FleetLogix - Arquitectura AWS

Avance 4: Implementación en la Nube

- Diagrama de Arquitectura
- Descripción de la Arquitectura

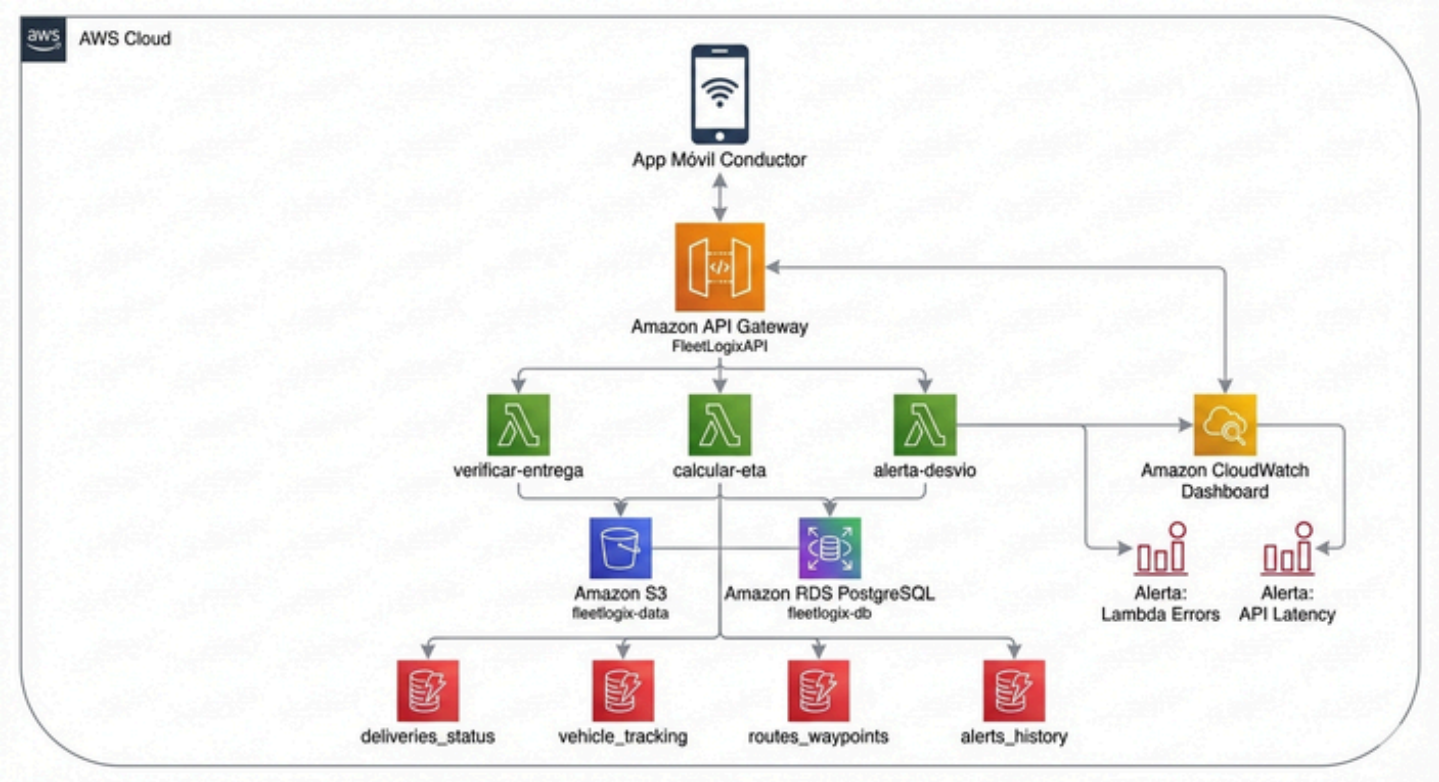
Esta arquitectura implementa un sistema completo de gestión de entregas en AWS que permite:

