Proyecto

AquaSenseCloud

Infraestructura para la
Computación de Altas Prestaciones
Curso: 2024/2025

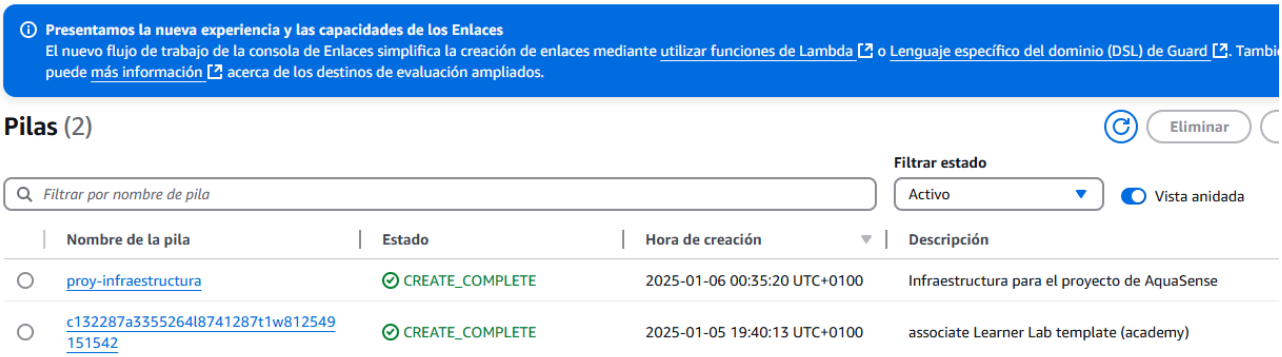
Víctor López Martínez
Javier García Fernández
Miguel Ángel Véliz Ayala

ÍNDICE

Introducción a la infraestructura serverless

El objetivo principal que nos hemos propuesto ha sido proporcionar una solución de infraestructura que nos permita enfocarnos en la lógica de negocio del proyecto. De este modo, hemos tratado de explotar las ventajas de emplear servicios serverless, que ofrecen una escalabilidad óptima, una eficiente gestión de recursos y una clara separación de responsabilidades en microservicios. En definitiva, todo ello garantiza que nuestra solución AWS sea sólida y confiable.

Para desplegar la infraestructura básica hemos hecho uso del servicio Cloudformation. En el panel de la consola de AWS hemos clicado en **Crear pila**, hemos cargado la plantilla *proy-infraestructura*, la hemos nombrado *proy-infraestructura* y en el rol de IAM le hemos asignado LabRole (como hemos hecho a lo largo del curso).



Con ello, hemos podido crear la mayor parte de los recursos que forman la infraestructura (el bucket donde cargamos los ficheros, las 3 funciones lambda, la tabla de DynamoDB, el tópico y la subscripción de SNS).

Servicio web

Funcionamiento del servicio web

Nuestro servicio web está formado por tres componentes principales: una API REST, una función AWS Lambda y una base de datos DynamoDB.

1. **API REST**: La API actúa como la puerta de entrada al servicio, recibiendo solicitudes HTTP y gestionando las rutas y parámetros proporcionados por los clientes. Es responsable de desencadenar la ejecución de la función Lambda correspondiente.
2. **Función Lambda**: La función Lambda se activa a partir de los eventos generados por la API REST. Su propósito es procesar las solicitudes, extrayendo la información relevante de la URI y sus parámetros. Con base en estos datos, la función realiza consultas a DynamoDB en busca de la información requerida.
3. **Base de datos DynamoDB**: DynamoDB almacena los datos que serán consultados. La función Lambda interactúa con esta base de datos para ejecutar las operaciones de lectura necesarias para satisfacer las solicitudes del cliente.

Finalmente, la función Lambda procesa el resultado de la consulta y lo devuelve en formato JSON, encapsulado como una respuesta HTTP adecuada para el cliente. Este diseño garantiza un servicio escalable, flexible y de bajo mantenimiento, aprovechando las características serverless de AWS.

Justificación del uso de una base de datos DynamoDB

Hemos elegido DynamoDB como base de datos idónea para almacenar los registros que requiere nuestro servicio web, principalmente por su escalabilidad y eficiencia.

Por un lado, DynamoDB, como base de datos NoSQL del tipo clave-valor, facilita la estructuración de los datos en función de nuestras necesidades. Hemos diseñado nuestra tabla para utilizar claves de partición representativas de los registros mensuales, mientras que las claves de ordenación se emplean para gestionar los distintos estadísticos que se exponen a través de las URI's de nuestra API REST. Este diseño nos permite organizar los datos de forma lógica y acceder a ellos de manera eficiente mediante consultas optimizadas.

Por otro lado, DynamoDB garantiza una escalabilidad automática y sin interrupciones, lo que resulta elemental en caso de que las necesidades de nuestra arquitectura evolucionen. Es posible que en un futuro queramos añadir nuevos campos o aumentar significativamente el volumen de datos. La arquitectura distribuida de DynamoDB permite manejar esta carga sin degradación del rendimiento. Además, su capacidad para gestionar operaciones a gran escala y altos volúmenes de tráfico se alinea con las exigencias de un servicio web dinámico y orientado a la nube como el nuestro.

