

Demostración de funcionamiento

A continuación, se muestra una recopilación de evidencias visuales que demuestran el despliegue, configuración y correcta ejecución de la arquitectura desarrollada. Cada figura ilustra una parte clave del sistema, desde la recepción del evento hasta la generación del reporte diario.

Figura 1. Cola SQS creada

Cola dispositivos-detectados creada correctamente en Amazon SQS. Se muestra su URL, región y configuración activa.

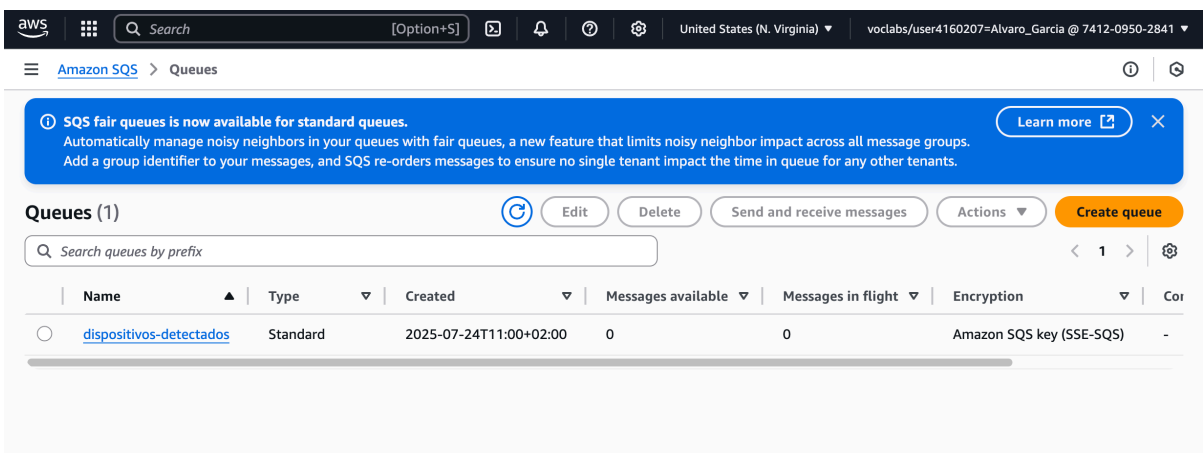


Figura 2. Trigger en Lambda procesar_evento

La función procesar_evento está conectada a la cola SQS como trigger, con estado habilitado. Esto permite que se dispare automáticamente al recibir eventos.

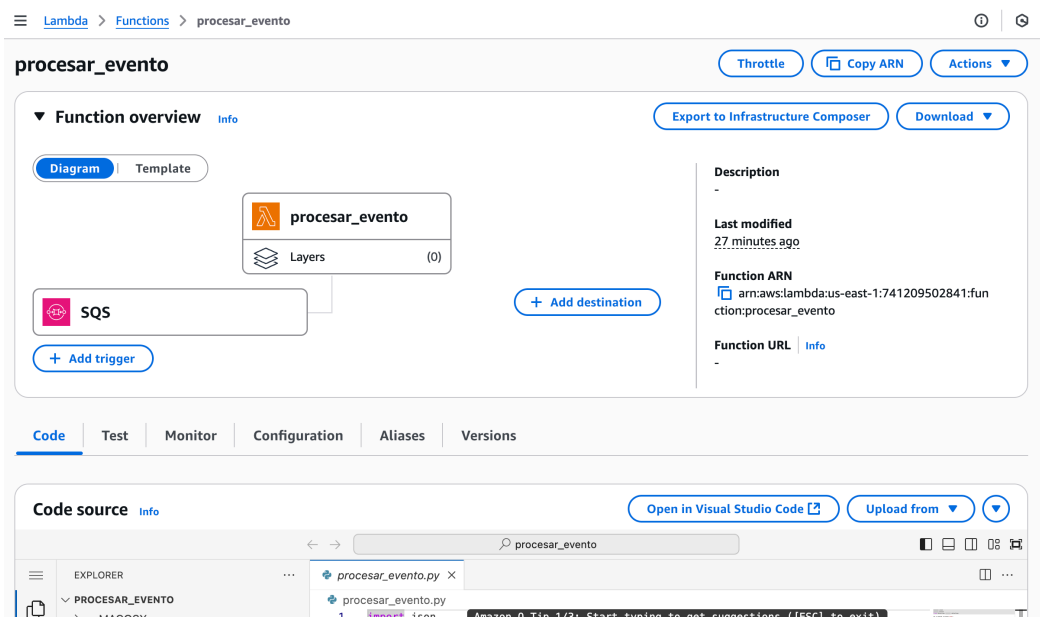


Figura 3. Código desplegado y handler configurado

Se muestra la configuración del handler como procesar_evento.lambda_handler y la subida del código empaquetado en ZIP.

