler
      # REPLACE THE ROLE IMPORT WITH THIS:
      Role: !Sub "arn:aws:iam::${AWS::AccountId}:role/LabRole"
```

```
    Timeout: 60
    MemorySize: 256
    Code:
      S3Bucket: !Sub "proy-marmenor-codebucket-${AWS::AccountId}"
      S3Key: lambda.zip
    Environment:
      Variables: # Variables de entorno
        SNS_TOPIC_ARN: !Ref MarMenorAlertsTopic
        DYNAMODB_TABLE:
         Fn::ImportValue:
            !Sub "${NetworkStackName}-DynamoDBTable"
 #########
 # Bucket
 #########
 S3Bucket:
   Type: AWS::S3::Bucket
   Properties:
     BucketName: !Sub "proy-marmenor-csv-raw-${AWS::AccountId}"
     NotificationConfiguration:
       LambdaConfigurations:
         - Event: s3:ObjectCreated:Put
           Function: !GetAtt ProcessS3FileLambda.Arn # Referencia la nueva Lambda


##########################
 # Permiso de invocación
 ##########################
 LambdaInvokePermission:
   Type: AWS::Lambda::Permission
   Properties:
     FunctionName: !Ref ProcessS3FileLambda # Referencia la nueva Lambda
     Action: lambda:InvokeFunction
     Principal: s3.amazonaws.com
     SourceArn: !Sub arn:${AWS::Partition}:s3:::proy-marmenor-csv-raw-${AWS::AccountId}


 ########################
 # SNS
 ########################
 MarMenorAlertsTopic:
   Type: AWS::SNS::Topic
   Properties:
     TopicName: proy-marmenor-alerts
     DisplayName: "Alerta Mar Menor"
     Subscription:
       - Protocol: email
         Endpoint: !Ref Email
     Tags:
       - Key: Proyecto
         Value: AquaSenseCloud
       - Key: Entorno
         Value: Producción
```

```
Outputs:
  LambdaFunctionName:
    Description: El nombre de la función Lambda creada.
    Value: !Ref ProcessS3FileLambda
```

6.2.3 B.3 Clúster ECS

Archivo: `proy-infraestructura-cluster.yaml`

Recursos creados:

- ECS Cluster: `proy-marmenor-cluster`
- Application Load Balancer + Listener
- Target Group (Health check: `/health`)
- Task Definition (awsvpc, 256 CPU, 256 RAM)
- ECS Service (Launch Type: EC2, DesiredCount: 3)
- Auto Scaling Group (min: 2, max: 5)

```
AWSTemplateFormatVersion: 2010-09-09
Description: Plantilla que crea la infraestructura ECS EC2 con awsvpc

Parameters:
  NetworkStackName:
    Description: Nombre de la pila de infraestructura base
    Type: String
    Default: proy-infraestructura-soporte

Resources:
  ####################
  # ECS Cluster
  ####################
  ECSCluster:
    Type: AWS::ECS::Cluster
    Properties:
      ClusterName: proy-marmenor-cluster

  ECSLaunchConfiguration:
    Type: AWS::AutoScaling::LaunchConfiguration
    Properties:
      ImageId: !Sub "{{resolve:ssm:/aws/service/ecs/optimized-ami/amazon-linux-2/recommended/image_id}}
          "
      InstanceType: t3.micro
      IamInstanceProfile: LabInstanceProfile
      SecurityGroups:
        - Fn::ImportValue: !Sub "${NetworkStackName}-PrivateFlaskSG"
      UserData:
        Fn::Base64: !Sub |
          #!/bin/bash
```

```
        echo ECS_CLUSTER=${ECSCluster} >> /etc/ecs/ecs.config

ECSAutoScalingGroup:
  Type: AWS::AutoScaling::AutoScalingGroup
  Properties:
    MinSize: 2
    MaxSize: 5
    DesiredCapacity: 3
    VPCZoneIdentifier:
      - Fn::ImportValue: !Sub "${NetworkStackName}-PrivateSubnet1Id"
      - Fn::ImportValue: !Sub "${NetworkStackName}-PrivateSubnet2Id"
    LaunchConfigurationName: !Ref ECSLaunchConfiguration
    Tags:
      - Key: Name
        Value: proy-ECS-I
        PropagateAtLaunch: true

