Infraestructura para la Computación de Altas Prestaciones
Proyecto Final: AquaSenseCloud

Francisco Javier Mercader Martínez
Javier Moreno Pagán
Pablo Meseguer Domenech

Curso: 2024/2025

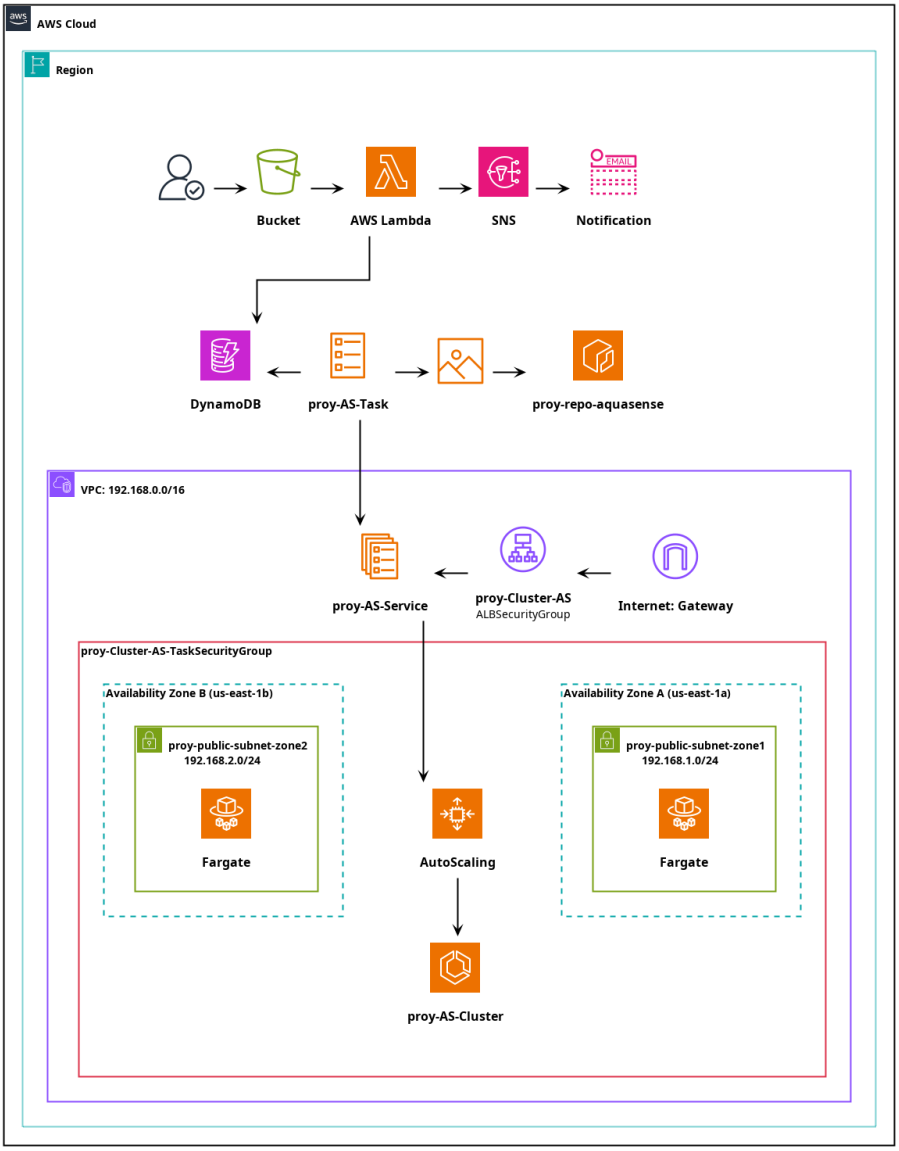
Índice

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):** Configuración de Application Load Balancer (ALB), Target Groups y creación del clúster ECS Fargate.	Javier Moreno	2 días
29 de nov - 1 de dic	**Escalabilidad:** Configuración de Auto Scaling Groups, políticas de escalado basadas en métricas y ejecución de pruebas de 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

Fecha	Tarea	Responsable	Duración
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):** Crea subredes públicas que permiten el acceso a Internet para los recursos dentro de la VPC.
- **Internet Gateway (AWS::EC2::InternetGateway):** Facilita la conexión de la VPC a Internet, permitiendo que los analistas accedan a los servicios configurados y desplegados.

- **Route Table (AWS::EC2::RouteTable):** Define las rutas de tráfico de red dentro de la VPC, asegurando que el tráfico fluya adecuadamente entre las subredes y el Internet Gateway.

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):** Este recurso describe las configuraciones necesarias para ejecutar contenedores en ECS. Especifica detalles como la imagen del contenedor a usar, la cantidad de CPU y memoria asignada, y la compatibildiad con el modo Fargate, que permite ejecutar contenedores sin gestionar servidores subyacentes. Además, define un mapeo de puertos, asignando el puerto anterior del contenedor (5000) al tráfico entrante. Es el punto central de la especificación para las tareas ECS.

