

Infraestructura para la Computación de Altas Prestaciones

Proyecto Final: AquaSenseCloud

Francisco Javier Mercader Martínez

Javier Moreno Pagán

Pablo Meseguer Domenech

Curso: 2024/2025

Índice

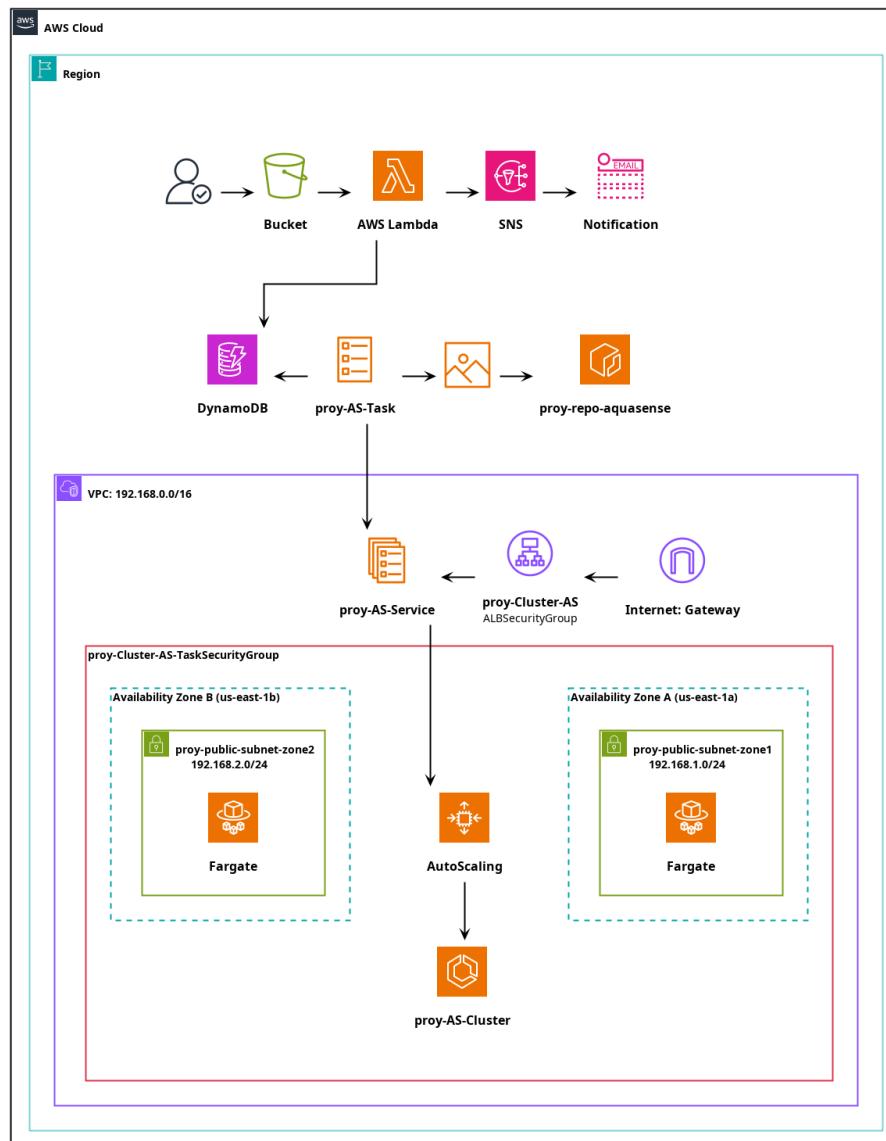
1 Cronograma	2
2 Diagrama Arquitectura de la solución	3
3 Listado de Recursos y Servicios con su funcionalidad	3
4 Ejemplos de demostración de correcta funcionalidad	7

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 alguna 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 compatibilidad 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 disponibilidad 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 períodos de baja demanda, manteniendo solo las tareas necesarias para soportar la carga en ese momento. Similar a la política de escalado ascendente, aplica un enfriamiento de 60 segundos.