###################################
# Load Balancer
###################################
ApplicationLoadBalancer:
  Type: AWS::ElasticLoadBalancingV2::LoadBalancer
  Properties:
    Name: proy-marmenor-alb
    Subnets:
      - Fn::ImportValue: !Sub "${NetworkStackName}-PublicSubnet1Id"
      - Fn::ImportValue: !Sub "${NetworkStackName}-PublicSubnet2Id"
    SecurityGroups:
      - Fn::ImportValue: !Sub "${NetworkStackName}-ALBSecurityGroup"
    Scheme: internet-facing
    Type: application

###################################
# Target Group
###################################
ECSTargetGroup:
  Type: AWS::ElasticLoadBalancingV2::TargetGroup
  Properties:
    Name: proy-tg
    VpcId:
      Fn::ImportValue: !Sub "${NetworkStackName}-VPCId"
    Port: 8080
    Protocol: HTTP
    TargetType: ip
    HealthCheckPath: /health
    HealthCheckPort: "8080"
    HealthCheckProtocol: HTTP
    HealthCheckIntervalSeconds: 30
    HealthCheckTimeoutSeconds: 5
    HealthyThresholdCount: 2
    UnhealthyThresholdCount: 3
    Matcher:
```

```
      HttpCode: "200"

###################################
# Listener
###################################
ALBListener:
  Type: AWS::ElasticLoadBalancingV2::Listener
  Properties:
    LoadBalancerArn: !Ref ApplicationLoadBalancer
    Port: 80
    Protocol: HTTP
    DefaultActions:
      - Type: forward
        TargetGroupArn: !Ref ECSTargetGroup

###################################
# Task
###################################
ECSTaskDefinitionEC2:
  Type: AWS::ECS::TaskDefinition
  Properties:
    Family: aquasense-api-task-ec2
    Cpu: 256
    Memory: 256
    NetworkMode: awsvpc
    RequiresCompatibilities:
      - EC2
    # Rol dinámico usando AccountId
    ExecutionRoleArn: !Sub "arn:aws:iam::${AWS::AccountId}:role/LabRole"
    TaskRoleArn: !Sub "arn:aws:iam::${AWS::AccountId}:role/LabRole"
    ContainerDefinitions:
      - Name: proy-aquasense-api
        # IMAGEN AUTOMATIZADA AQUÍ:
        Image: !Sub "${AWS::AccountId}.dkr.ecr.${AWS::Region}.amazonaws.com/aquasense-api:latest"
        PortMappings:
          - ContainerPort: 8080
        Environment:
          - Name: DYNAMODB_TABLE
            Value:
              Fn::ImportValue: !Sub "${NetworkStackName}-DynamoDBTable"
          - Name: AWS_REGION
            Value: !Ref AWS::Region
        Essential: true
        HealthCheck:
          Command:
            ["CMD-SHELL", "curl -f http://localhost:8080/health || exit 1"]
          Interval: 30
          Timeout: 10
          Retries: 5
          StartPeriod: 60

###################################
```

```
  # ECS Service
  ###################################
  ECSServiceEC2:
    Type: AWS::ECS::Service
    DependsOn: ALBListener
    Properties:
      Cluster: !Ref ECSCluster
      LaunchType: EC2
      DesiredCount: 3
      TaskDefinition: !Ref ECSTaskDefinitionEC2
      NetworkConfiguration:
        AwsvpcConfiguration:
          AssignPublicIp: DISABLED
          Subnets:
            - Fn::ImportValue: !Sub "${NetworkStackName}-PrivateSubnet1Id"
            - Fn::ImportValue: !Sub "${NetworkStackName}-PrivateSubnet2Id"
          SecurityGroups:
            - Fn::ImportValue: !Sub "${NetworkStackName}-PrivateFlaskSG"
      LoadBalancers:
        - ContainerName: proy-aquasense-api
          ContainerPort: 8080
          TargetGroupArn: !Ref ECSTargetGroup

######################
# Outputs section
######################

Outputs:
  LoadBalancerURL:
    Description: URL del Application Load Balancer para acceder a la API de AquaSense
    Value: !Sub "http://${ApplicationLoadBalancer.DNSName}"
    Export:
      Name: !Sub "${AWS::StackName}-LoadBalancerURL"

  APIEndpointTemp:
    Description: Endpoint para consultar temperatura media mensual
    Value: !Sub "http://${ApplicationLoadBalancer.DNSName}/temp?month=3&year=2017"

  APIEndpointSD:
    Description: Endpoint para consultar desviación estándar máxima mensual
    Value: !Sub "http://${ApplicationLoadBalancer.DNSName}/sd?month=3&year=2017"