  Describe los contenedores que ejecutan la aplicación AquaSense, incluyendo la configuración de recursos (CPU, memoria) y el puerto utilizado (5000).
- **ECSService (AWS::ECS::Service):** Administra la ejecución de las tareas ECS, asegura la ejecución continua de un número deseado de tareas definidas en la configuración de tarea. En este caso, se garantiza que siempre haya dos tareas activas para manejar las solicitudes entrantes. Se utiliza el modo de lanzamiento Fargate para simplificar la gestión

de la infraestructura. El servicio monitorea las tareas y las reinicia automáticamente si fallan, garantizando alta disponibildiad y resiliencia para nuestra aplicación.

9. **Escalado automático**

- **ECS Scalable Target (AWS::ApplicationAutoScaling::ScalableTarget):** Define el rango de capacidad del servicio ECS, asegurando que el número de tareas FARGATE activas esté entre un mínimo de 2 y máximo de 10. Este recurso actúa como el objetivo escalable que la política de escalado monitorea y ajusta según las métricas configuradas. Está asociado al recurso ECSService y permite mantener un nivel óptimo de disponibilidad y desempeño de nuestra aplicación.
- **Scaling Policy (AWS::ApplicationAutoScaling::ScalingPolicy)**
  - **High CPU:** Implementa una política de escalado automático que aumenta la cantidad de tareas activas cuando la utilización promedio de CPU en el clúster ECS supera el 75%. Esto asegura que la capacidad del servicio crezca dinámicamente para manejar un aumento en la carga de trabajo. Incluye un tiempo de enfriamiento (*cooldown*) de 60 segundos para evitar ajustes excesivamente frecuentes.
  - **Low CPU:** Establece una política de reducción de escala que disminuye el número de tareas activas cuando el uso de CPU cae por debajo del 25%. Esto permite optimizar costos al liberar recursos durante periodos de baja demanda, manteniendo solo las tareas necesarias para sorportar la carga en ese momento. Similar a la política de escalado ascendente, aplica un enfriamiento de 60 segundos.

10. **Seguridad**

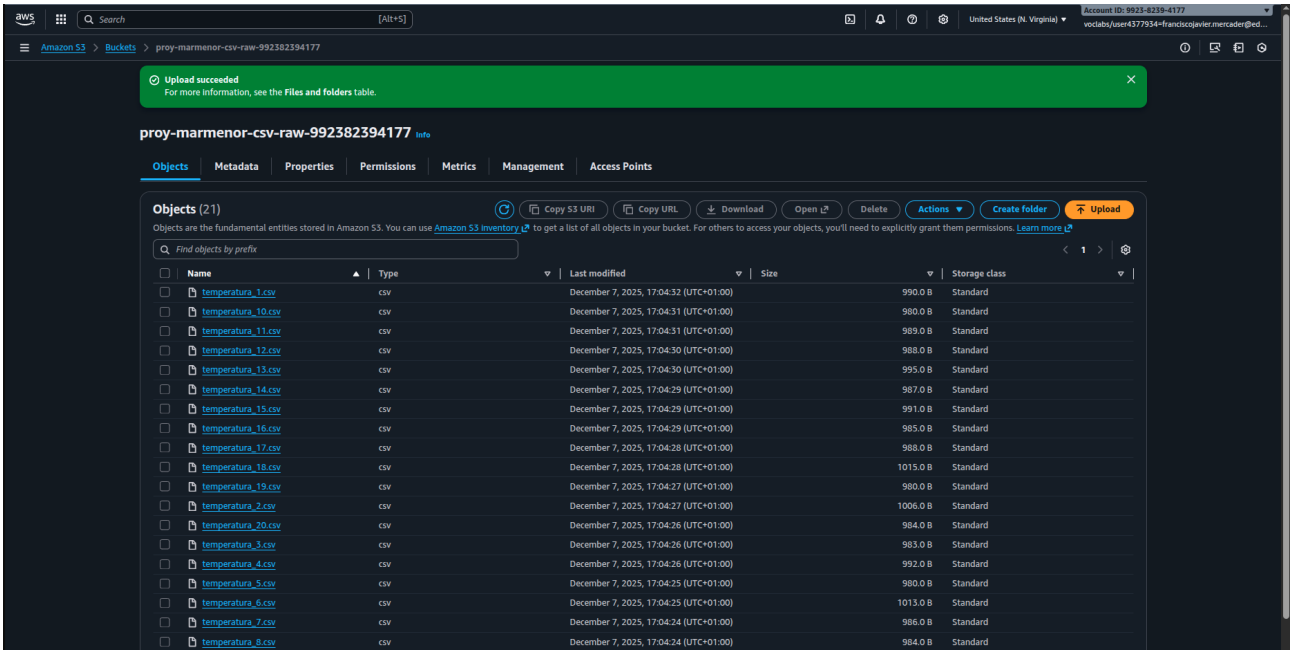
- **ALB Security Group (AWS::EC2::SecurityGroup):** Permite el acceso al balanceador de carga desde Internet (a través del puerto 80-HTTP), asegurando que el tráfico pueda alcanzar la aplicación.
- **Task Security Group (AWS::EC2::SecurityGroup):** Permite el tráfico del ALB a las tareas de ECS (a través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



