🧭 Proyecto Integrador II – *FleetLogix Data Modernization*

📘 Descripción general

FleetLogix es una empresa de transporte y logística que opera una flota de aproximadamente 200 vehículos que realizan entregas de última milla en cinco ciudades principales.
La compañía venía utilizando sistemas legacy y hojas de cálculo, lo que dificultaba la gestión operativa y el acceso a información en tiempo real.
El proyecto integrador tuvo como objetivo **modernizar la infraestructura de datos** mediante la creación de una base de datos transaccional, consultas analíticas optimizadas, un Data Warehouse en la nube y una arquitectura serverless orientada al análisis y la toma de decisiones basada en datos.

🎯 Objetivo general

Diseñar e implementar una solución integral de datos para FleetLogix, abarcando el ciclo completo de generación, análisis, almacenamiento y procesamiento de información, con el fin de permitir un manejo eficiente, escalable y analítico de la operación logística.

🚦 Estructura del proyecto

El proyecto se desarrolló en **cuatro etapas** o avances consecutivos, cada uno enfocado en una fase específica de la modernización del ecosistema de datos de la empresa.

🧩 AVANCE 1 — Generación y carga de datos sintéticos

Objetivo: Poblar la base de datos PostgreSQL con datos sintéticos masivos, coherentes y representativos de las operaciones reales de la empresa.

Desarrollo

Se implementó un script en **Python**, utilizando las librerías **Faker** y **pandas**, para generar más de **500.000 registros** distribuidos en seis tablas interrelacionadas:

Tipo	Tabla	Registros	Descripción
Maestra	vehicles	200	Tipos y características de los vehículos

Tipo	Tabla	Registros	Descripción
Maestra	drivers	400	Conductores con licencias válidas y fechas coherentes
Maestra	routes	50	Conexiones entre las principales ciudades
Transaccional	trips	100.000	Viajes históricos con fecha, vehículo y conductor
Transaccional	deliveries	400.000	Entregas asociadas a cada viaje
Transaccional	maintenance	5.000	Mantenimientos preventivos y correctivos

Validaciones de calidad

- Verificación de integridad referencial entre tablas.
- Control de consistencia temporal (arrival > departure).
- Implementación de logs de carga y control de errores.

Resultado

Se obtuvo una base de datos transaccional completamente poblada, coherente y validada, lista para análisis y consultas analíticas posteriores.

📊 AVANCE 2 — Consultas SQL y análisis de rendimiento

Objetivo: Desarrollar consultas SQL que respondan a problemáticas reales del negocio y optimizar su rendimiento mediante índices y análisis de planes de ejecución.

Actividades realizadas

- Desarrollo de **12 consultas SQL** que abordan aspectos operativos clave:
  - Eficiencia de rutas y consumo de combustible.
  - Carga de trabajo y desempeño de conductores.
  - Mantenimiento preventivo de vehículos.
  - Cumplimiento y puntualidad en las entregas.
- Análisis de **planes de ejecución con EXPLAIN ANALYZE** para detectar cuellos de botella.

- Creación de **índices** sobre columnas críticas (status, departure_datetime, route_id, trip_id, etc.).
- Medición comparativa de tiempos antes y después de la optimización.

Resultados

- Reducción promedio del tiempo de ejecución de consultas en **13%**.
- Identificación de patrones de eficiencia y costos operativos.
- Elaboración de un informe técnico de performance con justificación de cada mejora.

🧱 AVANCE 3 — Data Warehouse y pipeline ETL automatizado

Objetivo: Diseñar e implementar un **modelo analítico en estrella** en Snowflake y un pipeline de carga automatizado desde PostgreSQL.

Diseño del modelo

Se definió un esquema estrella compuesto por:

- **Tabla de hechos:** fact_deliveries
  Contiene métricas operativas y financieras (tiempo de entrega, distancia, combustible, ingresos, retrasos).
- **Tablas de dimensiones:**
  - dim_date: calendario completo con atributos temporales.
  - dim_vehicle: características técnicas y estado histórico de vehículos.
  - dim_driver: datos personales y desempeño.
  - dim_route: información geográfica y distancia.
  - dim_customer: datos de clientes y zonas de entrega.
  - dim_time: franja horaria y turnos.