10. Seguridad

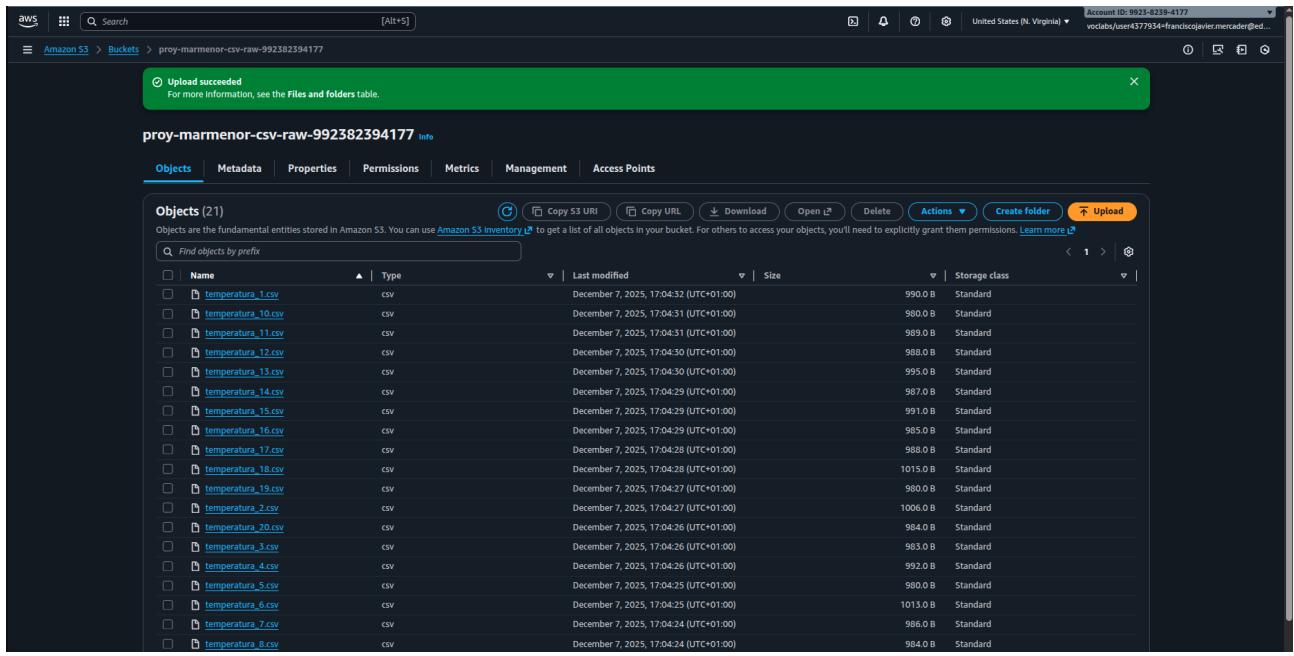
- **ALB Security Group (AWS::EC2::SecurityGroup):** Permite el acceso al balanceador de carga desde Internet (a través del puerto 80-HTTP), asegurando que el tráfico pueda alcanzar la aplicación.
- **Task Security Group (AWS::EC2::SecurityGroup):** Permite el tráfico del ALB a las tareas de ECS (a través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

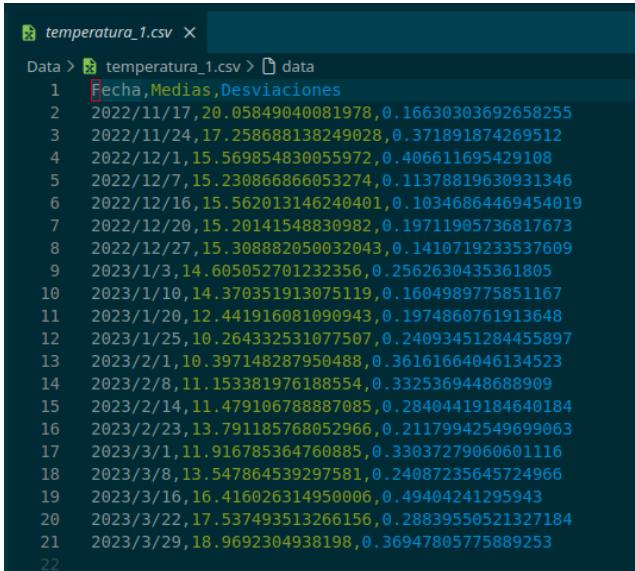
4 Ejemplos de demostración de correcta funcionalidad

- Función Lambda:



The screenshot shows the AWS S3 console with a green success message: "Upload succeeded". Below it, a table lists 21 objects (CSV files) in the bucket "proy-marmenor-csv-raw-992382394177". The columns include Name, Type, Last modified, Size, and Storage class. All files are CSV and have a size of 990.0 B, stored in Standard storage.