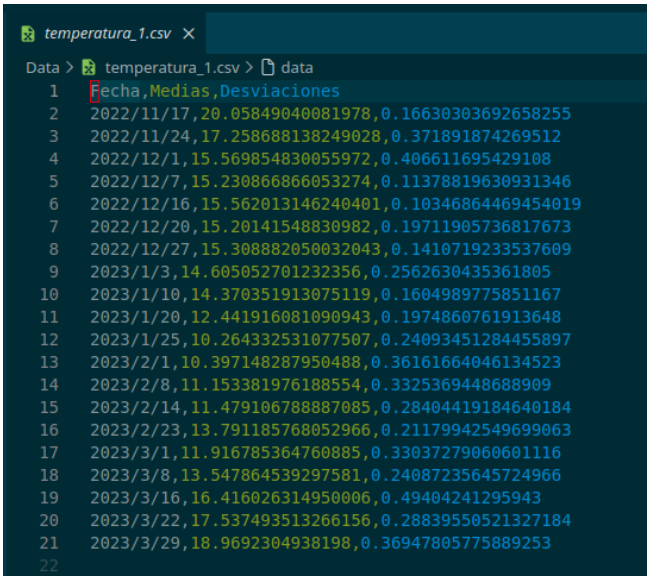


Figure 1: Contenido del fichero: "`temperatura_1.csv`"

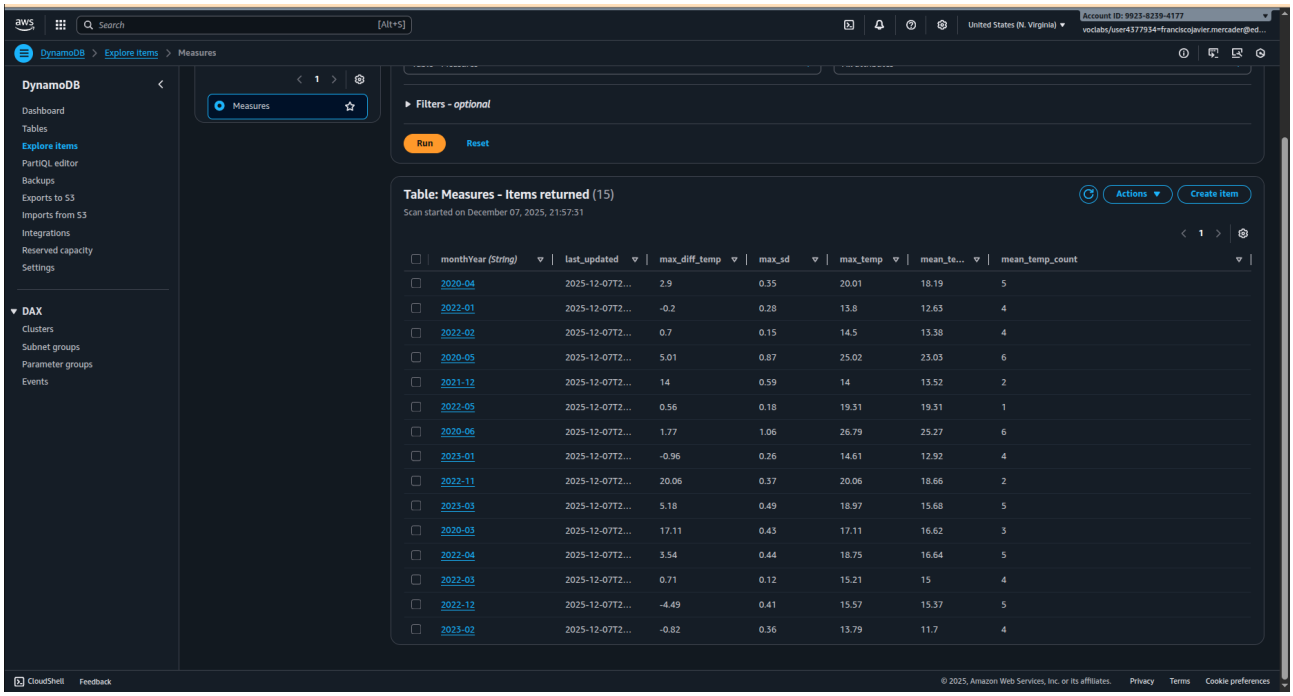


Figure 2: Almacenamiento de datos transformados en DynamoDB

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.