- Recibir y procesar datos en tiempo real desde apps móviles

- Almacenar información histórica y en tiempo real

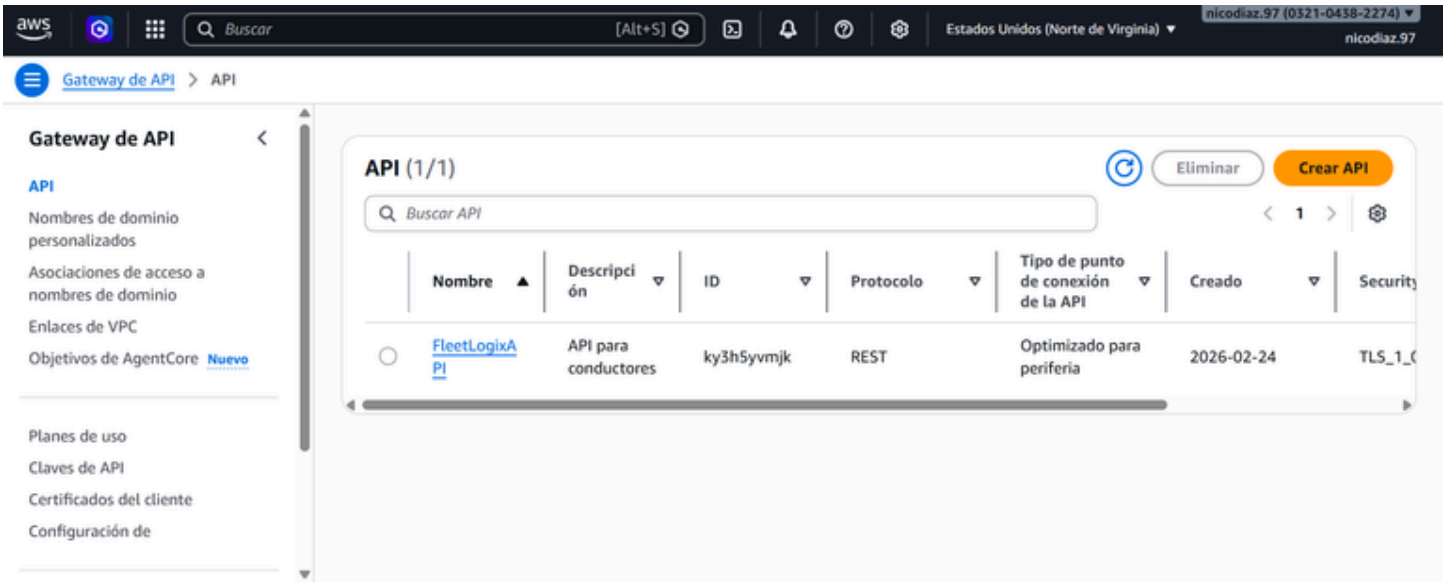
- Monitorear el estado del sistema automáticamente

- Detectar y alertar sobre problemas operacionales



- ## Componentes Implementados

- ### API Gateway (Puerta de Entrada)



- Nombre: FleetLogixAPI

- Tipo: REST API

- Función: Punto de entrada para las aplicaciones móviles de conductores

- Endpoints:

  - `POST /verificar-entrega` → Consulta estado de entregas

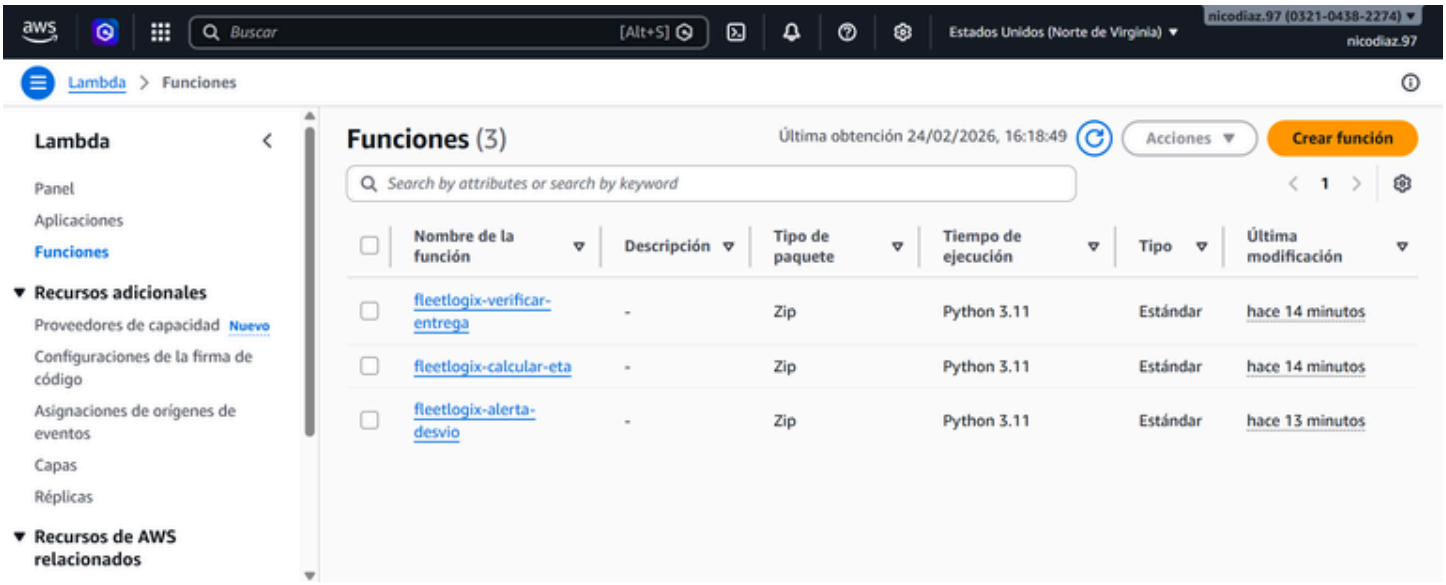
  - `POST /calcular-eta` → Calcula tiempo estimado de llegada

  - `POST /alerta-desvio` → Registra desvíos de ruta

- Seguridad: HTTPS, integración con Lambda mediante AWS_PROXY

- **Lambda Functions (Procesamiento)**

Tres funciones serverless que procesan las solicitudes:



- **Lambda 1: `verificar-entrega`**

- Propósito: Verificar si una entrega se completó

- Input: `delivery_id`, `tracking_number`

- Proceso: Consulta tabla DynamoDB `deliveries_status`

- Output: Estado de la entrega (completed/pending)

- **Lambda 2: `calcular-eta`**

- Propósito: Calcular tiempo estimado de llegada

- Input: `vehicle_id`, `current_location`, `destination`, `current_speed_kmh`

- Proceso:

  - Calcula distancia usando fórmula Haversine simplificada

  - Estima tiempo basado en velocidad actual

  - Guarda tracking en DynamoDB

- Output: Distancia restante y ETA

- **Lambda 3: `alerta-desvio`**

- Propósito: Detectar desvíos de ruta

- Input: `vehicle_id`, `current_location`, `route_id`, `driver_id`

- Proceso:

- Obtiene waypoints de la ruta desde DynamoDB

- Calcula distancia mínima a la ruta

- Si excede 5 km, registra alerta

- Output: Información del desvío y alerta generada

- **S3 (Almacenamiento de Archivos)**

- Bucket: `fleetlogix-data`

- Estructura:

fleetlogix-data

1. raw-data # Datos crudos por fecha
2. processed-data # Datos procesados
3. backups # Respaldos
4. logs # Logs del sistema

- **RDS PostgreSQL (Base de Datos Relacional)**

- Instancia: `fleetlogix-db`

- Tipo: db.t3.micro (Free Tier)

- Engine: PostgreSQL 15.4

- Almacenamiento: 20 GB SSD (gp2)