Name	Type	Last modified	Size	Storage class
temperatura_1.csv	csv	December 7, 2025, 17:04:32 (UTC+01:00)	990.0 B	Standard
temperatura_10.csv	csv	December 7, 2025, 17:04:31 (UTC+01:00)	980.0 B	Standard
temperatura_11.csv	csv	December 7, 2025, 17:04:31 (UTC+01:00)	989.0 B	Standard
temperatura_12.csv	csv	December 7, 2025, 17:04:30 (UTC+01:00)	988.0 B	Standard
temperatura_13.csv	csv	December 7, 2025, 17:04:30 (UTC+01:00)	995.0 B	Standard
temperatura_14.csv	csv	December 7, 2025, 17:04:29 (UTC+01:00)	987.0 B	Standard
temperatura_15.csv	csv	December 7, 2025, 17:04:29 (UTC+01:00)	991.0 B	Standard
temperatura_16.csv	csv	December 7, 2025, 17:04:29 (UTC+01:00)	985.0 B	Standard
temperatura_17.csv	csv	December 7, 2025, 17:04:28 (UTC+01:00)	988.0 B	Standard
temperatura_18.csv	csv	December 7, 2025, 17:04:28 (UTC+01:00)	1015.0 B	Standard
temperatura_19.csv	csv	December 7, 2025, 17:04:27 (UTC+01:00)	980.0 B	Standard
temperatura_2.csv	csv	December 7, 2025, 17:04:27 (UTC+01:00)	1006.0 B	Standard
temperatura_20.csv	csv	December 7, 2025, 17:04:26 (UTC+01:00)	984.0 B	Standard
temperatura_3.csv	csv	December 7, 2025, 17:04:26 (UTC+01:00)	983.0 B	Standard
temperatura_4.csv	csv	December 7, 2025, 17:04:26 (UTC+01:00)	992.0 B	Standard
temperatura_5.csv	csv	December 7, 2025, 17:04:25 (UTC+01:00)	980.0 B	Standard
temperatura_6.csv	csv	December 7, 2025, 17:04:25 (UTC+01:00)	1013.0 B	Standard
temperatura_7.csv	csv	December 7, 2025, 17:04:24 (UTC+01:00)	986.0 B	Standard
temperatura_8.csv	csv	December 7, 2025, 17:04:24 (UTC+01:00)	984.0 B	Standard



The screenshot shows the contents of the file "temperatura_1.csv". The first few rows are as follows:

Fecha	Medias	Desviaciones
2022/11/17, 20.05849040081978	0.16630303692658255	
2022/11/24, 17.258688138249028	0.371891874269512	
2022/12/1, 15.569854830055972	0.406611695429108	
2022/12/7, 15.230866866053274	0.11378819630931346	
2022/12/16, 15.562013146240401	0.10346864469454019	
2022/12/20, 15.20141548830982	0.19711905736817673	
2022/12/27, 15.308882050032043	0.1410719233537609	
2023/1/3, 14.605052701232356	0.2562630435361805	
2023/1/10, 14.370351913075119	0.1604989775851167	
2023/1/20, 12.441916081090943	0.1974860761913648	
2023/1/25, 10.264332531077507	0.24093451284455897	
2023/2/1, 10.397148287950488	0.36161664046134523	
2023/2/8, 11.153381976188554	0.3325369448688999	
2023/2/14, 11.47910678887085	0.28404419184640184	
2023/2/23, 13.791185768052966	0.21179942549699063	
2023/3/1, 11.916785364760885	0.33037279860601116	
2023/3/8, 13.547864539297581	0.24087235645724966	
2023/3/16, 16.416026314950006	0.49404241295943	
2023/3/22, 17.537493513266156	0.28839550521327184	
2023/3/29, 18.9692304938198	0.36947805775889253	
22		

