Informe Tecnico: Analisis de Estrategias de Integracion ETL para ERP

Fecha: 30 de Diciembre de 2024
Elaborado por: Equipo de Mejoramiento Continuo
Version: 1.1

---

1. Resumen Ejecutivo

Este informe presenta un analisis comparativo entre dos estrategias para implementar procesos ETL (Extract, Transform, Load) que se integraran con el ERP corporativo basado en Java. Se evaluaron las opciones de **Java puro** versus un enfoque **hibrido (Java + Python)** para la carga masiva de datos desde archivos Excel hacia PostgreSQL.

Hallazgo principal: El enfoque hibrido presenta un rendimiento **8.7 veces superior** (3.6s vs 31.5s) con una complejidad de implementacion similar.

---

2. Contexto y Alcance

2.1 Problema a Resolver

- Cargar datos desde archivos Excel (.xlsm) hacia base de datos PostgreSQL
- Volumen de datos: ~38,000 filas por archivo
- Integracion con ERP monolitico existente en Java
- Ambiente de ejecucion: Servidor Windows

2.2 Opciones Evaluadas

Opcion	Descripcion
A. Java Puro	Apache POI + JDBC/PostgreSQL Driver
B. Hibrido - Python con Entorno Virtual	Java orquesta, Python (Polars) procesa en venv aislado

---

3. Benchmark de Rendimiento

3.1 Condiciones de Prueba

Parametro	Valor
Archivo origen	BASE DE DATOS GENERAL.xlsm
Filas procesadas	37,879
Columnas	30
Ubicacion archivo	Red compartida (\\192.168.0.3\...)
Base de datos destino	PostgreSQL 15
Hardware	Servidor Windows estandar

3.2 Resultados Medidos

Opcion A: Java Puro (Apache POI + JDBC)

```
[1/3] Leyendo archivo Excel...
     Filas brutas encontradas: 37889
     Lectura completada en 29,77 segundos
     Filas de datos: 37879

[2/3] Preparando subida a PostgreSQL...
     Tabla creada exitosamente.

[3/3] Insertando datos...
     Insertados: 37879 filas. COMPLETADO!
     Subida completada en 1,70 segundos

TIEMPO TOTAL JAVA: 31,47 segundos
```

Opcion B: Python (Polars + ADBC)

```
Leyendo archivo Excel...
     Archivo leido en 2.8 segundos
     Filas: 37879, Columnas: 30

Subiendo datos a PostgreSQL...
     Usando motor ADBC (Ultra Rapido)

Tiempo total operacion: 3.62 segundos
```

3.3 Comparativa de Tiempos

Fase	Java Puro	Python Polars	Diferencia
Lectura Excel	29.77s	2.8s	**-90.6%**
Subida PostgreSQL	1.70s	0.82s	**-51.8%**
TOTAL	**31.47s**	**3.62s**	**-88.5%**

Factor de mejora: 8.7x

---

4. Analisis Tecnico

4.1 Causa de la Diferencia de Rendimiento

Lectura de Excel

Aspecto	Java (Apache POI)	Python (Polars)
Implementacion	Java puro, orientado a objetos	Rust compilado, bindings Python
Modelo de memoria	Carga completa en heap JVM	Streaming con memoria optimizada

Paralelizacion	Single-threaded	Multi-threaded nativo
Optimizacion	Proposito general	Optimizado para datos tabulares

Apache POI es una libreria madura pero disenada para manipulacion general de documentos Office, no para procesamiento de alto rendimiento de datos.

Polars esta construido sobre Rust y utiliza optimizaciones de bajo nivel (SIMD, paralelismo, zero-copy) especificas para operaciones de datos.

Escritura a Base de Datos

Aspecto	Java (JDBC)	Python (ADBC)
Protocolo	JDBC estandar	Arrow Database Connectivity
Transferencia	Serializacion fila por fila	Columnar en bloques
Overhead	Mayor por abstraccion	Minimo, datos en formato nativo

ADBC (Arrow Database Connectivity) transfiere datos en formato columnar Apache Arrow, evitando conversiones intermedias.

