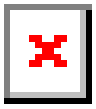
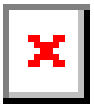
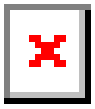
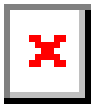
📊 Pipeline de Datos y Dashboard de Afluencia del Metro CDMX

Solución ETL y analítica para el procesamiento y visualización de datos de afluencia del Sistema de Transporte Colectivo de la Ciudad de México

---

📋 Tabla de Contenidos

🎯 Descripción General

Este proyecto implementa una solución integral de ingeniería de datos para el análisis de la afluencia diaria del Sistema de Transporte Colectivo Metro de la Ciudad de México. La solución comprende:

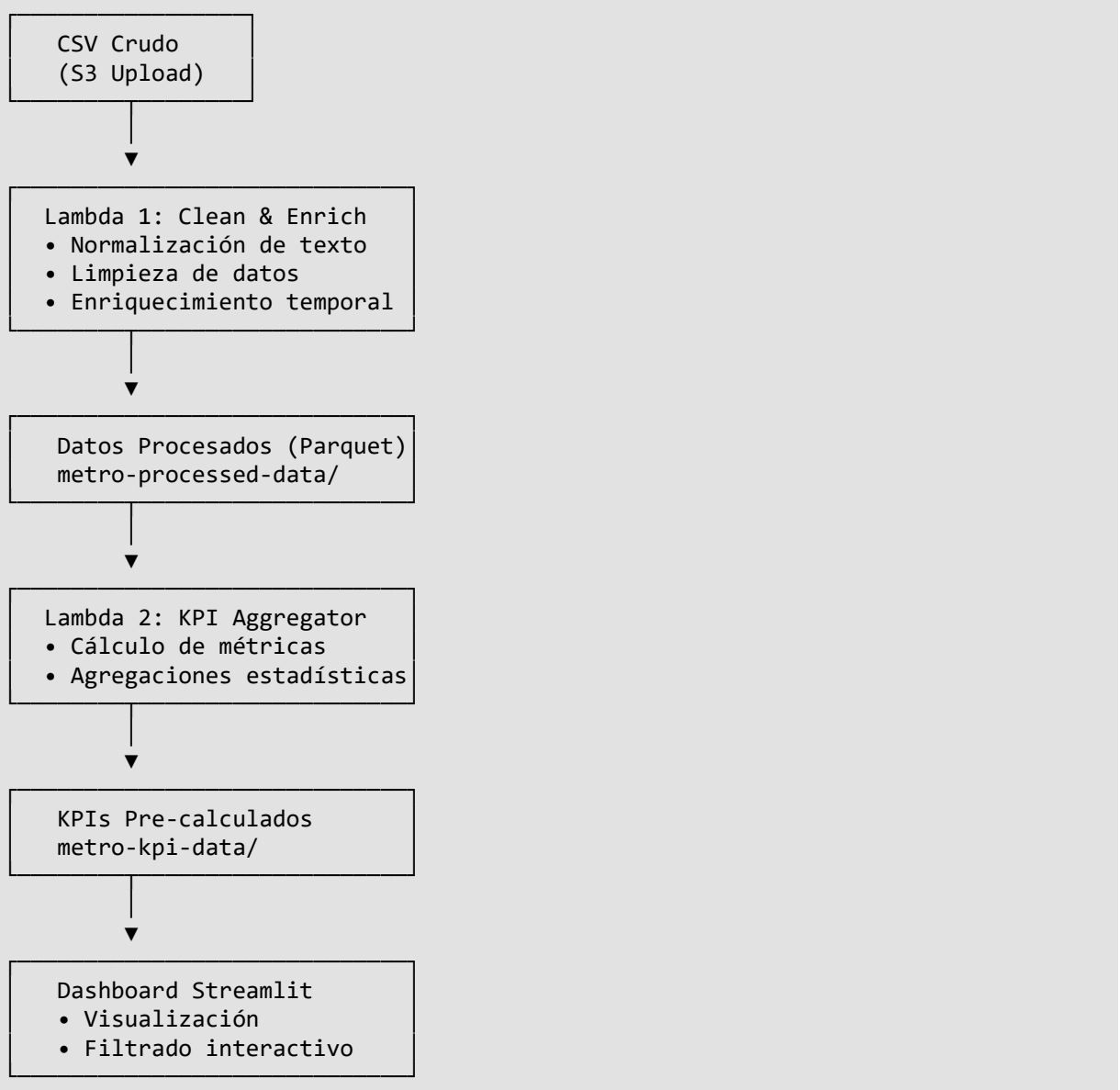
- **Pipeline ETL** en AWS con arquitectura serverless
- **Procesamiento de datos** mediante funciones Lambda especializadas
- **Almacenamiento optimizado** en formato Parquet sobre S3
- **Dashboard interactivo** desarrollado con Streamlit

Contexto Operativo

Debido a restricciones de permisos en el entorno AWS, el pipeline opera bajo un modelo de **ejecución manual controlada**, donde cada función Lambda se invoca secuencialmente. El dashboard implementa una **estrategia de carga híbrida** que combina KPIs pre-calculados para visualización rápida con procesamiento en tiempo real para análisis filtrado.