Además, DynamoDB es una base de datos stateful, con lo que satisface la condición de garantizar la persistencia de los datos. Los datos van a permanecer almacenados después de realizar cualquier operación de escritura, lectura, eliminación o actualización.

Por último, DynamoDB destaca por su baja latencia y alto rendimiento, características esenciales para garantizar la rapidez en las consultas realizadas a través de nuestras URI's. Esto no solo mejora la experiencia del cliente, sino que también optimiza la eficiencia general del sistema al minimizar los tiempos de respuesta de la API.

Detalles de la forma de la tabla de DynamoDB:

- La clave de partición: Month&Year -> Año y mes (%Y/%m)
- Claves de ordenación: Son 5
    - maxdiff: Diferencias de máxima temperatura asociadas a cada mes.
        - Tipo de dato que almacena: Real
        - URI del servicio web
    - sd: Identifica las máximas desviaciones de temperatura calculadas de forma mensual.
        - Tipo de dato que almacena: Real
        - URI del servicio web
    - temp: Medias de temperatura calculadas de forma mensual.
        - Tipo de dato que almacena: Diccionario {"mean": Real, "days": Entero}
        - URI del servicio web
    - temp_max: Temperatura máxima mensual
        - Tipo de dato que almacena: Real
        - Usado para calcular las diferencias para /maxdiff
    - num_day_set: Conjunto de días (desde el 1 hasta el 31) asociados a los registros que se han procesado para calcular los estadísticos anteriores.
        - Tipo de dato que almacena: Conjunto
        - Usado para evitar procesar el registro de un día ya procesado.

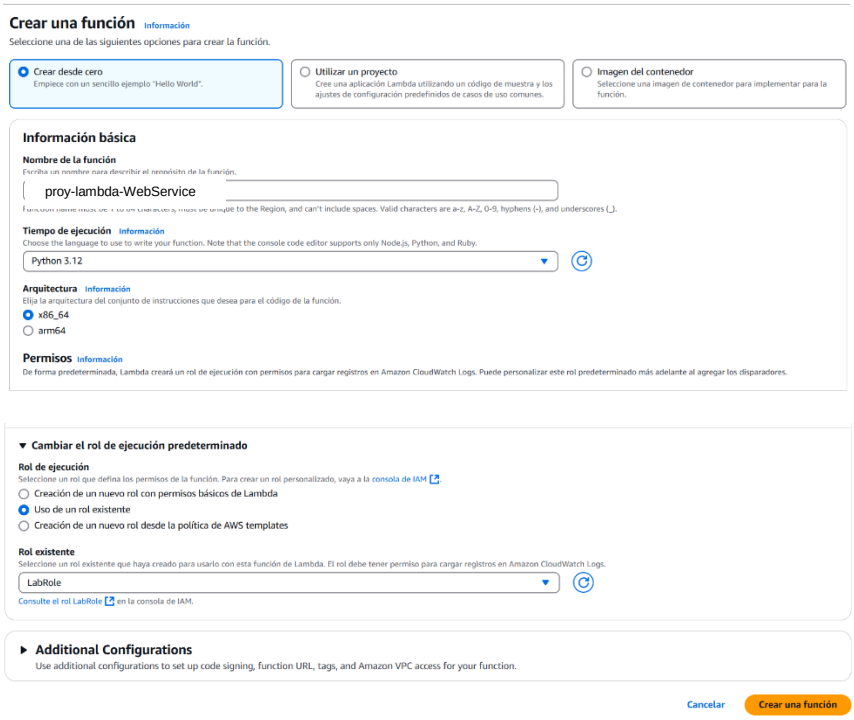
Primary key		Attributes
Partition key: PK	Sort key: SK	
Month&Year (p. ej: *2017/03*)	maxdiff	data 0.3231
	sd	data 0.6723
	temp	data { "mean" : 14.4672, "days": 8}
	temp_max	data 18.7321
	num_day_set	data {22, 7, 8, 13, 4, 29, 15, 21}
Month&Year (p. ej: *2017/04*)	maxdiff	data 0.4721
	sd	data 0.5813
	temp	data { "mean" : 15.1562, "days": 4}
	temp_max	data 19.7451
	num_day_set	data {12, 17, 1, 26}

Elaboración de la infraestructura

1. Creación de una función lambda que haga la consulta a la base de datos DynamoDB

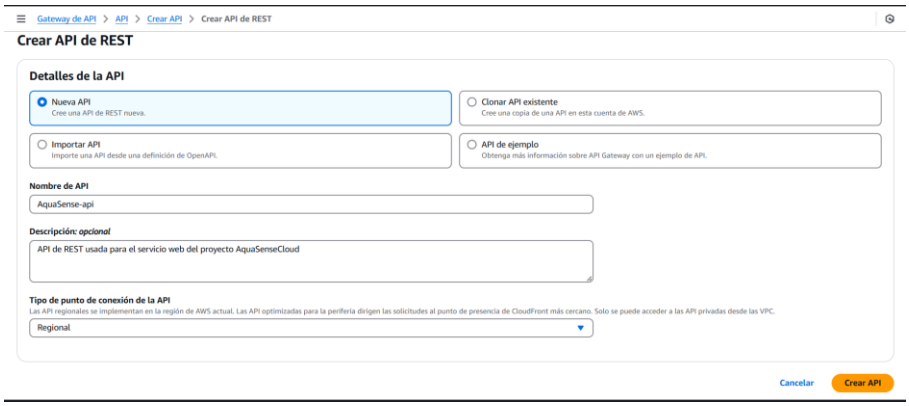
La plantilla de CloudFormation ya crea la función Lambda. Estos pasos son para crearla desde la consola de AWS.