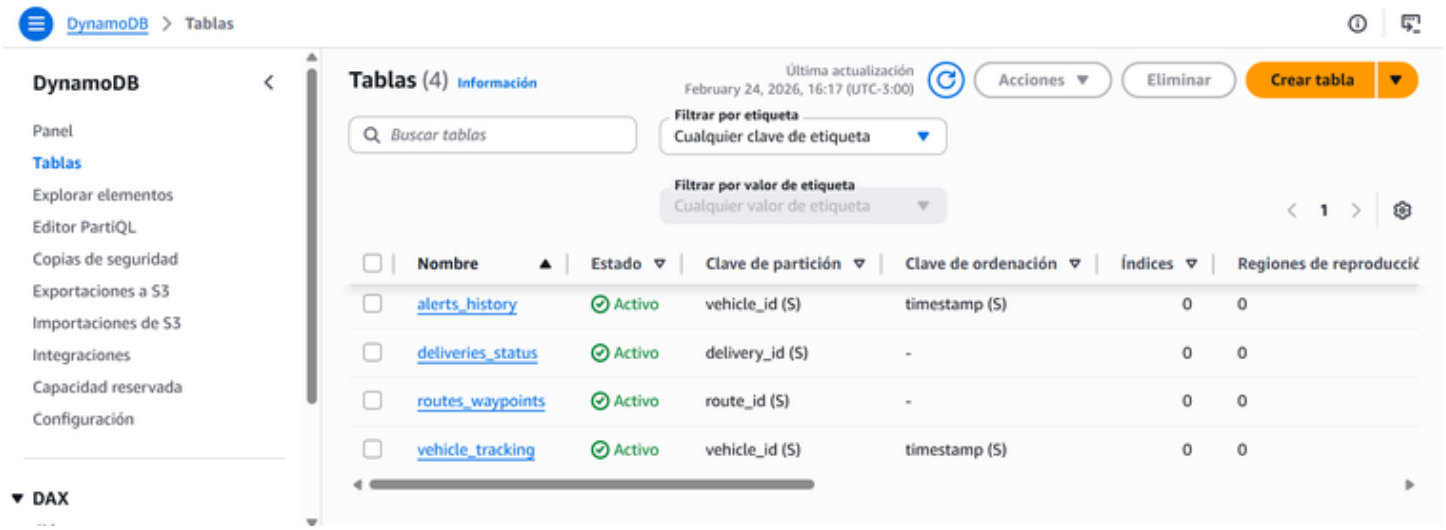
- Función: Base de datos principal migrada desde entorno local

- Backups: Automáticos cada 7 días (ventana: 03:00-04:00 UTC)

- Mantenimiento: Domingos 04:00-05:00 UTC

- ## DynamoDB (Base de Datos NoSQL)

Cuatro tablas para datos en tiempo real:



`deliveries_status` | delivery_id | Estado actual de cada entrega

`vehicle_tracking` | vehicle_id + timestamp | Ubicación GPS histórica

`routes_waypoints` | route_id | Puntos de control de cada ruta

`alerts_history` | vehicle_id + timestamp | Historial de alertas de desvíos

- Billing: On-demand (pago por uso)

- Ventaja: Latencia de milisegundos para consultas en tiempo real

CloudWatch (Monitoreo y Alertas)

Dashboard: `FleetLogix-Dashboard`

5 métricas principales monitoreadas:

- Lambda Invocations (Total de invocaciones)

- Namespace: `AWS/Lambda`

- Métrica: `Invocations`

- Statistic: Sum

- Lambda Duration (Tiempo de ejecución promedio)

- Namespace: `AWS/Lambda`

- Métrica: `Duration`

- Statistic: Average

- DynamoDB Read Capacity (Capacidad de lectura consumida)

- Namespace: `AWS/DynamoDB`

- Métrica: `ConsumedReadCapacityUnits`

- Statistic: Sum

- 4. API Gateway Requests (Total de peticiones)

- Namespace: `AWS/ApiGateway`

- Métrica: `Count`

- Statistic: Sum

- Lambda Errors (Errores en ejecución)