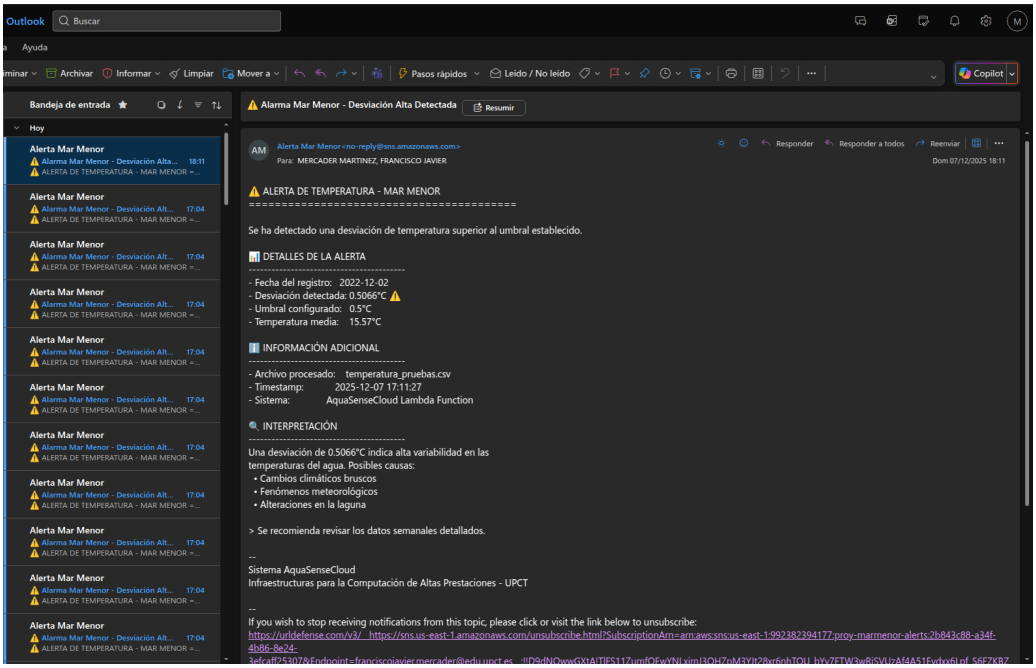
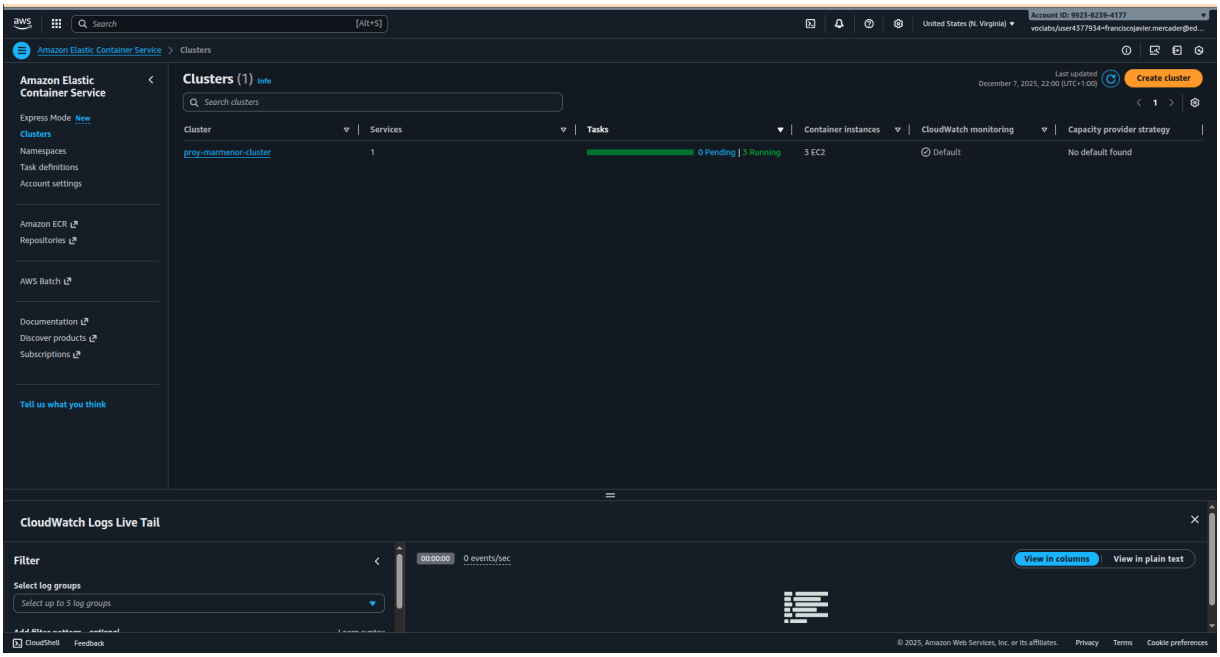
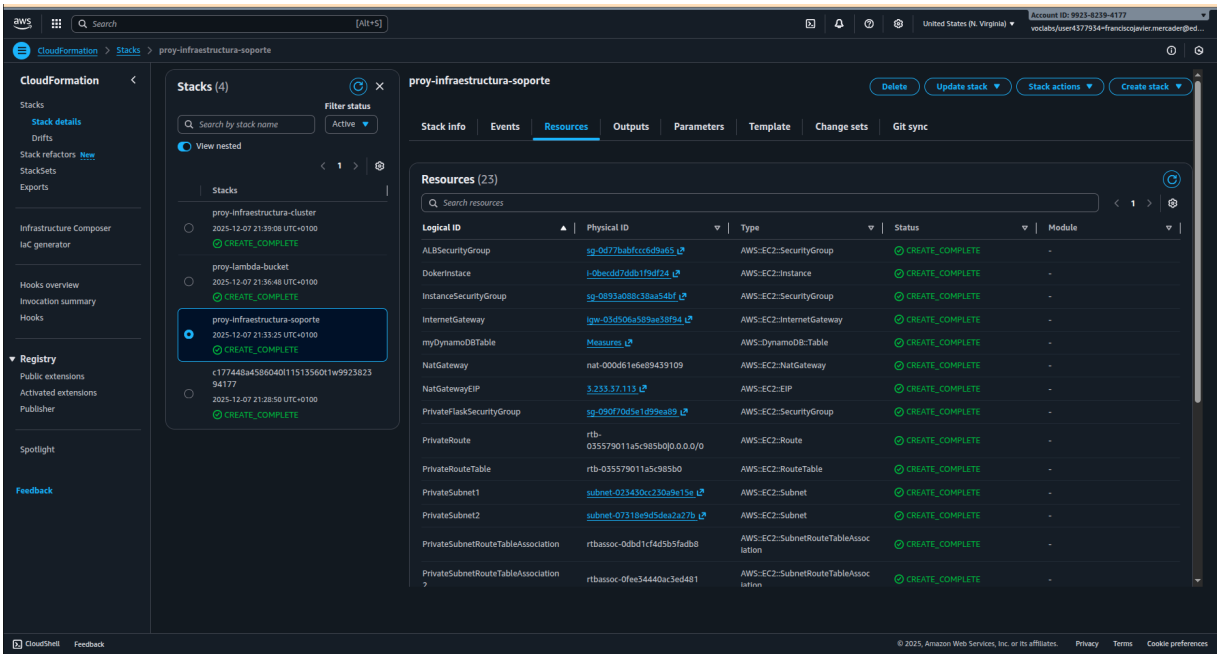