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

DynamoDB > Explore Items > Measures

Table: Measures - Items returned (15)

Scan started on December 07, 2025, 21:57:31

	monthYear (String)	last_updated	max_diff_temp	max_sd	max_temp	mean_t...	mean_temp_count
1	2020-04	2025-12-07T2...	2.9	0.35	20.01	18.19	5
2	2022-01	2025-12-07T2...	-0.2	0.28	13.8	12.63	4
3	2022-02	2025-12-07T2...	0.7	0.15	14.5	13.38	4
4	2020-05	2025-12-07T2...	5.01	0.87	25.02	23.03	6
5	2021-12	2025-12-07T2...	14	0.59	14	15.52	2
6	2022-05	2025-12-07T2...	0.56	0.18	19.31	19.31	1
7	2020-06	2025-12-07T2...	1.77	1.06	26.79	25.27	6
8	2023-01	2025-12-07T2...	-0.96	0.26	14.61	12.92	4
9	2022-11	2025-12-07T2...	20.06	0.37	20.06	18.66	2
10	2023-03	2025-12-07T2...	5.18	0.49	18.97	15.68	5
11	2020-03	2025-12-07T2...	17.11	0.43	17.11	16.62	3
12	2022-04	2025-12-07T2...	3.54	0.44	18.75	16.64	5
13	2022-03	2025-12-07T2...	0.71	0.12	15.21	15	4
14	2022-12	2025-12-07T2...	-4.49	0.41	15.57	15.37	5
15	2023-02	2025-12-07T2...	-0.82	0.36	13.79	11.7	4

Figure 2: Almacenamiento de datos transformados en DynamoDB

Nota: Observar que la función lambda trata con la posibilidad de ingestión 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 ingestión de manera controlada y con ficheros de un tamaño medio (2-3 Kbytes) como los utilizados en las pruebas realizadas.

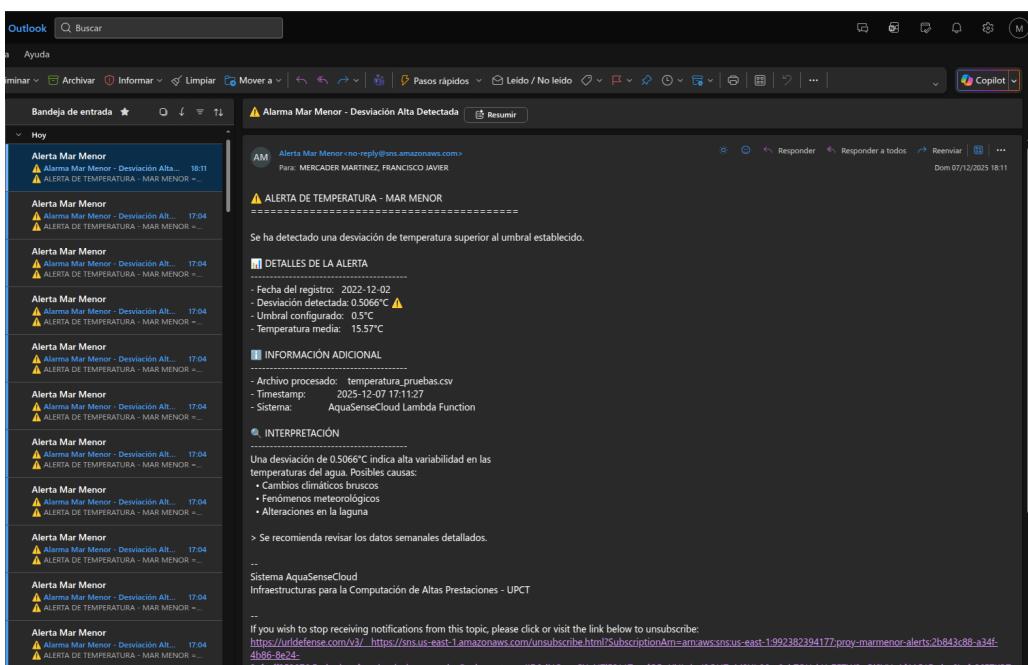


Figure 3: Envío de correo a los analistas mediante un tópico definido mediante AWS SNS