Implementación técnica

- Creación de base de datos y tablas en **Snowflake** con escalado automático y “time travel”.
- Pipeline de carga **ETL** desarrollado en Python:
  - Extracción de datos desde PostgreSQL.

    - Transformación y validación de métricas con **pandas**.
    - Manejo de cambios históricos mediante **SCD Type 2**.
    - Carga incremental y registro de logs.
- Automatización diaria con schedule o tareas programadas.

Resultado

Data Warehouse operativo y actualizado, preparado para análisis históricos, tableros ejecutivos y explotación analítica en herramientas BI.

☁️ **AVANCE 4 — Arquitectura cloud en AWS**

Objetivo: Implementar una arquitectura en la nube que permita la ingesta y el procesamiento de datos de flota en tiempo real, bajo un enfoque **serverless**.

Diseño de arquitectura

La solución se construyó sobre **AWS**, integrando los siguientes componentes:

A. Recepción y almacenamiento de datos

- **API Gateway:** punto de entrada para aplicaciones móviles de los conductores.
- **Amazon S3:** almacenamiento de datos históricos organizados por fecha.
- **AWS Lambda:** funciones para procesar, validar y almacenar información de entregas.

B. Procesamiento automático

Se desarrollaron funciones **Lambda** en Python para:

1. Verificar la finalización de entregas.
2. Calcular tiempos estimados de llegada (ETA).
3. Enviar alertas en caso de desvíos de ruta.

C. Bases de datos en la nube

- **AWS RDS (PostgreSQL):** base transaccional migrada desde el entorno local.
- **AWS DynamoDB:** almacenamiento del estado actual de las entregas en tiempo real.

- **Backups automáticos:** configurados para garantizar la disponibilidad y recuperación de datos.

Resultado

Arquitectura cloud escalable y sin servidores, capaz de recibir, procesar y almacenar datos en tiempo real, facilitando la analítica inmediata y el control operativo.

🧱 Tecnologías utilizadas

Categoría	Herramientas
Lenguaje principal	Python 3.x
Bases de datos	PostgreSQL, Snowflake, DynamoDB
Cloud & Serverless	AWS S3, Lambda, RDS, API Gateway
ETL / Automatización	pandas, SQLAlchemy, boto3, schedule
Generación de datos	Faker
Analítica SQL	PostgreSQL (EXPLAIN ANALYZE, índices, funciones ventana)
Control de calidad	Logs, validaciones, auditoría de cargas
Visualización (opcional)	Power BI, Looker Studio

🗂️ Estructura de entregables

📦 FleetLogix_PI2/

│

├── data_generation/

│ ├── generate_data.py

│ └── logs/

│

├── sql_analysis/

```
│   ├── queries.sql
│   ├── explain_analyze_results.txt
│   └── performance_report.docx
│
├── warehouse_snowflake/
│   ├── ddl_snowflake.sql
│   ├── etl_pipeline.py
│   └── config_scd2_notes.md
│
├── cloud_architecture/
│   ├── aws_diagram.png
│   ├── lambda_functions/
│   └── boto3_scripts.py
│
└── README.md
```

🧠 Conclusiones generales

- Se logró la evolución de una base local estática hacia una **plataforma integral de datos en la nube**, con capacidad de análisis histórico y monitoreo en tiempo real.
- La combinación de **PostgreSQL, Python, Snowflake y AWS** permitió cubrir el ciclo completo de ingeniería y análisis de datos.
- Se estableció una arquitectura escalable, automatizada y segura que habilita la toma de decisiones basada en datos.
- El proyecto sienta las bases para futuras integraciones de analítica avanzada, machine learning y dashboards ejecutivos.