---

5. Analisis de Opciones para Produccion

5.1 Opcion A: Java Puro

Ventajas

- Integracion nativa con el ERP (mismo stack tecnologico)
- Un solo lenguaje a mantener
- Sin dependencias externas al ecosistema Java
- Despliegue simplificado (un solo JAR)

Desventajas

- Rendimiento significativamente inferior (31.5s vs 3.6s)
- Apache POI consume mucha memoria para archivos grandes
- Complejidad para optimizar (requiere reescritura significativa)

Riesgos

- Timeouts en operaciones si los archivos crecen
- Presion de memoria en el servidor

Costo estimado de optimizacion

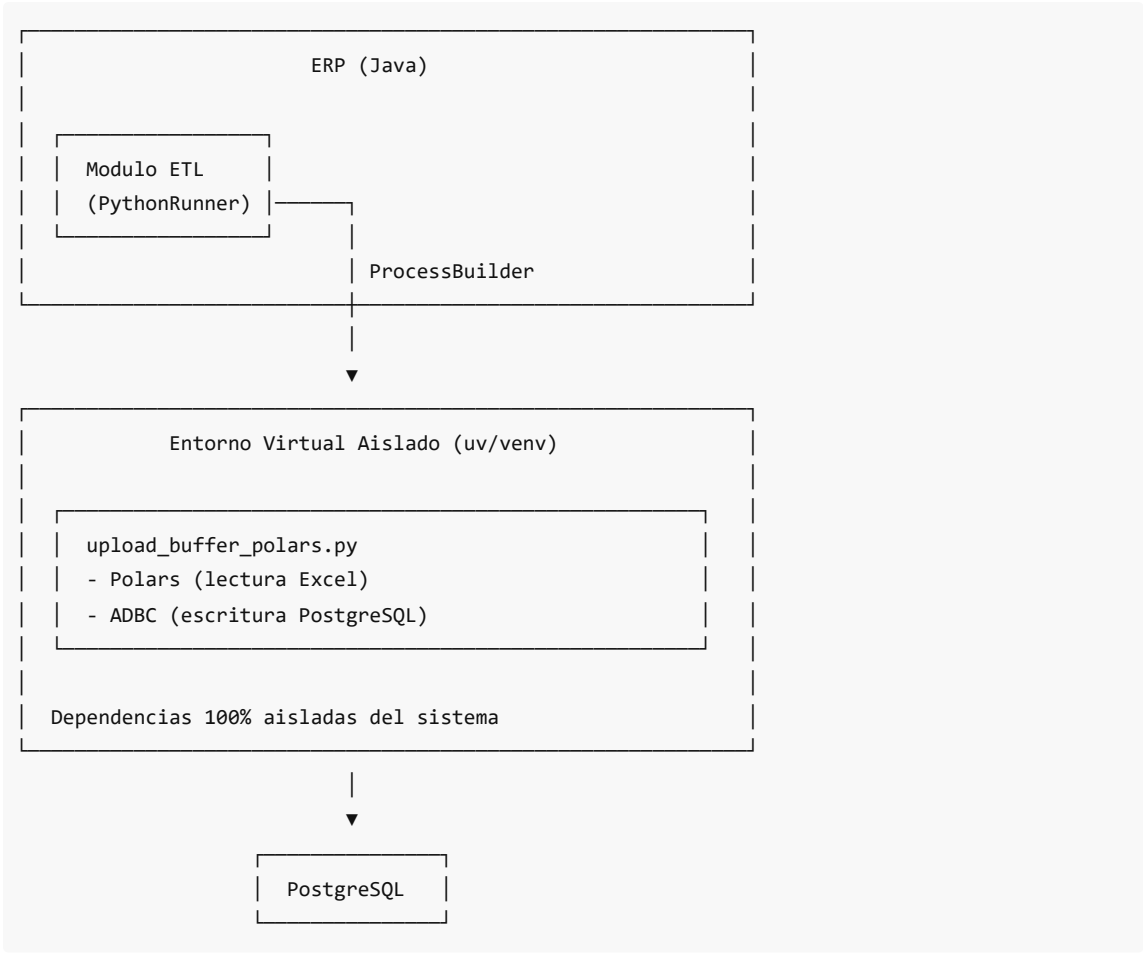
Para igualar el rendimiento de Python seria necesario:

- Evaluar librerias alternativas (FastExcel, StreamingReader)
- Implementar procesamiento paralelo manual
- Posiblemente reescribir logica de insercion con COPY command

Estimacion: 40-60 horas de desarrollo adicional, sin garantia de igualar rendimiento.

