Informe:
Práctica 5: Ingesta, almacenamiento y Serveless

Jordi Blasco Lozano
Infraestructuras y Servicios Cloud
Universidad de Alicante

17 de noviembre de 2025

Resumen

Índice

1 Flujo IoT: Ingesta y Almacenamiento

1.1 Preparación de Destinos y Configuración IoT

1.1.1 Creación de la Tabla DynamoDB

Se ha creado una tabla en DynamoDB llamada `vigia_farolas_estado` para almacenar el último estado reportado por cada farola junto con métricas del consumo y el momento de medir estas metricas.

1.1.2 Configuración de AWS IoT Core

Utilicé el asistente de AWS IoT Core para registrar un nuevo dispositivo. Durante este proceso, se generaron y descargaron los certificados de seguridad (`farolasestado.cert.pem` y la clave privada correspondiente). Finalmente se asoció una política de seguridad (la cual modifiqué) para que permitiera la conexión y publicación a cualquier tema, yo lo haré en el tema`"smartcity/consumo/test"`. Usamos 'test' porque en ningún momento serán farolas reales por lo que en un supuesto despliegue de la aplicación deberíamos de cambiar el tema.

Para verificar la conectividad, se ejecutó el script de prueba `start.ps1` proporcionado por AWS. El script instaló las dependencias necesarias y envió con éxito cinco mensajes de prueba, que fueron validados tanto en la terminal local como en el cliente de pruebas MQTT de la consola de AWS.

1.1.3 Creación de la Regla IoT

- **Regla SQL** Antes de definir la regla, se creó un bucket en S3 para el almacenamiento histórico de la telemetría. La regla de IoT se configuró para procesar los mensajes que llegan al tema `smartcityconsumotest`. Para la consulta SQL tuve que cambiar un par de instrucciones ya que no me funcionaba correctamente el `processed_time`. Luego veremos lo que use en la función de node-red pero por ahora analizaremos el SQL que usé.

Bloque 1: Consulta SQL para la regla IoT

```
SELECT
    consumption_kw,
    timestamp,
    concat(day, '-', month, '-', year, ' ', hour, ':', minute, ':', seconds) AS
  processed_time
FROM
    'smartcity/consumo/test'
```

Como vemos en el SQL hemos quitado el **home_id** y hemos cambiado el **processed_time**, esto se debe a que AWS no me dejaba utilizar DynamoDBv2 con plantilla y al usar DynamoDB legacy la clave_id la he mapeado directamente en el apartado correspondiente de DynamoDB, de esta forma ya no me apareciera doble en la tabla de dynamo (como clave y como atributo). Posteriormente cambie el proccesed_time porque no hubo forma de procesar el tiempo con la función correspondiente, y esta ha sido la forma más sencilla de pasar el tiempo procesado. Podría haberlo procesado al crear el mensaje o después y he decidido hacerlo después para que las carpetas del s3 pudieran mapear mucho mas facil el tiempo, y así crear las carpetas sin errores.

- **Acción S3 bucket** En esta acción también cambiamos un apartado, la "clave", en el enunciado se pedía utilizar un almacenamiento en carpetas clasificadas por año por mes por dia, el problema que le vi principalmente fue que en un sistema real podriamos tener diferentes farolas por lo que utilice una ultima clasificacion en la carpeta dias que contiene una carpeta por "farola_id". Dentro de esta ultima carpeta ya tenemos cada archivo ordenado por la hora minuto y segundo usando "hh:mm:ss-data.json" en vez timestamp (que a mi se me ponia en milisegundos). He usado esta clave con los parametros obtenidos directamente desde sus variables del mensaje (en vez de desde el timestamp) y los archivos estan ordenados como se puede observar en la imagen.

Bloque 2: Clave del S3

```
farolas/year=${year}/month=${month}/day=${day}/${home_id}/${hour}:${minute}:${seconds}-data.json
```

- **Accion DynamoDB** En esta otra acción se pedía usar DynamoDBv2, despues de probar mil formas cambie a la versión a la de DynamoDB normal en la cual nosotros mismos debemos de vincular la variable clave para usarla como clave de particion de nuestra tabla. Lo hice manualmente porque no sabia como insertar la plantilla de atributos que proporciona el enunciado. Y no conseguí que las claves se vincularan automáticamente, por lo que lo hice manualmente con DynamoDB de la siguiente forma.

1.2 Simulación de Telemetría con Node-RED

Node-RED permite conectarse mediante MQTT a nuestro tema de 'thing' de IoT para simular el envío de datos de las farolas. El flujo consta de tres nodos:

- **Inject:** Un disparador manual para iniciar el flujo, cada 10 segundos se envia una señal al siguiente nodo para que mande un mensaje.
- **Function:** Un nodo que construye el mensaje JSON con los datos de la farola (ID, consumo y el tiempo) y establece el tema de destino en `smartcity/consumo/test`. Este nodo consta de una serie de variables que hemos randomizado para simular la farola. He utilizado 6 variables de más para el tiempo, de forma que tengamos en variables separadas el año, el mes, el dia, la hora, los minutos y los segundos. Consiguiendo que los pasos anteriores hayan sido más sencillos de implementar sabiendo que no tenemos que parsear ni usar funciones diferentes para obtener los datos del tiempo.

Bloque 3: Funcion constructora del mensaje

```
  const now = new Date();
msg.topic = "smartcity/consumo/test";
msg.payload = {
    home_id: "farola-001",
    consumption_kw: +(Math.random() * 2).toFixed(2),
    timestamp: now.getTime(),
    year: now.getFullYear(),
    month: String(now.getMonth() + 1).padStart(2, "0"),
    day: String(now.getDate()).padStart(2, "0"),
    hour: String(now.getHours()).padStart(2, "0"),
    minute: String(now.getMinutes()).padStart(2, "0"),
    seconds: String(now.getSeconds()).padStart(2, "0"),
};
return msg;
```

- **MQTT Out:** Este nodo está configurado para publicar el mensaje en el endpoint de AWS IoT (a1paa0c5brn8mp-ats.iot.us-east-1.amazonaws.com) a través del puerto 8883 con TLS. La autenticación se realizó utilizando los mismos certificados de dispositivo que en la prueba de conexión inicial. Usamos TLS el cual contiene todos los certificados necesarios para conecarnos al endpoint que anteriormente hemos introducido.

1.3 Pruebas

Para comprobar que cada una de las acciones anteriores de la regla funcionen debemos de comprobarlo lanzando el Node-RED. El primer paso es comprobar que los mensajes son enviados al tema correspondiente, para esto nos vamos al cliente de prueba de MQTT dentro de la pestaña de IoT de AWS y nos subscribimos a `smartcity/consumo/#` cuando nos subscribamos nos empezaran a salir mensajes como estos.

Respecto al guardado de mensajes historicos debemos de comprobarlo dentro del S3 de la practica, nos dirigimos a la sucesiones de carpetas que tenemos configurada y dentro de la ultima vemos como

tenemos estos datos. Los datos de dentro de cada archivo tambien corresponden con el .json definido en el SQL.

Finalemente debemos de comprobar la tabla de DynamoDB, dentro de vigia_farolas_estado, y en explorar elementos de la tabla vemos como nuestra tabla contiene una farola001 con el json correspondiente.

2 Preguntas de reflexión

2.1 Seguridad en IoT

2.2 Escalabilidad del sistema

2.3 Diferencias entre S3 y DynamoDB para este caso de uso

2.4 Ventajas de Node-RED para prototipado

2.5 Alternativas a la regla de IoT