1. En el buscador de servicios de la consola de administración de AWS, seleccionamos *Lambda*.
2. Después, creamos una función Lambda con el nombre proy-lambda-WebService, en tiempo de ejecución seleccionamos Python 3.12 y en rol de ejecución usamos el rol existente *LabRole*.
3. Copiamos el script dado en *Código fuente* y presionamos el botón *Deploy*.



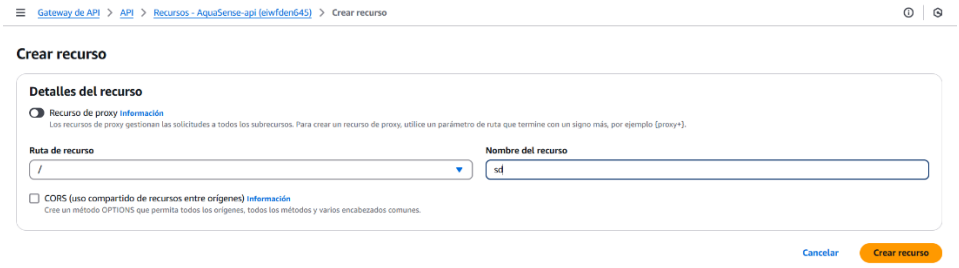
2. Crear una API

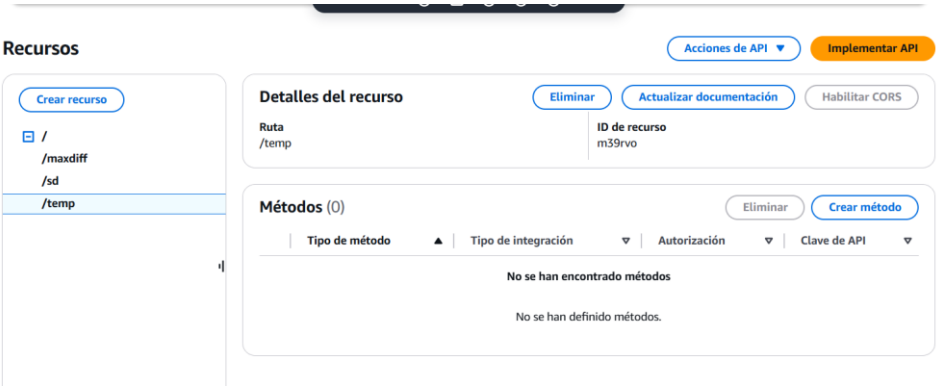
1. En el buscador de servicios de la consola de administración de AWS, seleccionamos API Gateway.
2. Le damos a Crear API y Crear API REST.
3. En detalles de la API, seleccionamos Nueva API. Para el campo nombre escribimos *AquaSense-api* y para descripción: *API de REST usada para el servicio web del proyecto AquaSenseCloud*. Por último, para el tipo de punto de conexión de la API: Regional.



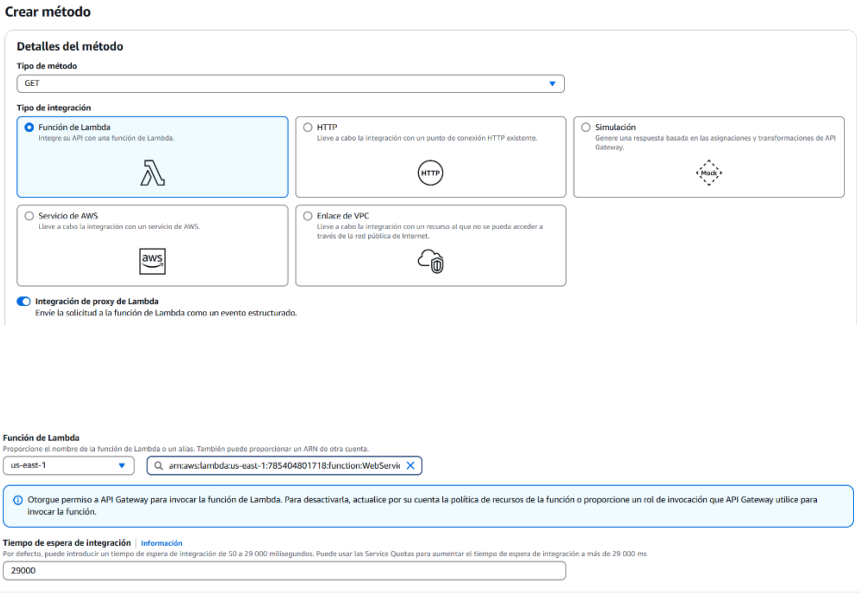
4. Una vez creada la API, creamos varios recursos. Cada recurso tendrá un método asignado correspondiente a cada URI.