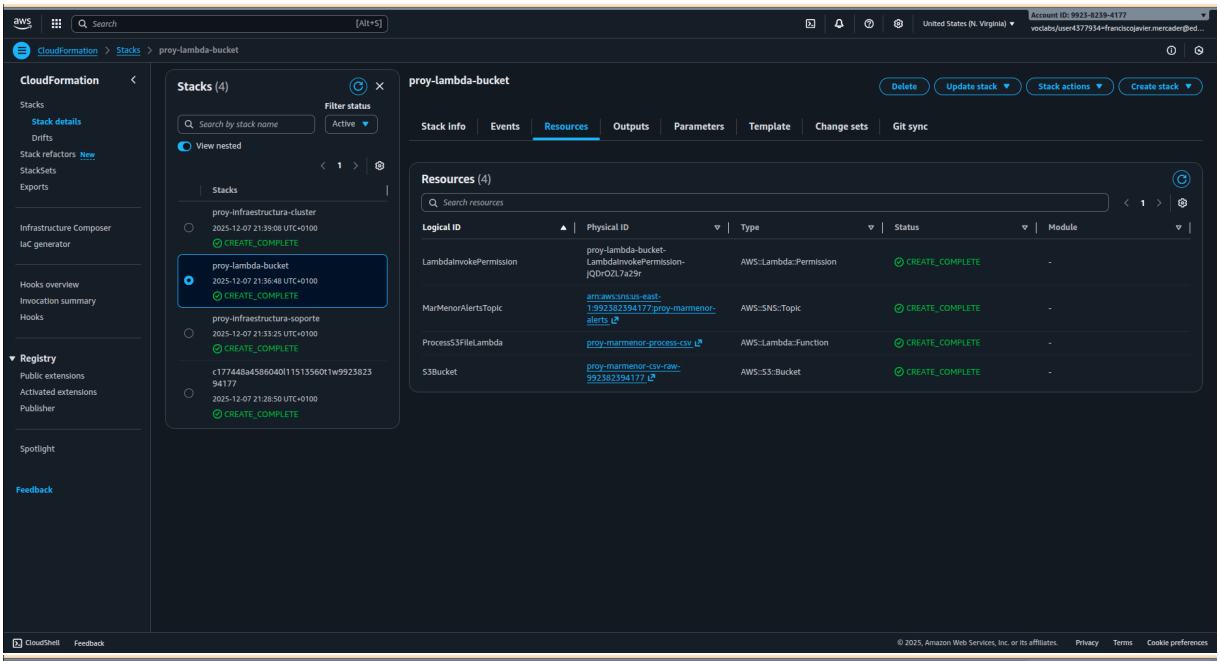
Figure 3: Envío de correo a los analistas mediante un tópico definido mediante AWS SNS