🏗️ Arquitectura del Sistema

Flujo de Datos



Componentes Principales

1. **Ingesta de Datos**

- Carga manual de archivos CSV a `s3://xideralaws-curso-alan/metro-raw-data/`
- Formato de entrada: CSV con datos de afluencia desglosada

2. **Lambda 1: Limpieza y Enriquecimiento**

Responsabilidades:

- Normalización de codificación de caracteres
- Limpieza de caracteres especiales y acentos
- Estandarización de nombres de columnas y líneas

- Conversión de tipos de datos
- Enriquecimiento con dimensiones temporales (`anio`, `mes`, `dia_semana`)

Output: Datos normalizados en formato Parquet en `metro-processed-data/`

3. **Lambda 2: Agregador de KPIs**

Responsabilidades:

- Cálculo de métricas agregadas sobre el histórico completo
- Generación de datasets especializados por KPI
- Optimización de consultas analíticas

Output: KPIs pre-calculados en `metro-kpi-data/`

4. **Dashboard Híbrido**

Estrategia de carga:

- **Carga inicial:** KPIs pre-calculados desde `metro-kpi-data/` (rendimiento óptimo)
- **Carga filtrada:** Dataset completo desde `metro-processed-data/` (análisis dinámico)

🛠️ Stack Tecnológico

Cloud Infrastructure

Tecnología	Propósito
AWS S3	Data Lake para almacenamiento de objetos
AWS Lambda	Funciones serverless para procesamiento ETL
AWS IAM	Gestión de identidades y permisos
AWS Jupyter Environment	Hosting del dashboard

Python Ecosystem

Librería	Versión	Uso
Pandas	Latest	Manipulación y análisis de datos
AWS Data Wrangler	Latest	Integración con servicios AWS
Streamlit	Latest	Framework de dashboard
Plotly Express	Latest	Visualizaciones interactivas

Data Format

- **Parquet**: Formato columnar optimizado para consultas analíticas

---

🗁 Estructura del Proyecto

```
metro-cdmx-pipeline/
│
├── 📄 ETL_limpieza_metro.py        # Lambda 1: Clean & Enrich
├── 📄 lambda_aggregator.py         # Lambda 2: KPI Aggregator
├── 📄 Dashboard_metro_app.py       # Aplicación Streamlit
├── 📄 requirements.txt             # Dependencias Python
└── 📄 README.md                    # Documentación principal
```

Estructura en S3

```
s3://xideralaws-curso-alan/
│
├── metro-raw-data/                 # Datos crudos (CSV)
│   └── afluenciastc_desglosado_08_2025.csv
│
├── metro-processed-data/           # Datos procesados (Parquet)
│   └── [archivos_parquet]
│
└── metro-kpi-data/                 # KPIs pre-calculados
    ├── afluencia_por_linea/
    ├── top_estaciones/
    ├── distribucion_pago/
    ├── afluencia_dia_semana/
    └── evolucion_mensual/
```

🚀 Guía de Implementación

Prerequisitos

1. **Configuración de AWS:**
    - Acceso a consola de AWS
    - Roles IAM con permisos S3 configurados
    - Funciones Lambda desplegadas
2. **Entorno Python:**

```
pip install streamlit pandas plotly awswrangler
```

Paso 1: Configuración de Funciones Lambda

Lambda 1: Clean & Enrich

- **Memoria:** 512 MB (recomendado)
- **Timeout:** 5 minutos
- **Código fuente:** `ETL_limpieza_metro.py`
- **Evento de prueba:** S3 Put Event apuntando a `metro-raw-data/`

Lambda 2: KPI Aggregator

- **Memoria:** 1024 MB (recomendado)

- **Timeout:** 10 minutos
- **Código fuente:** `lambda_aggregator.py`
- **Evento de prueba:**

```
{
"bucket": "xideralaws-curso-alan",
"key_folder": "metro-processed-data/"
}
```

Paso 2: Ejecución del Pipeline

2.1 Limpieza y Enriquecimiento

```
1. Subir CSV crudo a s3://xideralaws-curso-alan/metro-raw-data/
2. AWS Console → Lambda → lambda-clean-and-enrich
3. Ejecutar evento de prueba "S3 Put"
4. Verificar creación de archivos en metro-processed-data/
```