5. Creamos primero el recurso con el nombre *maxdiff*, después *sd* y *temp*. Establecemos como ruta de recurso el raíz para todos los recursos.

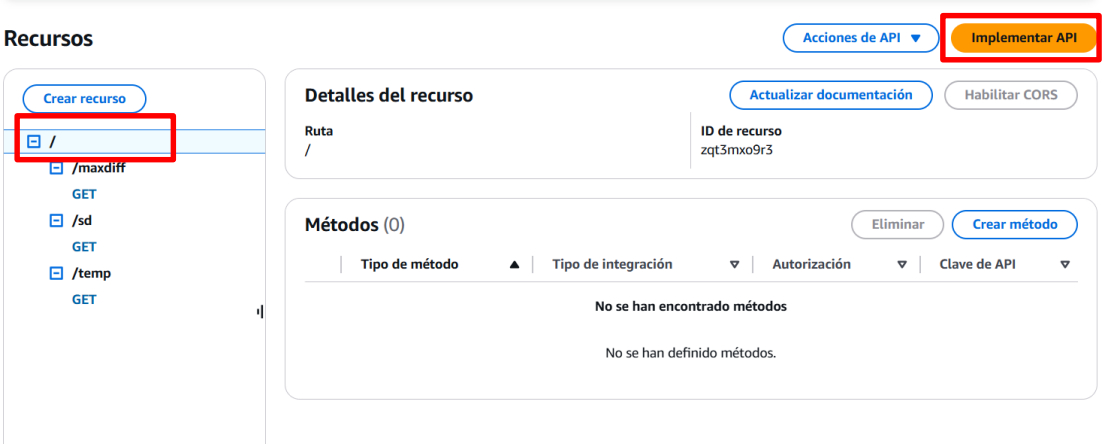




6. Presionamos en crear método. En detalles asignamos el tipo de método GET (porque queremos obtener un dato) y en tipo de integración seleccionamos *Función de Lambda*. Activamos la integración de proxy de Lambda para recibir el evento de forma estructurada. Más abajo, en **Función de Lambda** seleccionamos us-east-1 y el arn de la función lambda creada llamada ***WebService***.

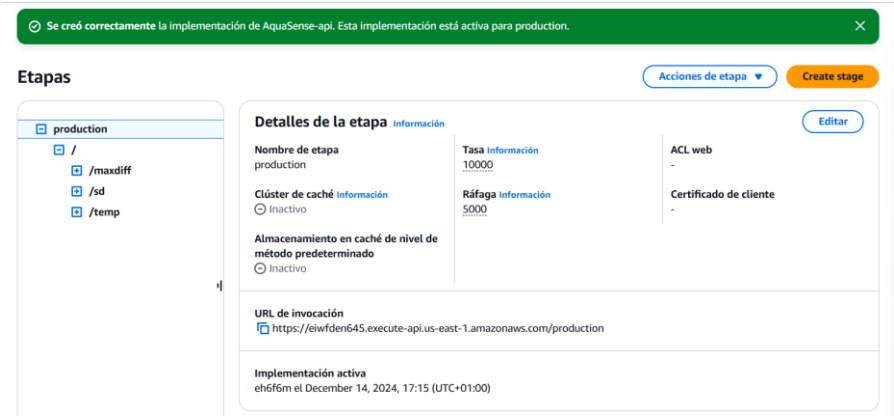


7. Creamos este método con esta misma configuración para los 3 recursos.

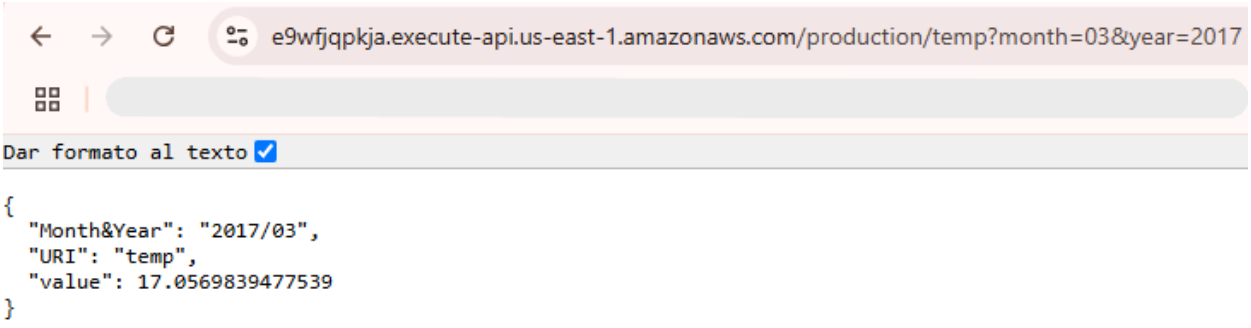


8. Ahora, seleccionando la ruta que engloba a los 3 recursos creados y presionamos **Implementar API**. En etapa seleccionamos una nueva etapa/stage que llamaremos *production* y presionamos en implementación.