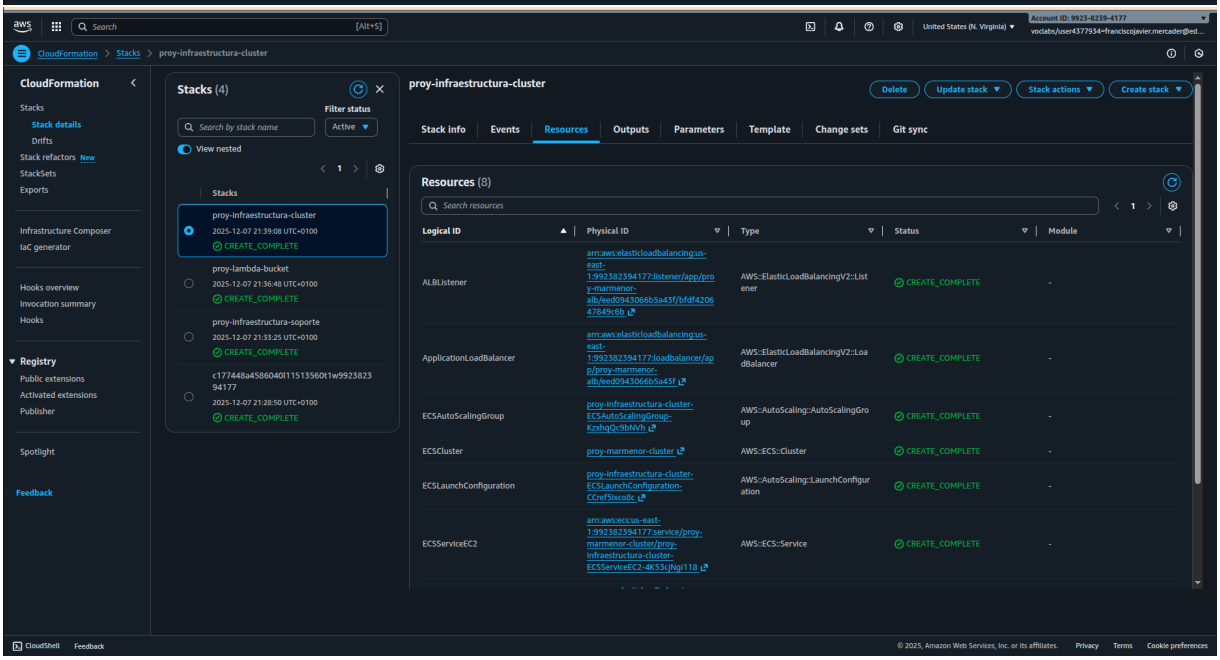
- **ECR:**



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



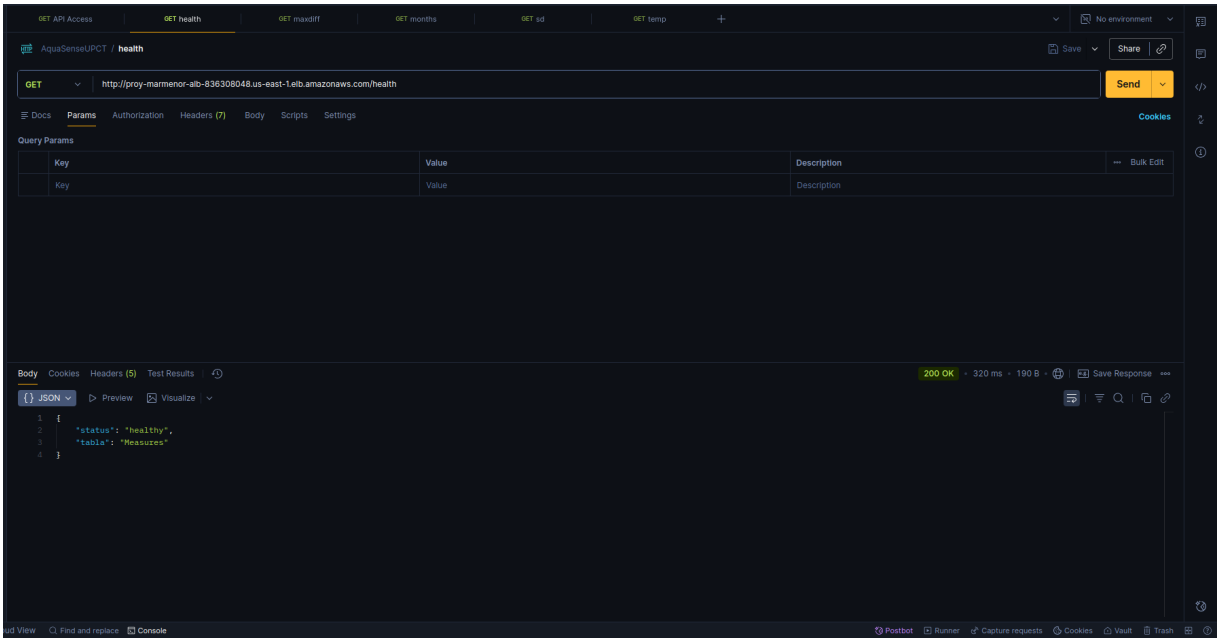