  APIEndpointMaxDiff:
    Description: Endpoint para consultar diferencia de temperatura máxima mensual
    Value: !Sub "http://${ApplicationLoadBalancer.DNSName}/maxdiff?month=4&year=2017"

  APIEndpointMonths:
    Description: Endpoint para listar todos los meses disponibles
    Value: !Sub "http://${ApplicationLoadBalancer.DNSName}/months"

  APIEndpointHealth:
    Description: Endpoint de health check del servicio
```

```
    Value: !Sub "http://${ApplicationLoadBalancer.DNSName}/health"

  LoadBalancerDNS:
    Description: DNS del Load Balancer (sin protocolo)
    Value: !GetAtt ApplicationLoadBalancer.DNSName
    Export:
      Name: !Sub "${AWS::StackName}-LoadBalancerDNS"
```

6.3 Anexo C - Configuración Docker

6.3.1 C.1 Dockerfile

Archivo: `Dockerfile.txt`

Características:

- Imagen base: `python:3.11-slim`
- Servidor: Gunicorn (2 workers, 4 threads)
- Puerto expuesto: 8080
- Health check: curl a `/health` cada 30s
- Usuario no-root (appuser:1000)

```
# Dockerfile para AquaSenseCloud API
# Imagen optimizada para ECS Fargate

# Imagen base oficial de Python
FROM python:3.11-slim

# Metadata
LABEL maintainer="aquasensecloud@upct.es"
LABEL description="API REST para consultar datos del Mar Menor"
LABEL version="1.0"
LABEL project="AquaSenseCloud - ICAP UPCT"

# Variables de entorno
ENV PYTHONUNBUFFERED=1 \
  PYTHONDONTWRITEBYTECODE=1 \
  PORT=8080 \
  AWS_DEFAULT_REGION=us-east-1

# Instalar dependencias del sistema necesarias
RUN apt-get update && \
  apt-get install -y --no-install-recommends \
  curl \
  && rm -rf /var/lib/apt/lists/*

# Crear directorio de trabajo
WORKDIR /app

# Copiar requirements primero (para aprovechar cache de Docker)
COPY requirements.txt .
```

```
# Instalar dependencias de Python
RUN pip install --no-cache-dir --upgrade pip && \
  pip install --no-cache-dir -r requirements.txt

# Copiar código de la aplicación
COPY aquasense.py .

# Cerar usuario no-root para seguridad
RUN useradd -m -u 1000 appuser && \
  chown -R appuser:appuser /app

# Cambiar a usuario no-root
USER appuser

# Exponer puerto
EXPOSE 8080

# Health check
HEALTHCHECK --interval=30s --timeout=5s --start-period=10s --retries=3 \
  CMD curl -f http://localhost:8080/health || exit 1

# Comando de inicio (usando gunicorn para producción)
CMD [ "gunicorn", "--bind", "0.0.0.0:8080", "--workers", "2", "--threads", "4", "--timeout", "60", "--
    access-logfile", "-", "--error-logfile", "-", "aquasense:app" ]
```

6.3.2 C.2 Script de Automatización

Archivo: `docker_setup.sh`

Acciones:

1. Instalación y configuración de Docker
2. Descarga de código desde GitHub
3. Construcción de imagen Docker
4. Creación de repositorio ECR
5. Autenticación en ECR
6. Push de imagen a ECR
7. Descarga del código de la función Lambda (`funcion_lambda.py`) y compresión en `lambda.zip`
8. Subida de `lambda.zip` al bucket S3 de código (`proy-marmenor-codebucket-<ACCOUNT_ID>`) para que CloudFormation pueda desplegar la Lambda

```
#!/bin/bash
# Script de configuración e inicio de Docker para AquaSenseCloud API

# --- 1. INSTALACIÓN Y CONFIGURACIÓN DE DOCKER ---
yum update -y
yum -y install docker
systemctl start docker
```

```
systemctl enable docker
usermod -a -G docker ec2-user

# Esperar un momento para que Docker se inicialice (necesario en algunos entornos)
sleep 5

#   Variables
export AWS_DEFAULT_REGION="us-east-1"
export REGION=$AWS_DEFAULT_REGION
ACCOUNT_ID=$(aws sts get-caller-identity --query Account --output text)
BUCKET=$(aws s3api list-buckets \
  --query "Buckets[?starts_with(Name, 'proy-marmenor-codebucket-$ACCOUNT_ID')].Name" \
  --output text)