9. En etapas tenemos la URL de invocación.



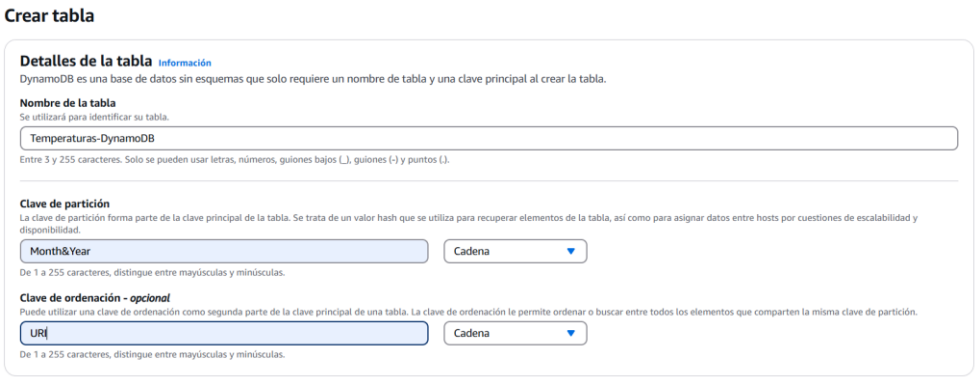
10. Probamos que funciona nuestro servicio web:



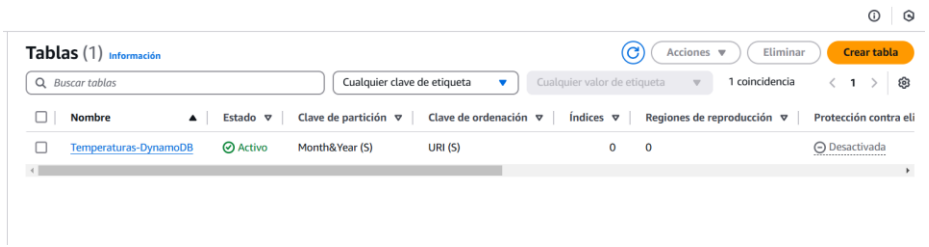
3. Creación de la base de datos DynamoDB

En caso de lanzar la pila con CloudFormation no será necesario seguir estos pasos:

1. En la consola de administración de AWS buscamos DynamoDB.
2. Clicamos en **Crear tabla**.
3. A continuación, en detalles le asignamos los siguientes valores:
   - Nombre de la tabla: *Temperaturas-DynamoDB*
   - Clave de partición: *Month&Year*
   - Clave de ordenación: URI



4. Por último, presionamos en crear tabla.



Servicio SNS

Funcionamiento del servicio SNS

El servicio SNS está formado por tres componentes principales: una función Lambda, un tema SNS, y una suscripción.

1. **Función Lambda**: Esta función Lambda se desencadena a partir de la función Lambda encargada de procesar el fichero. En concreto, será una función interna la que se encargue de, en caso de que alguna desviación típica supere el umbral de 0.5, desencadenar esta función de notificación llamada ***proy-lambda-notification***. Esta función se encargará de generar un mensaje personalizado que le llegará al usuario suscrito al tema con la fecha en la que se superó dicho umbral y con qué valor.

2. **Tema SNS**: El tema, denominado *proy-sns-topic*, actúa como el núcleo de distribución de mensajes dentro del servicio. Es responsable de recibir los mensajes que se publican en él y de gestionar las notificaciones hacia los suscriptores registrados.

3. **Suscripción**: La suscripción define los destinatarios y el protocolo por el cual se recibirán las notificaciones enviadas desde el tema. En este caso, se utiliza el protocolo de suscripción basado en correo electrónico. La dirección de correo electrónico suscrita al tema es ****@gmail.com*.

El servicio SNS asegura la entrega eficiente y segura de los mensajes, proporcionando un mecanismo de comunicación escalable que aprovecha las características serverless de AWS.

Elaboración de la infraestructura

En caso de lanzar la pila de CloudFormation, no será necesario seguir ninguno de los pasos siguientes, salvo entrar al correo y confirmar la suscripción al servicio.

1. Crear un tema para la alerta

1. Accede a *Simple Notification Service* (SNS).
2. Haz clic en *Crear un tema*.
3. Asigna el nombre del tema como: *ThresholdAlert.*
4. Asegúrate de que el tipo de tema esté configurado como "Estándar" para poder enviar correos electrónicos.
5. Desplázate hacia abajo y haz clic en *Crear tema*.

2. Crear una suscripción al tema

1. En la página del tema creado, desplázate hacia el fondo de la página.
2. En el apartado de suscripciones:
    a. Selecciona el protocolo *Correo electrónico*.
    b. Introduce la dirección de correo asociada al proyecto: ***@gmail.com.
3. Importante:
    a. Ingresa al correo proporcionado.
    b. Busca el correo enviado por SNS y confirma la suscripción haciendo clic en el enlace proporcionado.