5.2 Opcion B: Hibrido con Python en Entorno Virtual Aislado

Arquitectura Propuesta



Ventajas

- Rendimiento optimo (3.6 segundos)
- **Aislamiento total**: Las librerias no afectan ni son afectadas por el sistema
- **Cero conflictos**: Cada proyecto puede tener su propio entorno
- Mantenimiento sencillo (editar script Python directamente)
- Facil actualizacion de dependencias
- Debugging simple con logs

Desventajas

- Requiere Python instalado en el servidor (solo el interprete base)
- Dos lenguajes a mantener

Por que el Entorno Virtual Evita Conflictos

Problema Comun	Solucion con venv
"Ya tengo otra version de X instalada"	Cada venv tiene sus propias versiones
"Actualizar libreria rompe otro proyecto"	Los venvs son independientes
"No tengo permisos para instalar global"	venv no requiere permisos admin

"El sistema usa Python 3.8, necesito 3.11"	Cada venv puede usar diferente Python

---

6. Gestion del Entorno Virtual: uv vs pip tradicional

6.1 Que es uv

uv es un gestor de paquetes Python moderno creado por Astral (los mismos de Ruff). Esta escrito en Rust y es significativamente mas rapido que pip.

6.2 Comparativa uv vs pip

Aspecto	pip + venv tradicional	uv
Velocidad instalacion	~30 segundos	~2 segundos
Resolucion dependencias	Lenta, a veces inconsistente	Rapida, deterministica
Archivo de lock	No nativo	uv.lock integrado
Compatibilidad Windows	Si	**Si**
Creacion de venv	`python -m venv`	`uv venv`
Reproducibilidad	Media	Alta

6.3 Instalacion de uv en Windows

```
# Opcion 1: Con PowerShell (recomendado)
powershell -c "irm https://astral.sh/uv/install.ps1 | iex"

# Opcion 2: Con pip (si ya tienes Python)
pip install uv
```

6.4 Comandos Basicos uv

```
# Crear entorno virtual
uv venv C:\ERP\python-env

# Activar (Windows)
C:\ERP\python-env\Scripts\activate

# Instalar dependencias (10-100x mas rapido que pip)
uv pip install polars sqlalchemy psycopg2-binary openpyxl

# O desde archivo requirements
uv pip install -r requirements.txt

# Generar lock file (reproducibilidad exacta)
uv pip compile requirements.txt -o requirements.lock
uv pip sync requirements.lock
```

6.5 Recomendacion

Usar uv para el servidor de produccion por:

1. Instalacion 10-100x mas rapida
2. Resolucion de dependencias deterministica (evita "funciona en mi maquina")
3. Archivo lock para reproducibilidad exacta
4. Funciona perfectamente en Windows Server

---

7. Matriz de Decision

Criterio	Peso	Java Puro	Hibrido + venv
Rendimiento	30%	2	10
Mantenibilidad	25%	8	8
Complejidad despliegue	15%	10	8
Estabilidad	15%	9	9
Costo implementacion	15%	5	9
TOTAL PONDERADO	100%	**6.15**	**8.85**

Puntuacion: 1 (peor) a 10 (mejor)

---

8. Recomendacion Final

Opcion Recomendada: Hibrido con Python en Entorno Virtual (usando uv)

Justificacion

1. **Rendimiento Critico:** La diferencia de 28 segundos por operacion es significativa.
2. **Aislamiento Total:** El entorno virtual garantiza que las dependencias estan completamente aisladas del sistema operativo y de otros proyectos Python.
3. **Sin Conflictos:** No importa que otras librerias o versiones de Python existan en el servidor. El venv es independiente.
4. **Reproducibilidad:** Con uv y un archivo lock, el entorno se puede recrear exactamente igual en cualquier momento.
5. **Bajo Riesgo:** La implementacion es sencilla y reversible.