Runtime settings [Info](#)

Runtime
Python 3.11

Handler [Info](#)
procesar_evento.lambda_handler

Architecture [Info](#)
x86_64

[Edit](#) [Edit runtime management configuration](#)

[▶ Runtime management configuration](#)

Code source [Info](#)

[Open in Visual Studio Code](#) [Upload from](#) [▼](#)

EXPLORER

PROCESAR_EVENTO
 __MACOSX
 procesar_evento.py

DEPLOY
 Deploy (⌘U)
 Test (⌘I)

TEST EVENTS [NONE SELECTED]
 + Create new test event

procesar_evento.py

```
1 import json
2 import boto3
3 from datetime import datetime
4
5 s3 = boto3.client('s3')
6 dynamodb = boto3.resource('dynamodb')
7 table = dynamodb.Table('presencia_actual')
8
9 S3_BUCKET = 'agh-tracking-2025'
10
11 def lambda_handler(event, context):
12     for record in event['Records']:
13         body = json.loads(record['body'])
14
15         device_mac = body['device_mac']
16         zone = body['location']['zone']
17         building = body['location'].get('building', 'unknown')
18         timestamp = body['detected_at']
19         vendor = body.get('vendor', 'unknown')
20         rssi = body.get('rssi', -99)
21
22         # 1. Guardar en S3
23         s3_key = f"raw/{building}/{zone}/{device_mac}/{timestamp}.json"
24         s3.put_object(
25             Bucket=S3_BUCKET,
26             Key=s3_key,
27             Body=json.dumps(body)
28         )
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
```

Figura 4. Registro de ejecución en CloudWatch Logs

Log generado en CloudWatch tras la ejecución de la función `procesar_evento`. Se confirma que el evento fue recibido correctamente desde SQS y que se accedió tanto a S3 como a DynamoDB para procesarlo.

CloudWatch

Favorites and recents

Dashboards

AI Operations

Alarms

Logs

Log groups

Log Anomalies

Metrics

All metrics

Explorer

Streams

Log groups (5)

By default, we only load up to 10000 log groups.

Filter log groups or try pattern search

Exact match

Log group

Log class

Anomaly d...

D...

S...

Re

<input type="checkbox"/>	/aws/lambda/RedshiftEventSubscription	Standard	Configure	-	-	Ni
<input type="checkbox"/>	/aws/lambda/RedshiftOverwatch	Standard	Configure	-	-	Ni
<input type="checkbox"/>	/aws/lambda/RoleCreationFunction	Standard	Configure	-	-	Ni
<input type="checkbox"/>	/aws/lambda/generar_reporte	Standard	Configure	-	-	Ni
<input type="checkbox"/>	/aws/lambda/procesar_evento	Standard	Configure	-	-	Ni

Log class

Info

Standard

ARN

[arn:aws:logs:us-east-1:741209502841:log-group:/aws/lambda/procesar_evento:*](#)

Creation time

36 minutes ago

Retention

Never expire

Stored bytes

-

Metric filters

0

Subscription filters

0

Contributor Insights rules

-

KMS key ID

-

Anomaly detection

[Configure](#)

Data protection

-

Sensitive data count

-

Field indexes

[Configure](#)

Transformer

[Configure](#)

Log streams

Tags

Anomaly detection

Metric filters

Subscription filters

Contributor Insights

Log streams (2)

Filter log streams or try prefix search

Exact match

Show expired

Info

Log stream

Last event time

<input type="checkbox"/>	2025/07/23/[\$LATEST]5f14a82ed026431c9a406cba1ff1f44d	2025-07-24 09:53:34 (UTC)
<input type="checkbox"/>	2025/07/24/[\$LATEST]0330358791314eb8bc4de9dade7ce4f	2025-07-24 09:52:04 (UTC)