3. Crear la función Lambda “proy-lambda-notification"

1. Dirígete al servicio AWS Lambda y haz clic en *Crear función*.
2. Configura la función con las siguientes características:
    a. Nombre: *proy-lambda-notification*
    b. Versión de *Python: 3.12*
    c. Rol de ejecución: *LabRole*

4. Implementar el código de la función Lambda

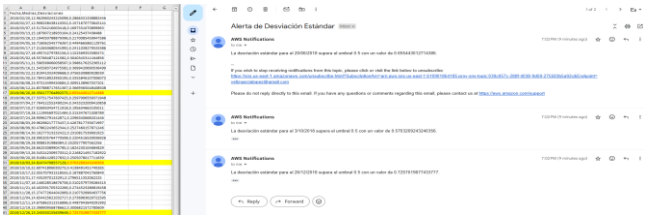
1. Usar el archivo *lambda_function_notification.py* para cargar el código de la función.
2. Asegúrate de que el código esté correctamente vinculado al evento de SNS para el tema *ThresholdAlert*.

5. Configurar el tiempo de espera de la función Lambda

1. En la configuración de la función Lambda, busca el parámetro *Tiempo de espera* (es necesario aumentar el tiempo de espera para que no se repliquen los mensajes).
2. Cambia el valor a 5 minutos.

6. Comprobación

1. Subiéremos un fichero para ver si recibimos el número de mensajes correctos, con el contenido acertado.



Fecha	Medias	Desviaciones
22/03/2017	16.7841	2.8715
30/03/2017	17.3299	4.0372

Cálculo de Estadísticos

Observando el formato de los datos proporcionados, utilizaremos una función Lambda para procesar los datos de temperatura media y desviación estándar de un fichero CSV subido en un bucket de S3 y conseguir los estadísticos necesarios para actualizar o añadir nuevos registros en DynamoDB. Esta función Lambda se dispara cada vez que ocurre un evento de S3, es decir, cuando un archivo CSV se carga en el bucket.

Para la creación de la función Lambda vamos a utilizar el SDK de AWS para Python (*boto3*), para interactuar con S3 y DynamoDB.

El programa **proy-lambda-statistics** que define esta función Lambda está estructurado en varias funciones que permiten modularizar el trabajo, haciéndolo más fácil de leer, mantener y actualizar. Cada función realiza una tarea concreta:

1. **comprobar_umbral(sd, fecha, lambda_client):** Esta función verifica si la desviación estándar supera un umbral definido (0.5). Si es así, invoca una función Lambda adicional para enviar una notificación.
2. **obtener_datos_mes(table, partition_key, uri):** Recupera datos de la tabla de DynamoDB para el mes indicado. Utiliza *partition_key* (que contiene el año y mes) y un identificador (*uri*) para buscar el tipo de dato.
3. **obtener_partition_key(fecha):** Calcula las claves de partición necesarias para acceder a los registros de DynamoDB para el mes actual, anterior y posterior, basándose en la fecha proporcionada.
4. **insert_or_upload_registers(table, partition_key, partition_key_posterior, temp_max, temp_media, dias, sd, num_day_set, maxdiff=None, maxdiff_mes_posterior=None):** Inserta o actualiza registros en DynamoDB para el mes correspondiente, modificando datos del mes posterior si fuera necesario.
5. **calcular_nuevos_valores(items, media, sd, day, temp_max_mes_anterior=None, temp_max_mes_posterior=None):** Calcula los nuevos valores de temperatura máxima, media, días, y desviación estándar, basándose en los registros existentes en DynamoDB y los nuevos valores proporcionados en el archivo CSV.
6. **procesar_fila(row, table, lambda_client):** Procesa una fila de datos del CSV. Para cada fila, calcula y actualiza los valores necesarios en DynamoDB, y si es necesario, invoca la función Lambda para la alerta.

Procesamiento de cada archivo CSV

Cada vez que ocurre un evento:

1. Obtenemos el bucket y la clave del evento

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

2. Leemos el contenido del CSV con el método *get_object* en la variable *csv_file.* Esta variable es un diccionario con metadatos y el contenido del objeto, que se encuentra en el campo *Body* de este diccionario. El contenido del archivo se almacena en binario. Entonces, para poder trabajar con él en Python debemos leerlo y decodificarlo en una cadena de texto.

```
csv_file = s3.get_object(Bucket=bucket, Key=key)
csv_content = csv_file['Body'].read().decode('utf-8')
```

3. Utilizamos la clase *DictReader* del módulo *csv* de Python que lee un archivo CSV y convierte cada fila en un diccionario, siendo las claves del diccionario los nombres de las columnas. Esto se hace para facilitar el acceso a los datos del fichero.
   Necesitamos utilizar la clase del módulo *io* de *StringIO* para tratar la cadena de texto *csv_content* como si fuera un archivo, porque el parámetro que recibe la clase DictReader debe ser un archivo. Usando *StringIO* evitamos crear archivos temporales en disco, lo cual es más rápido y eficiente en términos de rendimiento.

```
csv_reader = csv.DictReader(StringIO(csv_content))
```

4. Una vez tenemos el contenido del archivo, lo procesamos fila por fila utilizando la función ***procesar_fila()***.

Procesamiento de cada fila

1. Obtenemos los valores *fecha, media, sd* y *dia* a partir de la fila del archivo. Los valores de *media* y *sd* se convierten a decimal porque queremos trabajar con valores con precisión exacta.