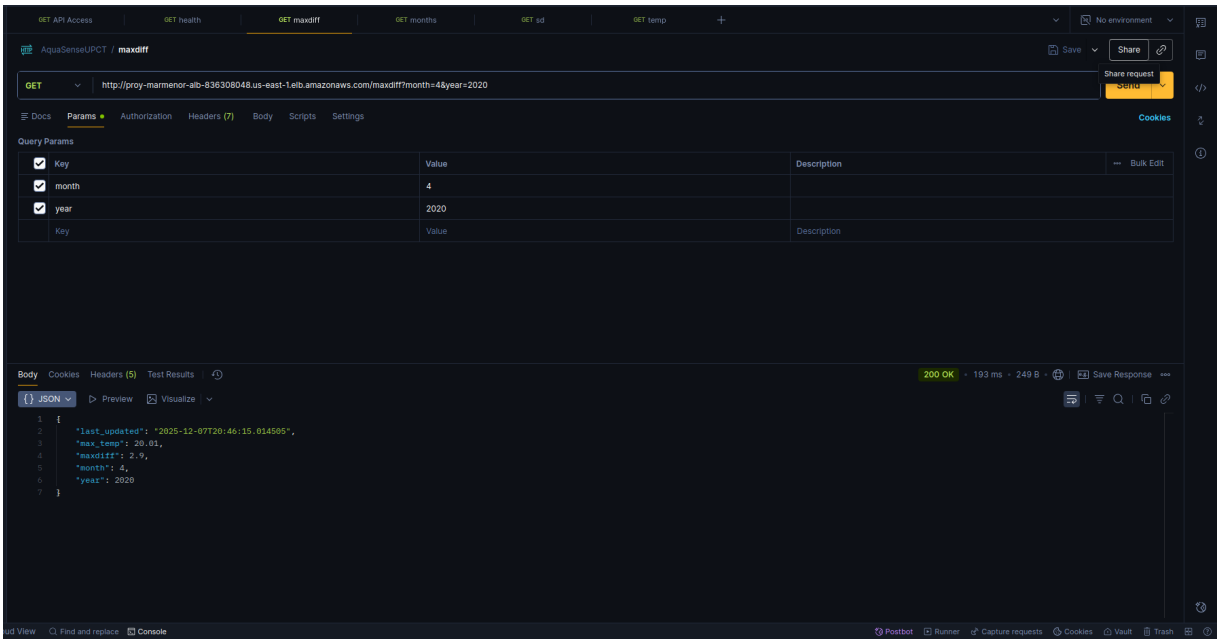
- **Funcionamiento de la aplicación:**

  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`.