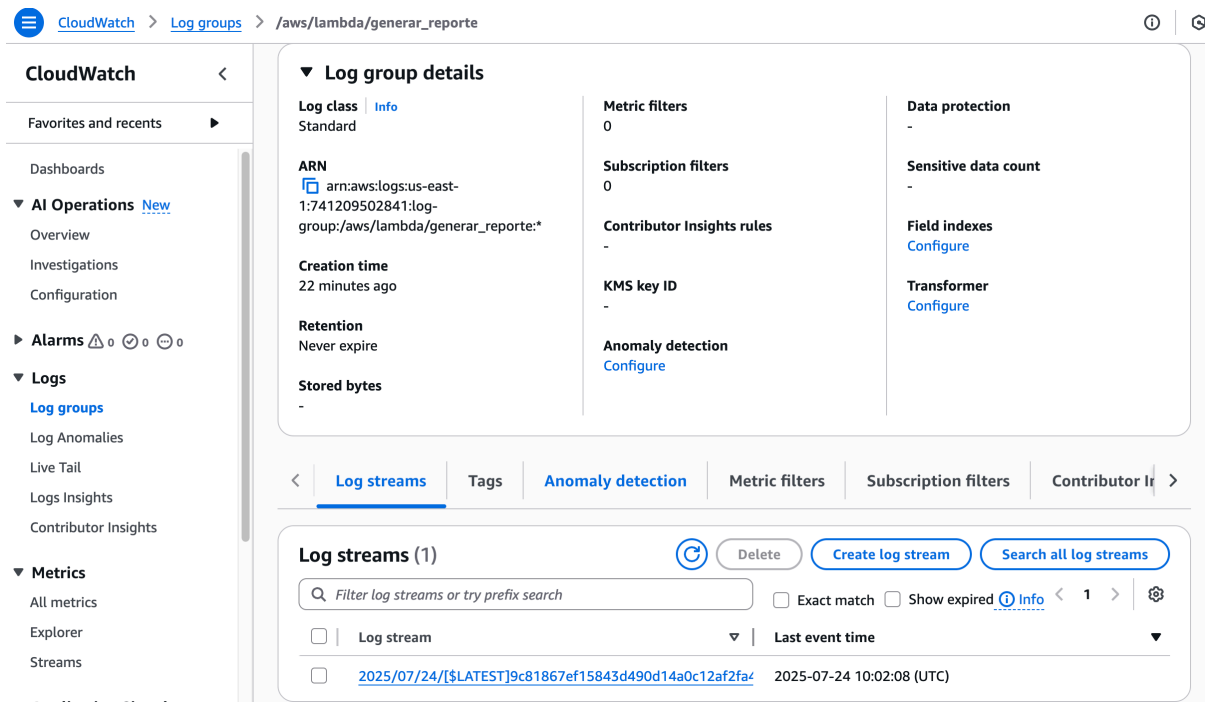


Figura 5. Archivo JSON almacenado en S3

Objeto almacenado en el bucket agh-tracking-2025 bajo la ruta /raw/Sede Central/Recepción/.... Contiene el evento completo recibido vía SQS.

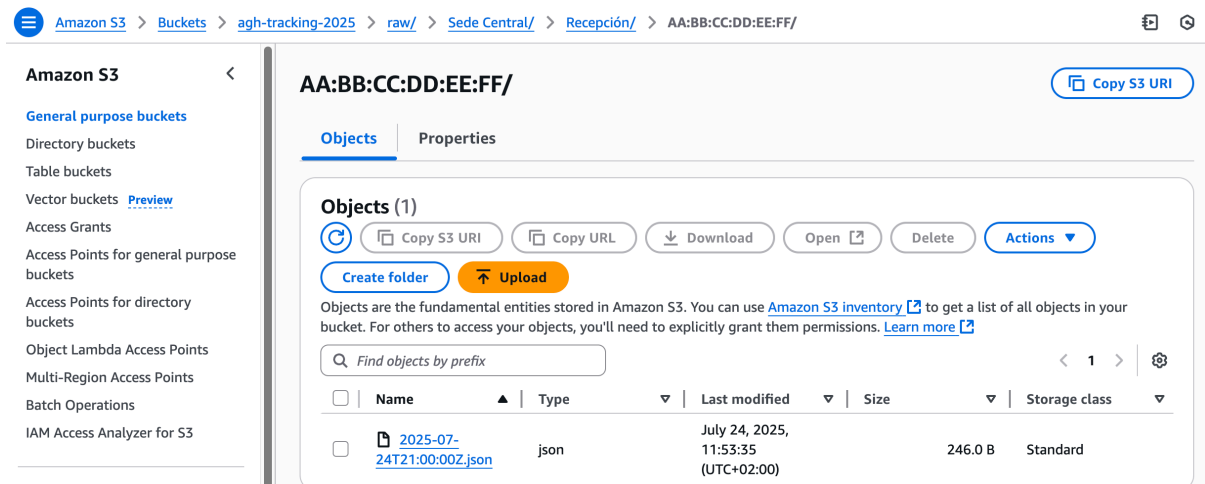


Figura 6. Entrada en DynamoDB

Entrada creada o actualizada en la tabla presencia_actual. Incluye device_mac, zone, check_in, last_seen y otros campos.