2.2 Agregación de KPIs

```
1. AWS Console → Lambda → lambda-aggregator
2. Ejecutar evento de prueba JSON
3. Verificar creación de archivos en metro-kpi-data/
```

Paso 3: Despliegue del Dashboard

En AWS Jupyter Environment:

```
# Instalación de dependencias
pip install streamlit pandas plotly awswrangler

# Navegación al directorio del proyecto
cd /path/to/project

# Ejecución del dashboard
streamlit run Dashboard_metro_app.py
```

Acceso:

- El comando generará una URL local
- Abrir en navegador para interactuar con el dashboard

📊 Dashboard y KPIs

Métricas Principales

Afluencia Total	Promedio Diario	Estación Líder
Suma agregada de todos los registros históricos	Afluencia promedio calculada por día	Estación con mayor volumen de usuarios

KPIs Implementados

◈ KPI 1: Afluencia por Línea

Visualización: Gráfico de barras verticales

- Colores oficiales por línea del Metro
- Ordenamiento descendente por volumen

◈ KPI 2: Top 10 Estaciones

Visualización: Gráfico de barras horizontales

- Ranking de estaciones más transitadas
- Análisis de puntos críticos de afluencia

◈ KPI 3: Distribución por Tipo de Pago

Visualización: Gráfico circular (pie chart)

- Porcentaje de uso por método de pago
- Análisis de preferencias de usuarios

◈ KPI 4: Afluencia por Día de Semana

Visualización: Gráfico de barras

- Patrones de comportamiento semanal
- Identificación de días pico

◈ KPI 5: Evolución Histórica Mensual

Visualización: Gráfico de líneas temporal

- Tendencias de afluencia a lo largo del tiempo
- Análisis de estacionalidad

Capacidades de Filtrado

- **Por Año:** Análisis de períodos específicos
- **Por Línea:** Enfoque en líneas particulares del Metro
- **Recalculación dinámica:** KPIs ajustados automáticamente según filtros activos

---

🗺 Roadmap

Fase 1: Optimización (Corto Plazo)

- [ ] Implementación de caché para consultas frecuentes
- [ ] Optimización de queries con Amazon Athena
- [ ] Mejora de tiempos de respuesta del dashboard

Fase 2: Automatización (Mediano Plazo)

- [ ] Configuración de triggers S3 automáticos
- [ ] Orquestación con AWS Step Functions
- [ ] Implementación de pipeline CI/CD

Fase 3: Expansión Analítica (Largo Plazo)

- [ ] Análisis comparativo año contra año
- [ ] Modelado predictivo de afluencia
- [ ] Análisis de estaciones de transbordo
- [ ] Integración con datos de incidencias

Fase 4: Calidad y Gobernanza

- [ ] Integración de Great Expectations para validación de datos
- [ ] Implementación de data lineage
- [ ] Alertas de calidad de datos
- [ ] Documentación automatizada de esquemas

Fase 5: Producción

- [ ] Despliegue en AWS App Runner
- [ ] Configuración de alta disponibilidad
- [ ] Implementación de monitoreo (CloudWatch)
- [ ] Documentación de runbooks operacionales

🤝 Contribuciones

Las contribuciones son bienvenidas y apreciadas. Para contribuir:

1. **Fork** el repositorio
2. Crea una **rama** para tu feature (`git checkout -b feature/AmazingFeature`)
3. **Commit** tus cambios (`git commit -m 'Add some AmazingFeature'`)
4. **Push** a la rama (`git push origin feature/AmazingFeature`)
5. Abre un **Pull Request**

Directrices de Contribución

- Mantener el estilo de código consistente
- Documentar nuevas funcionalidades
- Incluir tests cuando sea aplicable
- Actualizar la documentación correspondiente

📝 Notas Adicionales

Consideraciones de Seguridad

- Los datos crudos contienen información pública del STC Metro
- No se almacenan datos personales identificables
- Permisos IAM configurados con principio de mínimo privilegio

Rendimiento

- Archivos Parquet optimizados para consultas columnares
- Estrategia de carga híbrida minimiza latencia
- Dashboard responsivo para conjuntos de datos de tamaño medio

Limitaciones Conocidas

- Ejecución manual del pipeline debido a restricciones de permisos
- Filtros limitados para mantener rendimiento óptimo
- Dependencia de ambiente AWS específico

Desarrollado con ♡ para el análisis de movilidad urbana

Sistema de Transporte Colectivo Metro - Ciudad de México