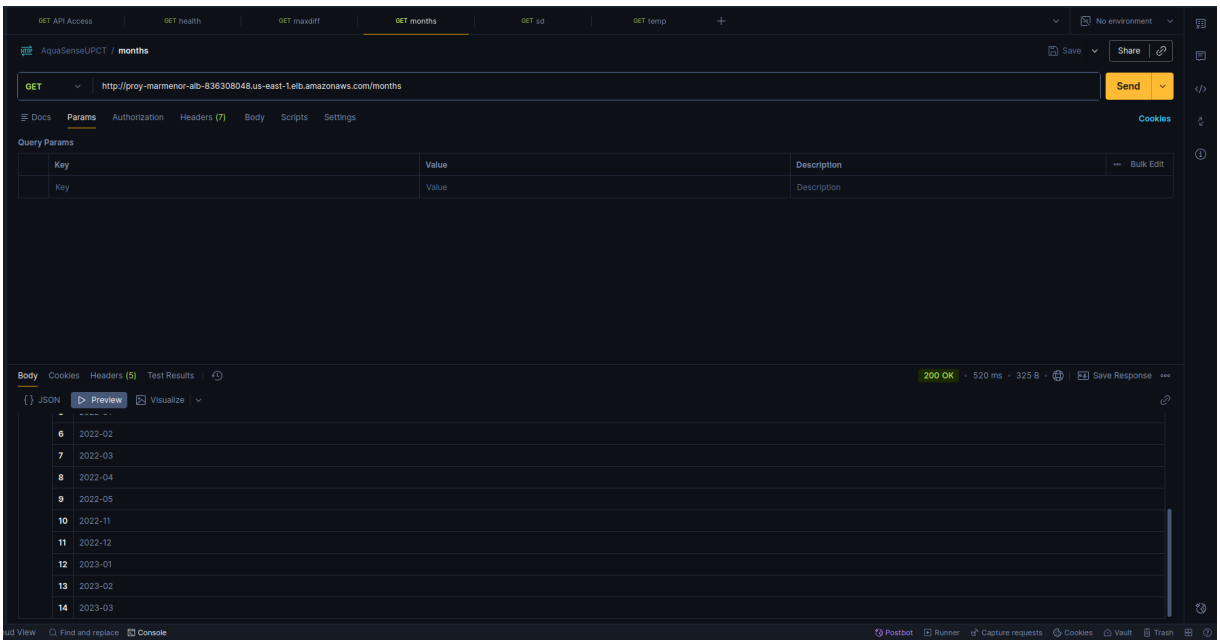
health/



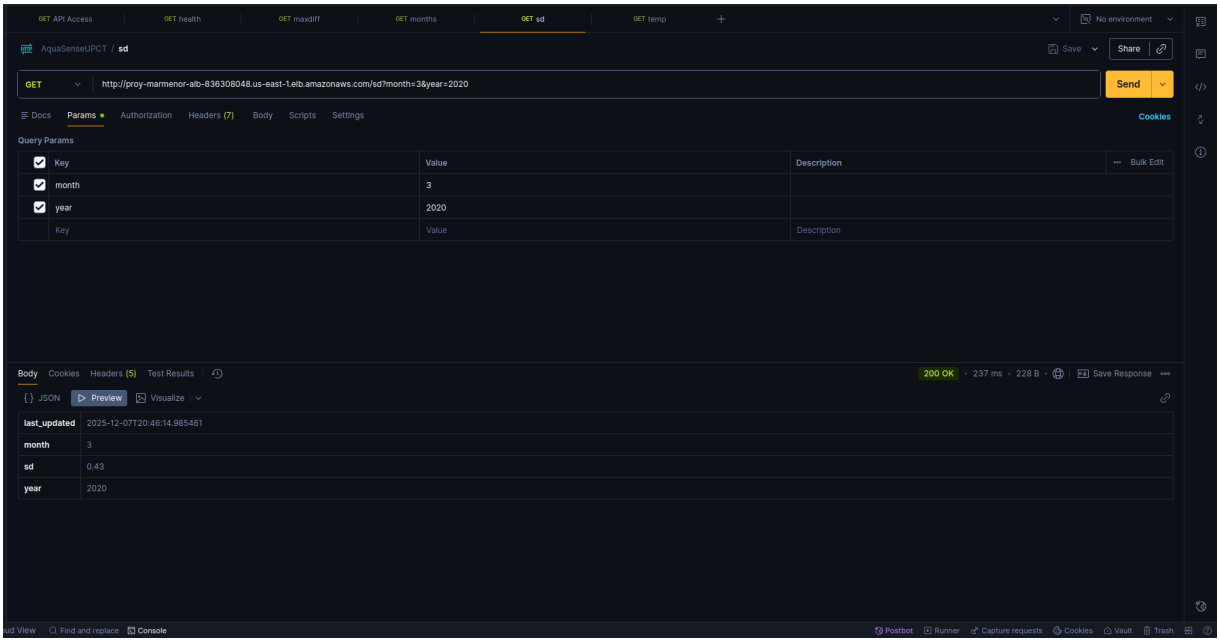
maxdiff/



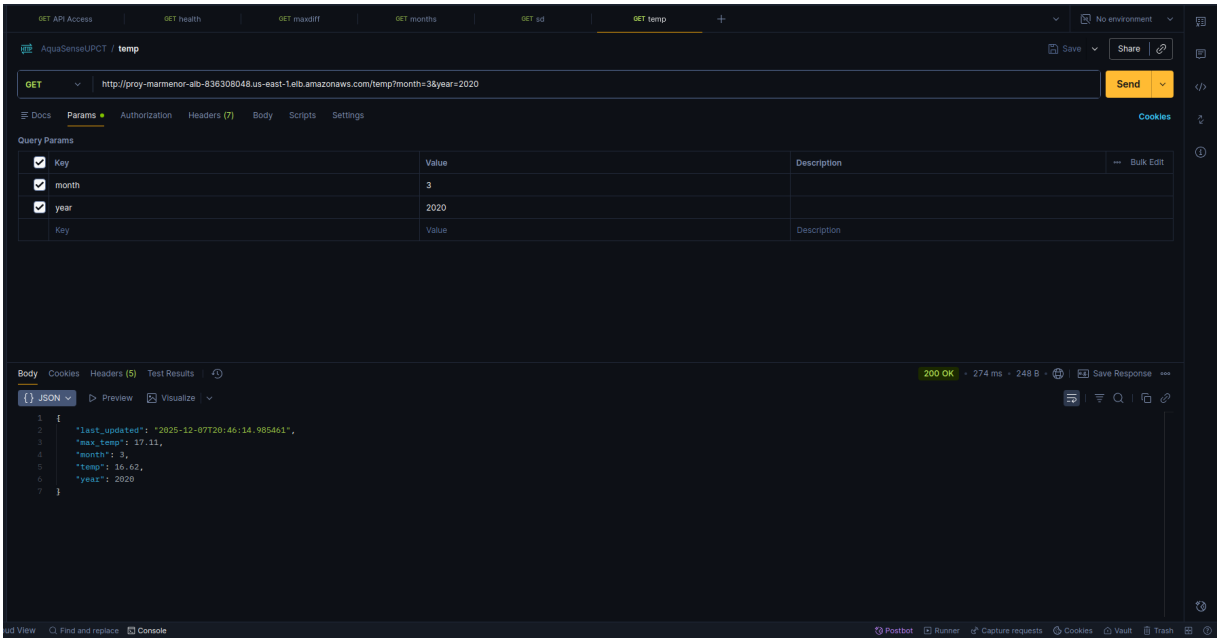
months/


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 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

Ver código completo en repositorio GitHub o archivo adjunto

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)

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

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

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)

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)

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

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"
}
```