- Namespace: `AWS/Lambda`

- Métrica: `Errors`

- Statistic: Sum

Alertas Configuradas

Alerta 1: Lambda Error

- Condición: Errors > 5 en 5 minutos

- Propósito: Detectar problemas en las funciones Lambda

- Estado: Activa

Alerta 2: API Latency

- Condición: Latency > 1000ms promedio en 10 minutos

- Propósito: Detectar degradación de performance

- Estado: Activa

Flujo de Datos

1**. Conductor** abre la app móvil en su dispositivo

2. App envía solicitud HTTPS → API Gateway

3. API Gateway valida y enruta la solicitud a la **Lambda** correspondiente

4. **Lambda** se ejecuta y:

- Lee/escribe en **DynamoDB** para datos en tiempo real

- Consulta **RDS** para datos históricos si es necesario

- Guarda logs y datos procesados en **S3**

5. **CloudWatch** recolecta métricas de todos los servicios

6. Si hay problemas, **CloudWatch** dispara **alertas** automáticamente

7. Lambda devuelve respuesta → **API Gateway** → **App Móvil**

- **Recursos Creados**

✅ RDS PostgreSQL: fleetlogix-db

✅ S3 Bucket: fleetlogix-data

✅ DynamoDB: 4 tablas

✅ IAM Role: FleetLogixLambdaRole

✅ Lambda Functions: 3 funciones desplegadas

✅ API Gateway: FleetLogixAPI con 3 endpoints

✅ EventBridge: Triggers automáticos configurados

✅ CloudWatch Dashboard: FleetLogix-Dashboard

✅ CloudWatch Alarms: 2 alertas

- **Justificación Técnica**

API Gateway: Proporciona un endpoint HTTPS público, maneja autenticación, throttling y logging automáticamente.

Lambda: Modelo serverless que escala automáticamente, solo se paga por tiempo de ejecución (sin servidores que mantener).

S3: Almacenamiento de objetos altamente disponible (99.999999999% durabilidad), ideal para datos históricos.

RDS PostgreSQL: Base de datos relacional administrada para datos estructurados con relaciones complejas.

DynamoDB: Base de datos NoSQL de baja latencia para consultas en tiempo real de datos de tracking.

CloudWatch: Monitoreo integrado de AWS, sin necesidad de herramientas adicionales.

- **Seguridad**

- Encriptación: Datos en tránsito (HTTPS) y en reposo (S3, RDS, DynamoDB)

- IAM Roles: Permisos mínimos necesarios para cada servicio

- VPC: RDS puede configurarse en VPC privada para mayor seguridad

- Backups: Automáticos y retenidos por 7 días

- **Costos Estimados (Free Tier)**

- Lambda: 1M invocaciones gratis/mes

- API Gateway: 1M llamadas gratis/mes (primer año)

- RDS: 750 horas/mes de db.t3.micro gratis (primer año)

- DynamoDB: 25 GB almacenamiento gratis

- S3: 5 GB almacenamiento gratis (primer año)

- CloudWatch: 10 métricas y 10 alarmas gratis

- **Escalabilidad**

Esta arquitectura puede escalar a:

- Miles de solicitudes por segundo (API Gateway + Lambda)

- Millones de registros (DynamoDB + S3)

- Cientos de vehículos rastreados simultáneamente

- **Notas Finales**

- Las funciones Lambda están diseñadas para ser simples y demostrativas

- En producción se requieren validaciones adicionales, manejo de errores robusto y testing

- El diagrama muestra la arquitectura completa implementada

- Todos los servicios están configurados para uso de Free Tier cuando es posible

Archivos del Proyecto

Este proyecto incluye 3 archivos principales:

1. 04_aws_setup.py

- Script que crea toda la infraestructura AWS

- Crea RDS, S3, DynamoDB

- Despliega las 3 funciones Lambda

- Configura API Gateway con 3 endpoints

- Configura triggers automáticos

- Genera archivo de configuración

2. lambda_handler.py

- Código de las funciones Lambda

- lambda_verificar_entrega()

- lambda_calcular_eta()

- lambda_alerta_desvio()

3. ARQUITECTURA_AWS.md

- Este documento con diagrama y explicación completa