2. Obtenemos las variables *partition_key , partition_key_anterior* y *partition_key_posterior* con la función ***obtener_partition_keys()***.

3. Comprobamos si el dato actual *sd* es superior a 0.5 con ***comprobar_umbral().***

4. Extraemos la temperatura máxima para los meses anterior y posterior para poder actualizar la base de datos con los nuevos valores obtenidos de la fila del archivo y actualizar los datos del mes posterior si fuera necesario.

5. Comprobamos si tenemos registros para la fecha actual:
   → Si no tenemos registros, calculamos los estadísticos para esa fecha por primera vez.
   → En caso de existir registros, utilizamos la función ***calcular_nuevos_valores()*** *para calcular los estadísticos*.

6. Finalmente actualizamos la base de datos DynamoDB con ***insert_or_upload_registers()***.

¿Cómo se actualizan los estadísticos?

- **maxdiff** : diferencia entre la temperatura máxima actual y la del mes anterior. Si no tenemos registrado ningún valor *maxdiff* del mes anterior, la variable toma el valor *None*. Una vez actualizamos este valor para el mes actual, debemos actualizar también el valor para el mes posterior, si no tenemos registros para el mes posterior, de nuevo, la variable toma el valor *None*.

- **sd** : valor máximo entre el sd actual y el nuevo sd que aporta la fila del fichero.

- **mean** : $\frac{(media_{actual} \times dias + temp_{actual})}{dias+1}$

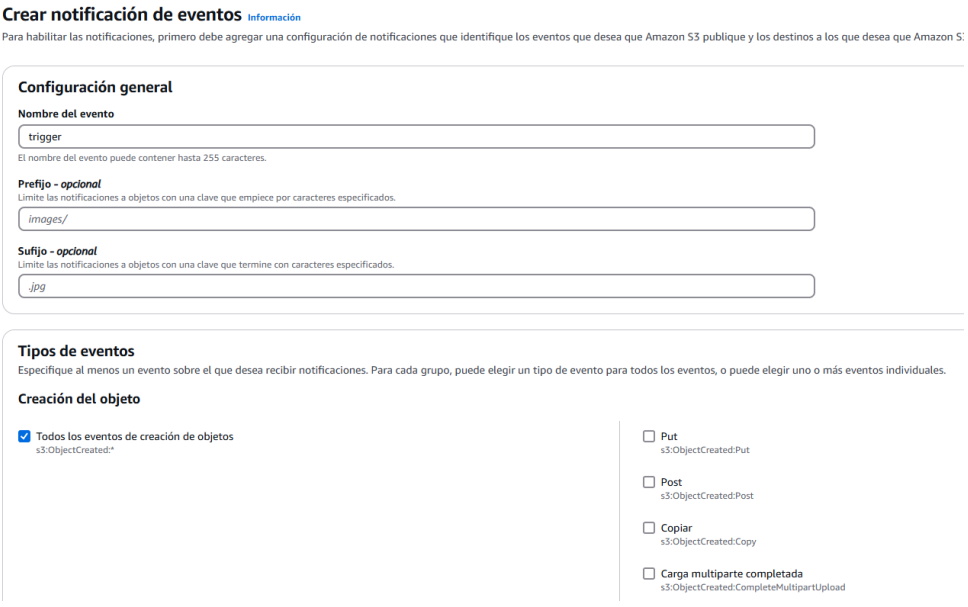
- **day** : sumamos 1 a esta variable. Importante para calcular la media (*mean*).

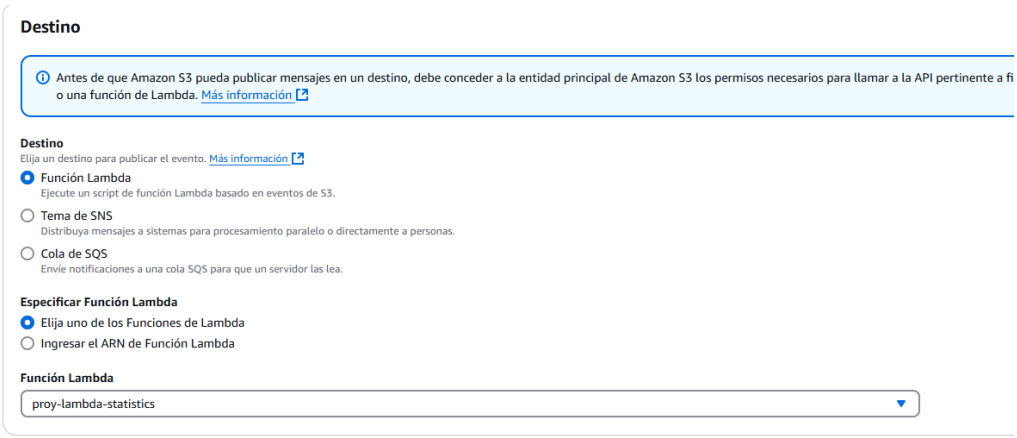
- **temp_max** : valor máximo entre la media de todo el mes registrada hasta el momento y el nuevo valor de temperatura media obtenido.

