Caso Técnico – Ingeniería de Datos Comercial

Implementación de Flujo ETL para Procesamiento de Entregas

Autor	José Carlos Porras
Versión	v1.0
Fecha	19/12/2025
Repositorio	**Repositorio-caso-ingenieria-de-datos**
Documento	Especificación técnica del flujo ETL

Índice

1. Resumen ejecutivo

Este documento describe el diseño e implementación de un flujo ETL en PySpark para procesar datos históricos de entregas a partir de archivos CSV. La solución incorpora parametrización centralizada mediante YAML y OmegaConf, permitiendo ajustar reglas de negocio (rango de fechas, país objetivo, factor de conversión y clasificación de entregas) sin modificar el código fuente. Como resultado, se genera una Capa Silver estandarizada y confiable, persistida en Parquet (consumo analítico) y replicada en CSV (auditoría), con particionamiento por fecha de proceso y trazabilidad mediante metadatos de carga.

2. Objetivo

Desarrollar un proceso robusto y parametrizable para la ingesta, transformación y carga de datos históricos de entregas.

3. Alcance

La solución abarca:

- Extracción de datos desde archivos planos (CSV).
- Filtrado por rango de fechas y país (aplicado tempranamente para optimizar recursos).
- Normalización de unidades de medida (por ejemplo, conversión de CS a ST).
- Clasificación de tipos de entrega (rutina y bonificación) basada en listas configurables.
- Reglas de calidad para consistencia, estandarización y limpieza (nulos, duplicados, tipos).
- Generación de salidas particionadas por fecha de proceso en Parquet y réplicas en CSV.

4. Supuestos y restricciones

Supuestos:

- El archivo fuente es un CSV accesible desde el equipo de ejecución.
- Las reglas de negocio se administran en config.yaml y se consideran la fuente de verdad para el procesamiento.
- La ejecución se realiza en modo local (Windows) utilizando una sesión de Spark en local[*].

Restricciones:

- No se incluye orquestación productiva (por ejemplo, Airflow) en el alcance del caso.
- La salida se escribe en disco local, por lo que depende de permisos de escritura y espacio disponible.

5. Arquitectura de la solución

La arquitectura sigue un patrón de ingesta y procesamiento por capas, con configuración centralizada:

- Entrada (Bronze): CSV histórico, filtrado temprano por fecha y país.
- Procesamiento: PySpark para transformaciones masivas y validaciones.
- Configuración: YAML + OmegaConf para parametrización de reglas de negocio.
- Salida (Silver): Parquet particionado por fecha de proceso + réplica CSV para auditoría.

6. Diseño del flujo ETL

6.1 Inicialización y parametrización

El proceso inicia leyendo el archivo config.yaml. Se cargan parámetros críticos como rango de fechas, país objetivo, factor de conversión (box_size) y listas de tipos de entrega válidos. Si el archivo de configuración no existe, el flujo se detiene (fail fast) para evitar ejecuciones con reglas indefinidas.

6.2 Módulo de extracción (get_data)

- Conexión: Localiza el archivo CSV utilizando la ruta definida en el archivo de configuración (config.yaml).
- Lectura inteligente: Utiliza Spark para inferir automáticamente el esquema (tipo de dato) de cada columna.
- Filtrado inmediato: Aplica filtros tempranos (país y rango de fechas) antes de transformaciones complejas, reduciendo el volumen procesado.

```
1
2  def get_data(spark, config):
3      #salida fecha proceso
4      #(output_path: data/processed/${fecha_proceso})
5      df = spark.read.option("header", True).option("inferSchema", True).csv(config.input.file_path)
6      df = df.withColumn("fecha_proceso", col("fecha_proceso").cast("string"))
7
8      return df.filter(
9          (col("fecha_proceso") >= config.filters.start_date) &
10         (col("fecha_proceso") <= config.filters.end_date) &
11         (col("pais") == config.filters.country)
12     )
```

6.3 Módulo de transformación (transform_data)

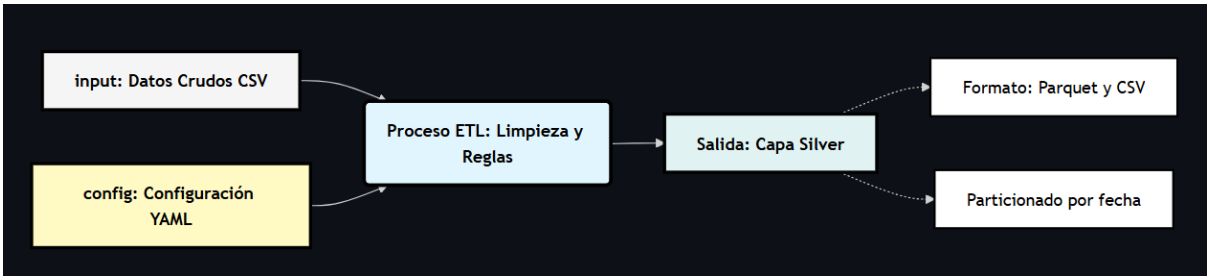
La etapa de transformación incluye normalización, enriquecimiento y limpieza:

- Normalización de unidades: conversión condicional de CS a ST mediante el factor box_size.
- Enriquecimiento: generación de atributos booleanos para entrega_rutina y entrega_bonificacion.
- Limpieza: eliminación de duplicados, manejo de nulos y validación de restricciones de negocio (cantidades y precios).
- Estandarización: filtrado de tipos de entrega únicamente a códigos válidos definidos en YAML.

```python

def transform_data(df, config):

    df = df.dropDuplicates()

    df = df.withColumn(
        "cantidad_unidades",
        when(
            col("unidad") == "CS",
            col("cantidad") * config.processing.box_size #llamamos al parametro box_size del config.yaml que tiene como valor 20
        ).when(
            col("unidad") == "ST",
            col("cantidad")
        ).otherwise(None) # Default null si no coincide ninguna unidad
    )


    routine_types = list(config.processing.delivery_types.routine)
    bonus_types = list(config.processing.delivery_types.bonus)
    all_types = routine_types + bonus_types


    df = df.filter(
        col("tipo_entrega").isin(all_types)
    )


    df = (
        df
        .withColumn(
            "entrega_rutina",
            col("tipo_entrega").isin(routine_types)
        )
        .withColumn(
            "entrega_bonificacion",
            col("tipo_entrega").isin(bonus_types)
        )
    )

    df = df.withColumn(
        "precio",
        when(col("precio") > 0, col("precio"))
        .cast(DecimalType(10, 2))
    )

    df = (
        df
        .dropna(subset=["cantidad_unidades"])
        .filter(col("cantidad_unidades") > 0)
    )

    return df
```

6.4 Load – Estandarización y persistencia (Capa Silver)

En la carga final se estandarizan nombres de columnas y se agregan metadatos de trazabilidad. Se persisten resultados en Parquet (comprimido con Snappy) particionado por fecha de proceso y se generan copias espejo en CSV para auditoría manual.



7. Parametrización mediante YAML (config.yaml)

Todos los parámetros del flujo son controlados desde un archivo YAML, incluyendo:

Parámetro	Descripción
Ruta de entrada	Directorio donde se ubica el CSV fuente.
Rango de fechas	Periodo a procesar; se utiliza para filtrar registros desde la lectura.
País	Código de país objetivo (por ejemplo, GT para Guatemala).
Conversión de unidades	Factor box_size para convertir CS (cajas) a ST (unidades).
Tipos de entrega	Listas de códigos válidos para clasificar entregas.

```yaml
# config/config.yaml
input:
  file_path: "../data/input/data_entrega_productos.csv"

filters:
  start_date: 20250101
  end_date: 20250630
  country: "GT"
  date_format: "yyyyMMdd"

processing:
  box_size: 20
  delivery_types:
    routine: ["ZPRE", "ZVE1"]
    bonus: ["Z04", "Z05"]

output:
  base_path: "../data/processed"
  format: "parquet"
  mode: "overwrite"
  partition_col: "fecha_proceso"
```

8. Reglas de calidad y consistencia de datos

El flujo aplica controles para garantizar integridad y estandarización:

- Eliminación de duplicados a nivel de fila completa.
- Conversión de unidades con manejo explícito de unidades no reconocidas (se asigna Null).
- Validación de precio: se preservan únicamente valores estrictamente positivos y se normalizan a Decimal(10,2).
- Validación de cantidad: se eliminan nulos y se restringen valores a cantidades mayores a cero.
- Filtrado de tipo_entrega contra listas de códigos válidos provenientes de configuración.

9. Orquestación y ejecución principal

La ejecución principal controla el ciclo de vida del proceso:

- Inicialización de Spark Session en modo local (local[*]) para utilizar todos los hilos disponibles del CPU.
- Bloque de control de errores (try-except-finally) para registrar fallas y liberar recursos de forma segura.
- Validación temprana (early stop): si no existen registros que cumplan con los filtros, el proceso finaliza sin generar salidas.

```yaml
# config/config.yaml
input:
  file_path: "../data/input/data_entrega_productos.csv"

filters:
  start_date: 20250101
  end_date: 20250630
  country: "GT"
  date_format: "yyyyMMdd"

processing:
  box_size: 20
  delivery_types:
    routine: ["ZPRE", "ZVE1"]
    bonus: ["Z04", "Z05"]

output:
  base_path: "../data/processed"
  format: "parquet"
  mode: "overwrite"
  partition_col: "fecha_proceso"

```

10. Tecnologías y librerías

Tecnologías utilizadas:

- Python 3
- PySpark
- OmegaConf (YAML)
- Jupyter Notebook / Visual Studio Code
- GitHub (repositorio público)

Librerías:

Librería	Uso principal
PySpark	Ingesta, filtrado y transformación masiva.
OmegaConf	Lectura de reglas de negocio y parametrización desde YAML.
Pandas	Escritura controlada de archivos locales (Parquet/CSV).