- ECR:

The screenshot shows the AWS ECS console. On the left, the navigation menu includes 'Clusters' (selected), 'Namespaces', 'Task definitions', 'Account settings', 'Amazon ECR', 'Repositories', 'AWS Batch', 'Documentation', 'Discover products', and 'Subscriptions'. The main area displays 'Clusters (1) Info' with a table showing 1 cluster ('proy-marmoner-cluster') with 3 EC2 instances, 0 pending tasks, and 3 running tasks. A 'Create cluster' button is at the top right. Below the table is a 'CloudWatch Logs Live Tail' window with a 'Filter' section, a timeline from 00:00:00 to 0 events/sec, and buttons for 'View in columns' and 'View in plain text'.

- Automatización del aprovisionamiento de la infraestructura mediante CloudFormation:

The screenshot shows the AWS CloudFormation console. The left sidebar includes 'Stacks' (selected), 'Drifts', 'Stack refactors', 'StackSets', 'Exports', 'Infrastructure Composer', 'IaC generator', 'Hooks overview', 'Invocation summary', 'Hooks', 'Registry', 'Public extensions', 'Activated extensions', 'Publisher', 'Spotlight', and 'Feedback'. The main area shows 'Stacks (4)' for the stack 'proy-infraestructura-soporte'. The 'Resources (23)' table lists the following resources:

Logical ID	Physical ID	Type	Status	Module
ALBSecurityGroup	sg-0d77babfccc6d9a65	AWS-EC2:SecurityGroup	CREATE_COMPLETE	-
DokerInstance	i-0bedcd7dab1f9df724	AWS-EC2:Instance	CREATE_COMPLETE	-
InstanceSecurityGroup	sg-0893a08c15aa54f2	AWS-EC2:SecurityGroup	CREATE_COMPLETE	-
InternetGateway	igw-05e506a589aae5bf94	AWS-EC2:InternetGateway	CREATE_COMPLETE	-
myDynamoDBTable	aws-dynamodb:Table	AWS-DynamoDB:Table	CREATE_COMPLETE	-
NatGateway	nat-000d16e89439109	AWS-EC2:NatGateway	CREATE_COMPLETE	-
NatGatewayEIP	3.253.57.113	AWS-EC2:EIP	CREATE_COMPLETE	-
PrivateFlaskSecurityGroup	sg-090f70d5e1d95ea89	AWS-EC2:SecurityGroup	CREATE_COMPLETE	-
PrivateRoute	rtb-035579011a5c985b0 0.0.0/0	AWS-EC2:Route	CREATE_COMPLETE	-
PrivateRouteTable	rtb-035579011a5c985b0	AWS-EC2:RouteTable	CREATE_COMPLETE	-
PrivateSubnet1	subnet-023430cc230a9e15e	AWS-EC2:Subnet	CREATE_COMPLETE	-
PrivateSubnet2	subnet-07318ed95deaa2a7b	AWS-EC2:Subnet	CREATE_COMPLETE	-
PrivateSubnetRouteTableAssociation	rtbassoc-0edb1cf4d5b5fad8	AWS-EC2:SubnetRouteTableAssociation	CREATE_COMPLETE	-
PrivateSubnetRouteTableAssociation?	rtbassoc-0fee34440ac3ed481	AWS-EC2:SubnetRouteTableAssociation	CREATE_COMPLETE	-

CloudFormation > Stacks > proy-lambda-bucket

Stacks (4)

Resources (4)

Logical ID	Physical ID	Type	Status
LambdaInvokePermission	proy-lambda-bucket-LambdaInvokePermission-jQDvO2L7a29r	AWS::Lambda::Permission	CREATE_COMPLETE
MarMenorAlertsTopic	arn:aws:sns:us-east-1:992382394177:proy-marmenor-alerts	AWS::SNS::Topic	CREATE_COMPLETE
ProcessSSFileLambda	proy-marmenor-process-cor	AWS::Lambda::Function	CREATE_COMPLETE
S3Bucket	proy-marmenor-csv-raw-992382394177	AWS::S3::Bucket	CREATE_COMPLETE

CloudFormation > Stacks > proy-infraestructura-cluster

Stacks (4)

Resources (8)