- **num_day_set** : actualizamos **num_day_set** añadiendo el día (*day)* de la nueva fila que estamos procesando. Este conjunto nos informa de que en el caso de que *day* esté ya en el conjunto, la fila ya habría sido procesada anteriormente y por tanto no se procesa de nuevo.

Creación del evento del bucket S3 que desencadena proy-lambda-statistics

En propiedades del bucket *proy-bucket-files-{ID-accout}* entramos en **Crear notificación de eventos**. Lo nombramos *trigger* y seleccionamos *Todos los eventos de creación de objetos*. En **Destino** seleccionamos la función *proy-lambda-statistics*. Por último, **Guardar cambios**.





PRUEBAS

Dividir y Generar varios ficheros CSV para tests

Vamos a comenzar preparando los ficheros con los que vamos a testear nuestro pipeline de datos. El fichero Temperatura.csv tiene la siguiente estructura:

```
Fecha,Medias,Desviaciones
2017/03/22,16.784072875976562,0.28715428709983826
2017/03/30,17.32989501953125,0.4037204384803772
2017/04/05,18.244800567626953,0.6253794431686401
2017/04/12,19.46531867980957,0.44888031482696533
2017/04/25,18.906261444091797,0.23249997198581696
2017/05/04,19.501352310180664,0.2725655138492584
2017/05/19,23.03464126586914,0.3904358744621277
```

La longitud del fichero original es de 384 registros. Hemos decidido dividirlo en 10 ficheros más pequeños de 40 registros cada uno (a excepción del último fichero).

Hemos implementado la función *particionar_csv*:

```python
import pandas as pd

def particionar_csv(archivo_entrada, filas_por_archivo, prefijo_salida):
    """
    Divide un archivo CSV en varios archivos más pequeños con un número específico de filas por archivo.

    :param archivo_entrada: Ruta al archivo CSV de entrada.
    :param filas_por_archivo: Número de filas que tendrá cada archivo resultante.
    :param prefijo_salida: Prefijo para los nombres de los archivos de salida.
    """
    # Leer el archivo CSV
    df = pd.read_csv(archivo_entrada)

    # Calcular el número total de filas y archivos necesarios
    total_filas = len(df)
    num_archivos = total_filas // filas_por_archivo + (1 if total_filas % filas_por_archivo != 0 else 0)

    for i in range(num_archivos):
        inicio = i * filas_por_archivo
        fin = inicio + filas_por_archivo

        # Dividir el DataFrame
        fragmento = df.iloc[inicio:fin]

        # Guardar el fragmento
        output_file = f"{prefijo_salida}_{i+1}.csv"
        fragmento.to_csv(output_file, index=False)
```

A continuación, solo tendría que ir subiendo los archivos que obtenemos al aplicar esta función al fichero inicial al bucket S3 y comprobar cómo se va actualizando la base de datos de DynamoDB, si nos llegan correctamente las alertas de desviación estándar superior a 0.5 y el funcionamiento de la API.

Resumen de recursos y servicios desplegados

Identificador de recurso	Nombre de recurso	Tipo de recurso en AWS	Comentario (opcional)
arn:aws:execute-api:us-east-1:812549151542:sjm8rm9r12/*/GET/maxdiff	/GET/maxdiff	Método de recurso de API Gateway	
arn:aws:execute-api:us-east-1:812549151542:sjm8rm9r12/*/GET/sd	/GET/sd	Método de recurso de API Gateway	
arn:aws:execute-api:us-east-1:812549151542:sjm8rm9r12/*/GET/temp	/GET/temp	Método de recurso de API Gateway	
arn:aws:lambda:us-east-1:812549151542:function:proy-lambda-WebService	proy-lambda-WebService	Función AWS Lambda	Función Lambda que hace consultas a la base de datos. Desencadenada por los métodos de API Gateway.
arn:aws:dynamodb:us-east-1:812549151542:table/AquaSenseCloudDB	AquaSenseCloudDB	Tabla de DynamoDB	
sjm8rm9r12	AquaSense-api	API	API de REST usada para el servicio web del proyecto AquaSenseCloud
arn:aws:s3:::proy-bucket-files-812549151542	proy-bucket-files-812549151542	Bucket S3	Bucket donde se introducen archivos de datos particionados

arn:aws:lambda:us-east-1:812549151542:function:proy-lambda-statistics	proy-lambda-statistics	Función AWS Lambda	Procesam cada fichero (evento) que llega al bucket S3
arn:aws:lambda:us-east-1:812549151542:function:proy-lambda-notification	proy-lambda-notification	Función AWS Lambda	Desencadenada por proy-lambda-statistics, envía un correo a los usuarios suscritos al tema si se supera cierto umbral
arn:aws:sns:us-east-1:812549151542:proy-sns-topic	proy-sns-topic	Tema de SNS	
12d0c65e-c2bd-48a6-8e7c-13473b2fff50	***@gmail.com	Suscripción	