11. Entregables

Como resultado del caso técnico se entregan:

- Capa Silver en formato Parquet particionada por fecha de proceso.
- Copias espejo en CSV para auditoría y verificación manual.
- Archivo config.yaml de ejemplo para parametrización del flujo.
- Código fuente del flujo (módulos de extracción, transformación y carga) y README de ejecución.

12. Conclusión

La solución cumple con los requerimientos del caso técnico mediante un flujo ETL desarrollado en PySpark, aplicando prácticas de ingeniería de datos orientadas a calidad, estandarización y trazabilidad. La parametrización con OmegaConf y YAML aporta flexibilidad y reutilización, al permitir el ajuste de reglas de negocio sin modificar el código fuente. La estrategia de escritura local mediante Pandas habilita la generación de salidas en entornos Windows sin dependencias adicionales complejas, produciendo datasets listos para consumo analítico y auditoría.

Anexo A. Fragmentos representativos de transformación

A continuación se incluyen fragmentos representativos de la lógica aplicada sobre el DataFrame de Spark:

A.1 Eliminación de duplicados

```
df = df.dropDuplicates()
```

- Operación: Eliminación de redundancia.
- Descripción Técnica: Se aplica una transformación para identificar y remover filas idénticas en todo el dataset, garantizando los registros antes de iniciar el procesamiento computacional.

A.2 Normalización condicional de unidades (cantidad_unidades)

```
df = df.withColumn("cantidad_unidades",

  when(col("unidad") == "CS", col("cantidad") * config.processing.box_size)

  .when(col("unidad") == "ST", col("cantidad"))

  .otherwise(None)
```

- Condición**:** (Case When).
- Descripción Técnica**:** Se estandariza la unidad de medida aplicando una lógica de ramificación:
    - Caso CS**:** Se realiza una operación aritmética de multiplicación utilizando el factor de conversión **(box_size = 20)** desde la configuración de **YAML**.
    - Caso ST**:** Se deja el valor original.
    - (Otherwise): Se asigna None (Null) a cualquier unidad no reconocida para facilitar su depuración o eliminación en etapas posteriores.

A.3 Filtrado por pertenencia de conjuntos (tipo_entrega)

```
all_types = routine_types + bonus_types

df = df.filter(col("tipo_entrega").isin(all_types))
```

- Operación: Filtrado de tipo de entrega.
- Descripción Técnica: Se restringe el dataset únicamente a los registros cuyo tipo_entrega pertenezca a la lista de códigos válidos concatenados desde la configuración de **YAML**.

A.4 Creación de atributos booleanos

.withColumn("entrega_rutina", col("tipo_entrega").isin(routine_types))

.withColumn("entrega_bonificacion", col("tipo_entrega").isin(bonus_types))

- Operación: Generación de atributos booleanos.
- Descripción Técnica: Se crean dos nuevas columnas de tipo BooleanType evaluando la pertenencia del código de entrega a las sub-listas de "Rutina" o "Bonificación". Esto optimiza consultas.

A.5 Validación y normalización de precio

when(col("precio") > 0, col("precio")).cast(DecimalType(10, 2))

- Operación: Casteo de Tipo (*Type Casting*) con Redondeo.
- Descripción Técnica:
  - Validación: Se aplica una condición para preservar solo valores estrictamente positivos (> 0). Los valores incorrectos se convierten en Null.
  - Redondeo y Casteo: Se fuerza la conversión a DecimalType(10, 2). Esto no solo cambia el tipo de dato, sino que redondea aritméticamente los valores a 2 decimales, asegurando la precisión necesaria para cálculos monetarios y evitando errores de "punto flotante".

A.6 Integridad de datos para cantidades

.dropna(subset=["cantidad_unidades"])

.filter(col("cantidad_unidades") > 0)

- Operación: Manejo de Nulos y Restricciones Lógicas.
- Descripción Técnica:
  - dropna: Elimina los registros que resultaron en Null durante la normalización de unidades, limpiando datos con unidades de medida inválidas.
  - filter: Aplica una restricción de negocio para asegurar que no existan transacciones con cantidad cero o negativa en el dataset final.