DynamoDB

Explore items

presencia_actual

1

Tables (1)

Any tag key

Any tag value

Find tables

<

1

>

presencia_actual

presencia_actual

Autopreview

View table details

▼ Scan or query items

Scan

Query

Select a table or index

Table - presencia_actual

Select attribute projection

All attributes

► Filters - optional

Run

Reset

Completed

Items returned: 1

Items scanned: 1

Efficiency: 100%

RCUs consumed: 2

Table: presencia_actual - Items returned (1)

Actions

Create item

Scan started on July 24, 2025, 12:26:44

<

1

>

device_mac (String)

building

check_in

last_seen

rsi

vendor

zone

AA:BB:CC:DD:EE:FF

Sede Central

2025-07-2...

2025-07-2...

-60

Apple

Recepción

Figura 7. Lambda generar_reporte ejecutada

Prueba manual de la función generar_reporte con éxito. Se observa la programación de la función y la configuración del handler (7.1), así como la salida del log y la confirmación del número de registros procesados. (7,2)

The screenshot displays the AWS Lambda console for the function 'generar_reporte'. The left sidebar shows the function's configuration, including the handler 'generar_reporte.py' and deployment options. The main area shows the Python code for the lambda handler, which interacts with DynamoDB to scan a table named 'presencia_actual' and process the results. The code includes a loop to iterate over items and calculate the duration between 'last_seen' and 'check_in' timestamps.

```
5 dynamodb = boto3.resource('dynamodb')
6 s3 = boto3.client('s3')
7
8 TABLE_NAME = 'presencia_actual'
9 BUCKET_NAME = 'agh-tracking-2025'
10
11
12 def lambda_handler(event, context):
13     table = dynamodb.Table(TABLE_NAME)
14     response = table.scan()
15
16     items = response.get('Items', [])
17     resumen = []
18
19     for item in items:
20         try:
21             device_mac = item['device_mac']
22             zone = item['zone']
23             check_in = datetime.fromisoformat(item['check_in'])
24             last_seen = datetime.fromisoformat(item['last_seen'])
25             duration = (last_seen - check_in).total_seconds() / 60 # minutos
26
27             resumen.append({
28                 'device_mac': device_mac,
29                 'zone': zone,
30                 'duration_minutes': round(duration, 2),
31                 'vendor': item.get('vendor', ''),
32                 'last_seen': item['last_seen']
33             })
34         except Exception as e:
35             print(f"Error procesando item {item}: {e}")
36
```

Below the code, the 'Code properties' section shows the package size (1.2 kB), SHA256 hash, and last modified time (28 minutes ago). The 'Runtime settings' section shows the runtime (Python 3.11), handler (generar_reporte.lambda_handler), and architecture (x86_64).

The screenshot displays the AWS CloudWatch console for the function 'generar_reporte'. The left sidebar shows the function's configuration, including the handler 'generar_reporte.py' and deployment options. The main area shows the log events for the function, including the timestamp, message, and request ID. The logs show the function's execution, including the initialization of the DynamoDB client, the scanning of the table, and the calculation of the duration between 'last_seen' and 'check_in' timestamps.

Log events:

Timestamp	Message
2025-07-24T10:02:07.838Z	INIT_START Runtime Version: python:3.11.v83 Runtime Version ARN: arn:aws:lambda:us-east-1::runtime:26afe95b80f712a3837463ff3166f54bef5a...
2025-07-24T10:02:08.425Z	START RequestId: 498b13ac-199e-4741-9c4c-35c98f77daf8 Version: \$LATEST
2025-07-24T10:02:08.880Z	Reporte generado con 1 registros y guardado en reports/2025-07-24.json
2025-07-24T10:02:08.880Z	END RequestId: 498b13ac-199e-4741-9c4c-35c98f77daf8
2025-07-24T10:02:08.880Z	REPORT RequestId: 498b13ac-199e-4741-9c4c-35c98f77daf8 Duration: 454.44 ms Billed Duration: 455 ms Memory Size: 128 MB Max Memory Used:...

Figura 8. Reporte diario generado en S3

Archivo JSON almacenado en la carpeta /reports/ del bucket, con nombre de fecha (2025-07-25.json). Contiene el resumen de permanencias por dispositivo y zona.