Logical ID	Physical ID	Type	Status
ALBListener	arn:aws:elasticloadbalancing:us-east-1:992382394177:listener/app/proy-marmenor-alb/ed094306d65a43fbfd4206d7845cb	AWS::ElasticLoadBalancingV2::Listener	CREATE_COMPLETE
ApplicationLoadBalancer	arn:aws:elasticloadbalancing:us-east-1:992382394177:loadbalancer/app/proy-marmenor-alb/ed094306d65a43fbfd4206d7845cb	AWS::ElasticLoadBalancingV2::LoadBalancer	CREATE_COMPLETE
ECSAutoScalingGroup	proy-infraestructura-cluster-ECSAutoScalingGroup-K2zHqQ9bNvH	AWS::AutoScaling::AutoScalingGroup	CREATE_COMPLETE
ECSCluster	proy-marmenor-cluster	AWS::ECS::Cluster	CREATE_COMPLETE
ECSLaunchConfiguration	proy-infraestructura-cluster-ECSLaunchConfiguration-CCorfSuocdo	AWS::AutoScaling::LaunchConfiguration	CREATE_COMPLETE
ECSServiceEC2	arn:aws:ecs:us-east-1:992382394177:service/proy-marmenor-cluster/proy-infraestructura-cluster-ECSERVICEC2-4K9ScNgj118	AWS::ECS::Service	CREATE_COMPLETE

- **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/

The screenshot shows the Postman interface with a successful API call to `/health`. The response body is:

```
{  
  "status": "healthy",  
  "tabla": "Measures"  
}
```

maxdiff/

The screenshot shows the Postman interface with a successful API call to `/maxdiff` with parameters `month=4` and `year=2020`. The response body is:

```
{  
  "last_updated": "2025-12-07T20:46:15.014505",  
  "max_temp": 26.01,  
  "maxdiff": 2.9,  
  "month": 4,  
  "year": 2020  
}
```

months/

The screenshot shows the AquasenseUPCT API interface in a browser. The top navigation bar includes links for 'GET API Access', 'GET health', 'GET maxdiff', 'GET months' (highlighted in yellow), 'GET sd', 'GET temp', and a '+' button. On the far right, there are 'Save', 'Share', and 'No environment' buttons.

The main content area displays two separate API requests:

GET /months

Request URL: <http://proxy-marmenor-alb-836308048.us-east-1.elb.amazonaws.com/months>

Method: GET

Headers: Authorization, Headers (7), Body, Scripts, Settings

Query Params:

Key	Value	Description	Bulk Edit
Key	Value	Description	...

Body, Cookies, Headers (5), Test Results, Preview, Visualize, JSON.

Response status: 200 OK, 520 ms, 325 B, Save Response.

Results (14 items):

- 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 View, Find and replace, Console, Postbot, Runner, Capture requests, Cookies, Vault, Trash.

GET /sd/

Request URL: <http://proxy-marmenor-alb-836308048.us-east-1.elb.amazonaws.com/sd?month=3&year=2020>

Method: GET

Headers: Authorization, Headers (7), Body, Scripts, Settings

Query Params:

Key	Value	Description	Bulk Edit
month	3		...
year	2020		
Key	Value	Description	

Body, Cookies, Headers (5), Test Results, Preview, Visualize, JSON.

Response status: 200 OK, 237 ms, 228 B, Save Response.

Results:

last_updated	2025-12-07T20:46:14.985461
month	3
sd	0.43
year	2020

Find View, Find and replace, Console, Postbot, Runner, Capture requests, Cookies, Vault, Trash.

temp/

The screenshot shows the Postman interface for an API endpoint named 'temp/'. The URL is set to `http://proxy-marmenor-alb-838308048.us-east-1.elb.amazonaws.com/temp?month=3&year=2020`. The 'Params' tab is selected, showing query parameters: 'Key' (checked), 'month' (value 3), and 'year' (value 2020). The 'Body' tab shows the JSON response:

```
[{"last_updated": "2025-12-07T20:46:14.985461", "max_temp": 17.11, "month": 3, "temp": 16.62, "year": 2020}]
```

The response status is 200 OK, with a duration of 274 ms and a size of 248 B. The response body is displayed in JSON format.