---

9. Plan de Implementacion

9.1 Paso a Paso para Servidor Windows

```
# ============================================
# PASO 1: Instalar uv (una sola vez)
```

```
# ============================================
powershell -c "irm https://astral.sh/uv/install.ps1 | iex"

# Verificar instalacion
uv --version

# ============================================
# PASO 2: Crear estructura de carpetas
# ============================================
mkdir C:\ERP\etl
mkdir C:\ERP\etl\scripts
mkdir C:\ERP\etl\logs

# ============================================
# PASO 3: Crear entorno virtual aislado
# ============================================
uv venv C:\ERP\etl\venv

# ============================================
# PASO 4: Instalar dependencias
# ============================================
C:\ERP\etl\venv\Scripts\activate

uv pip install polars==0.20.0
uv pip install sqlalchemy==2.0.23
uv pip install psycopg2-binary==2.9.9
uv pip install openpyxl==3.1.2
uv pip install adbc-driver-postgresql==0.8.0

# Verificar instalacion
python -c "import polars; print('Polars OK:', polars.__version__)"

# ============================================
# PASO 5: Copiar script
# ============================================
copy "\\ruta\al\upload_buffer_polars.py" "C:\ERP\etl\scripts\"

# ============================================
# PASO 6: Probar ejecucion
# ============================================
C:\ERP\etl\venv\Scripts\python.exe C:\ERP\etl\scripts\upload_buffer_polars.py
```

9.2 Configuracion en PythonRunner.java

```
// Rutas de produccion
private static final String PYTHON_VENV = "C:\\ERP\\etl\\venv\\Scripts\\python.exe";
private static final String PYTHON_SCRIPT =
"C:\\ERP\\etl\\scripts\\upload_buffer_polars.py";
```

9.3 Tiempos Estimados

Fase	Actividad	Duracion
1	Instalar uv	5 min
2	Crear venv e instalar dependencias	10 min
3	Copiar y configurar scripts	15 min
4	Ajustar PythonRunner.java	30 min
5	Pruebas de integracion	2 horas
6	Documentacion	1 hora
Total		**~4 horas**

---

10. Mantenimiento Futuro

10.1 Actualizar Dependencias

```
# Activar entorno
C:\ERP\etl\venv\Scripts\activate

# Actualizar una libreria especifica
uv pip install polars --upgrade

# O reinstalar todo
uv pip install -r requirements.txt --upgrade
```

10.2 Recrear Entorno (si hay problemas)

```
# Eliminar entorno corrupto
Remove-Item -Recurse -Force C:\ERP\etl\venv

# Recrear desde cero
uv venv C:\ERP\etl\venv
C:\ERP\etl\venv\Scripts\activate
uv pip install -r requirements.txt
```

10.3 Backup del Entorno

```
# Exportar dependencias exactas
uv pip freeze > C:\ERP\etl\requirements-frozen.txt

# Este archivo permite recrear el entorno identico en otro servidor
```

---

11. Conclusion

El analisis demuestra que para el caso de uso especifico (carga masiva de Excel a PostgreSQL), el enfoque hibrido Java + Python ofrece una mejora de rendimiento de **8.7x** con un costo de implementacion minimo.

El uso de **entorno virtual con uv** garantiza:

- **Cero conflictos** con el sistema o otros proyectos
- **Instalacion rapida** (10-100x mas que pip)
- **Reproducibilidad** exacta del entorno
- **Compatibilidad total** con Windows Server

La implementacion puede completarse en aproximadamente **4 horas** de trabajo, con riesgo bajo y alta reversibilidad.

Anexos

A. requirements.txt

```
polars==0.20.0
sqlalchemy==2.0.23
psycopg2-binary==2.9.9
openpyxl==3.1.2
adbc-driver-postgresql==0.8.0
```

B. Estructura Final en Servidor

```
C:\ERP\etl\
├── venv\                    # Entorno virtual aislado
│   ├── Scripts\
│   │   ├── python.exe       # Interprete Python aislado
│   │   ├── pip.exe
│   │   └── activate.bat
│   └── Lib\
│       └── site-packages\   # Librerias instaladas (aisladas)
├── scripts\
│   └── upload_buffer_polars.py
├── logs\
└── requirements.txt
```

C. Verificacion de Aislamiento

```
# Python del sistema (si existe)
where python
# Salida: C:\Python311\python.exe

# Python del venv (completamente separado)
C:\ERP\etl\venv\Scripts\python.exe -c "import sys; print(sys.prefix)"
# Salida: C:\ERP\etl\venv

# Las librerias del venv NO afectan al sistema
# Las librerias del sistema NO afectan al venv
```

Fin del documento