# Descarga la lambda y la comprimo:
wget https://raw.githubusercontent.com/franjavi-upct-es/Proyecto-ICAP-AquaSense/main/
    Ingesta%20Datos%20y%20Pipeline/funcion_lambda.py


zip lambda.zip funcion_lambda.py

# Subir la lambda al bucket
aws s3 cp lambda.zip s3://$BUCKET/lambda.zip

# Crear directorio de trabajo y moverse a el
mkdir -p /app
cd /app

# 2. Descargar requirements.txt desde GitHub
wget -O requirements.txt https://raw.githubusercontent.com/franjavi-upct-es/Proyecto-ICAP
    -AquaSense/refs/heads/main/Servidor%20Web%20y%20Containers/requirements.txt

# 3. Descargar Dockerfile desde GitHub
wget -O Dockerfile.txt https://raw.githubusercontent.com/franjavi-upct-es/Proyecto-ICAP-
    AquaSense/refs/heads/main/Servidor%20Web%20y%20Containers/Dockerfile.txt
mv Dockerfile.txt Dockerfile

# 4. Descargar aquasense.py desde GitHub
wget -O aquasense.py https://raw.githubusercontent.com/franjavi-upct-es/Proyecto-ICAP-
    AquaSense/refs/heads/main/Servidor%20Web%20y%20Containers/aquasense.py

# --- 5. CONSTRUCCIÓN DE LA IMAGEN DOCKER ---
docker build -t aquasense-api .

# --- 6. EJECUCIÓN DEL CONTENEDOR DOCKER ---
# Estas variables se inyectarán desde CloudFormation a la EC2, y luego
# se pasan al contenedor.
DDB_TABLE_NAME="${DYNAMODB_TABLE_NAME}"
AWS_REGION="${AWS_REGION}"

# docker run -d \
#   --name aquasense-web \
```

```
#   -p 8080:8080 \
#   -e DYNAMODB_TABLE=$DDB_TABLE_NAME \
#   -e AWS_REGION=$AWS_REGION \
#   aquasense-api


aws ecr create-repository --repository-name aquasense-api

aws ecr get-login-password --region $REGION \
  | docker login --username AWS --password-stdin $ACCOUNT_ID.dkr.ecr.$REGION.amazonaws.
      com

docker tag aquasense-api:latest $ACCOUNT_ID.dkr.ecr.$REGION.amazonaws.com/aquasense-api:
    latest

docker push $ACCOUNT_ID.dkr.ecr.$REGION.amazonaws.com/aquasense-api:latest
```

6.4 Anexo D - Estructura de Datos

6.4.1 D.1 Formato CSV de Entrada

```
Fecha,Medias,Desviaciones
2017/03/22,16.784072875976562,0.28715428709983826
2017/03/30,17.32989501953125,0.4037204384803772
```

Notas:

- Separador: coma (,)
- Encoding: UTF-8
- Formato fecha: YYYY/MM/DD o YYYY-MM-DD
- Headers obligatorios: Fecha, Medias, Desviaciones

6.4.2 D.2 Esquema DynamoDB

Tabla: `Measures`

Estructura:

- **Partition Key**: `monthYear` (String, "YYYY-MM")
- **Atributos**:
  - `max_temp`: Decimal
  - `max_sd`: Decimal
  - `mean_temp`: Decimal
  - `max_diff_temp`: Decimal
  - `mean_temp_count`: Number
  - `last_updated`: String (ISO 8601)

6.5 Anexo E - Ejemplos de Respuestas API

6.5.1 E.1 Endpoint: /temp

```
GET http://[ALB-DNS]/temp?month=3&year=2017

Response (200 OK):
{
  "month": 3,
  "year": 2017,
  "temp": 17.06,
  "max_temp": 17.33,
  "last_updated": "2024-12-07T10:30:00Z"
}
```

6.5.2 E.2 Endpoint: /sd

```
GET http://[ALB-DNS]/sd?month=3&year=2017

Response (200 OK):
{
  "month": 3,
  "year": 2017,
  "sd": 0.6254,
  "last_updated": "2024-12-07T10:30:00Z"
}
```

6.5.3 E.3 Endpoint: /maxdiff

```
GET http://[ALB-DNS]/maxdiff?month=4&year=2017

Response (200 OK):
{
  "month": 4,
  "year": 2017,
  "maxdiff": 2.14,
  "max_temp": 19.47,
  "last_updated": "2024-12-07T10:30:00Z"
}
```