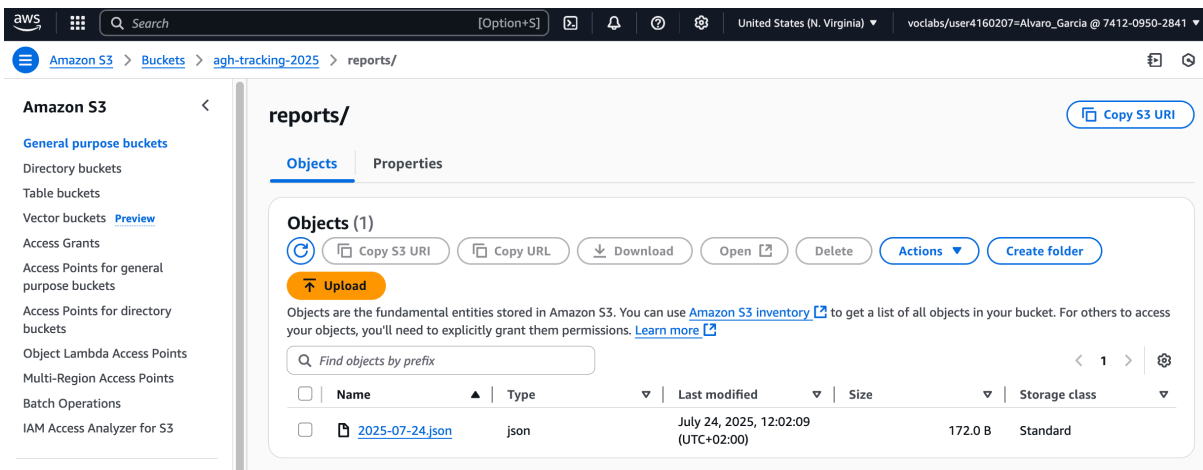
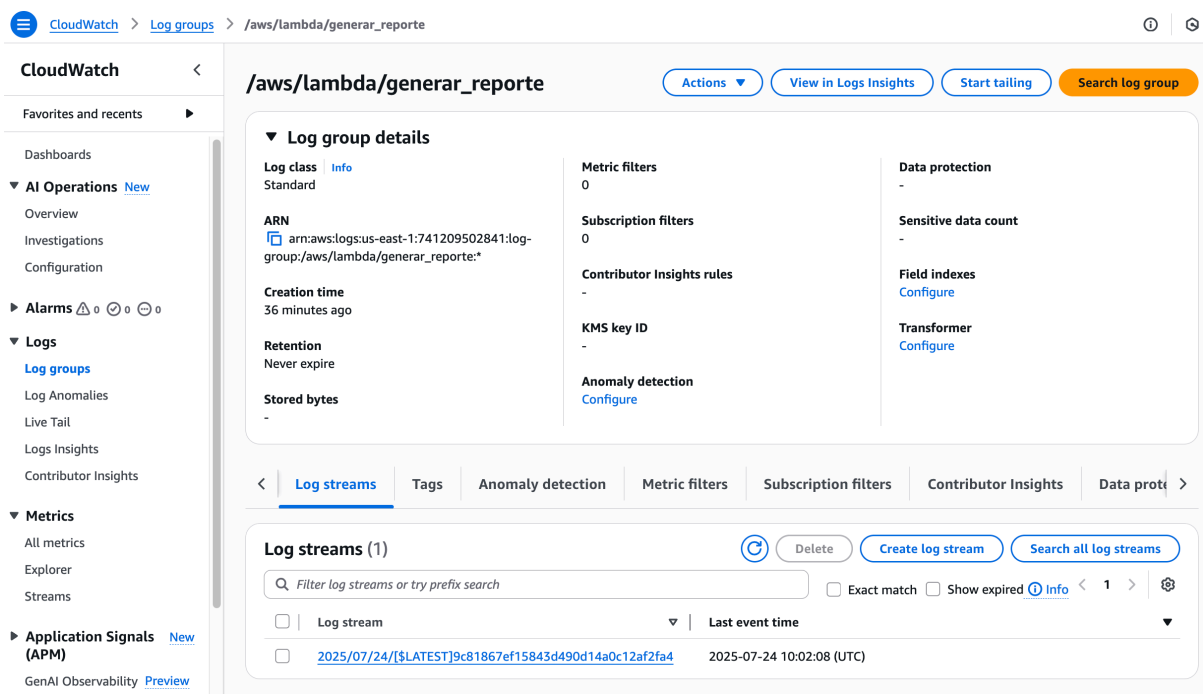


Figura 9. Regla de EventBridge programada

Regla configurada para invocar la Lambda generar_reporte cada día a las 00:00h UTC mediante una expresión cron 0 0 * * * *.



Conclusión

La arquitectura diseñada ha sido desplegada y validada con éxito. Se ha demostrado su correcto funcionamiento mediante pruebas manuales, con evidencia de procesamiento de eventos, almacenamiento, actualización en base de datos y generación automatizada de reportes diarios en S3.