

Avance 4: Arquitectura Cloud en AWS para FleetLogix

1. Introducción

FleetLogix necesita evolucionar desde un entorno local hacia una arquitectura cloud que permita recibir información en tiempo real, almacenar históricos, procesar eventos automáticamente y soportar decisiones basadas en datos.

Este avance propone el diseño de una solución serverless en AWS que cumple con esos requerimientos, integrando servicios como **API Gateway, AWS Lambda, S3, DynamoDB y RDS**.

El objetivo principal es demostrar comprensión de los componentes cloud y cómo se articulan para formar un sistema completo de ingesta, almacenamiento, procesamiento y análisis.

2. Arquitectura General en AWS

La arquitectura diseñada sigue un enfoque **serverless**, lo que significa que no se gestionan servidores y los recursos escalan automáticamente según la demanda.

Componentes Principales

API Gateway – Punto de entrada

Recibe datos enviados desde las aplicaciones móviles de los conductores.

Permite exponer endpoints seguros y escalables sin gestionar infraestructura.

AWS Lambda – Procesamiento en tiempo real

Tres funciones Lambda procesan los eventos logísticos:

- 1. Verificar si una entrega se completó**
Consulta DynamoDB para determinar estado actualizado.
- 2. Calcular ETA (tiempo estimado de llegada)**
Utiliza distancia restante + velocidad promedio para generar una estimación.
- 3. Enviar alerta de desvío de ruta**
Calcula distancia entre ubicación real y ruta planificada.
Si excede un umbral, dispara una alerta vía SNS y registra el incidente en DynamoDB.

S3 – Almacenamiento histórico

Guarda todos los registros crudos enviados por las apps móviles, organizados por fecha.

Sirve como "data lake" base para análisis futuros.

DynamoDB – Estado actual de entregas

Base NoSQL optimizada para accesos de baja latencia:

- Tabla 1: *FleetLogix_Deliveries_Status* (estado en tiempo real)
- Tabla 2: *FleetLogix_Alerts* (incidentes y desvíos)

RDS PostgreSQL – Migración de base transaccional

Es la base administrada que reemplaza el PostgreSQL local.

Incluye:

- Backups automáticos (7 días)
- Acceso público para conexión desde herramientas como DBeaver
- Instancia económica para ambiente de desarrollo

3. Data Flow

1. **Conductor envía actualización de entrega o GPS** → llega a **API Gateway**

2. Gateway envía el evento a la **Lambda correspondiente**

3. Lambda procesa el evento y puede:

- Actualizar estado en **DynamoDB**
- Calcular ETA
- Registrar alertas
- Enviar notificaciones vía **SNS**
- Guardar copia cruda en **S3**

4. En paralelo, los datos procesados quedan listos para integrarse al Data Warehouse en Snowflake.

La arquitectura permite ampliar la solución hacia análisis en tiempo real (Kinesis) o dashboards operativos.

4. Scripts Implementados

4.1 Infraestructura como Código (IaC)

Archivo: 01-aws_setup.py

Incluye la creación de:

- Bucket S3
- Tablas DynamoDB
- Instancia RDS PostgreSQL
- Configuración inicial de backups, almacenamiento y accesibilidad

Demuestra comprensión del aprovisionamiento automatizado mediante boto3.

4.2 Lambdas del sistema

Archivo: 02-lambda_functions.py

Implementa:

- Validación de entrega
- Cálculo de ETA
- Detección de desvíos
Incluye cálculos geográficos (Haversine), manejo de DynamoDB y uso de SNS.

4.3 API Gateway (Extra)

Archivo opcional avanzado que:

- Crea una API REST
 - Define rutas /delivery, /route, /vehicle
 - Integra Lambdas con métodos HTTP
 - Publica la API en stage prod
-

5. Conclusión

La arquitectura diseñada integra servicios gestionados, permite operación en tiempo real y se encuentra lista para escalar junto a futuras necesidades analíticas.

FleetLogix ahora posee:

- Punto de entrada único para eventos móviles
- Procesamiento automático e inteligente
- Almacenamiento de históricos
- Estado operativo en DynamoDB
- Base transaccional en RDS
- Capacidades de alertas en tiempo real

Este diseño posiciona a la empresa para futuros módulos de machine learning, dashboards operativos y analítica predictiva.
