Desarrollo de una herramienta para procesos ETL

Samuel Gil Arboleda

Práctica empresarial para optar al título de Ingeniero de Sistemas

Asesor
Luis Hernando Silva Florez, Magíster (MSc) en Ingeniería

Universidad de Antioquia
Facultad de Ingeniería
Ingeniería de Sistemas
Medellín
2023

Cita	Gil Arboleda [1]
Referencia Estilo IEEE (2020)	[1] S. Gil Arboleda, “Desarrollo de una herramienta para procesos ETL”, Práctica empresarial, Ingeniería de Sistemas, Universidad de Antioquia, Medellín, Antioquia, Colombia, 2023.

Centro de Documentación Ingeniería (CENDOI)

Repositorio Institucional: http://bibliotecadigital.udea.edu.co

Universidad de Antioquia - www.udea.edu.co

Rector: John Jairo Arboleda Céspedes.
Decano: Francisco Vargas Bonilla.
Jefe departamento: Diego José Luis Botía Valderrama

TABLA DE CONTENIDO

LISTA DE FIGURAS

SIGLAS, ACRÓNIMOS Y ABREVIATURAS

ETL.	Extract, transform and load.
BPO.	Business Process Outsourcing.
CRM.	Customer Relationship Management.
JAR.	Java Archive.
CSV.	Comma-separated values.
POJO.	Plain Old Java Object.
JPA.	Java Persistence API
CRUD.	Create, Read, Update, Delete.
JSON.	JavaScript Object Notation.

RESUMEN

Konecta es una empresa multinacional enfocada en BPO, contact center y soluciones de software. Tiene presencia en más de 9 países de 3 continentes. Además, cuenta con 15.000 talentos en Colombia que promueven el buen relacionamiento con clientes mediante soluciones y servicios de calidad. La fábrica de software Konecta requiere contar con un sistema que permita organizar, centralizar y ejecutar diferentes procesos de importaciones de datos o ETL. La solución que se busca en el semestre de industria consiste en el desarrollo de un sistema que pueda facilitar el proceso de importación masiva de datos de los CRM de la empresa.

***Palabras clave* — ETL, validar, leer, cargar.**

ABSTRACT

Konecta is a multinational company focused on BPO, contact center and software solutions. It has presence in more than 9 countries in 3 continents. In addition, it has 15,000 talents in Colombia that promote good customer relations through quality solutions and services. The software factory Konecta requires a system that allows organizing, centralizing and executing different data import processes or ETL. The solution sought in the industry semester consists of the development of a system that can facilitate the process of massive import of data from the CRM of the company.

***Keywords* — ETL, validate, read, load.**

I. INTRODUCCIÓN

La fábrica de software Konecta con base en los CRM de la empresa, requiere realizar de forma general y también de forma personalizable: la extracción de datos, la transformación y limpieza de estos, y por último, la carga en la base de datos correspondientes a los clientes, para la administracion y gestion en la toma de decisiones con relación a estos. La solución a desarrollar debe permitir organizar, centralizar y ejecutar diferentes procesos de importaciones de datos o ETL. Además, debe ser altamente parametrizable, de modo que tenga la capacidad de contener múltiples tipos de procesos de importaciones de datos y ser adaptable para varias secciones de la fábrica. El sistema debe estar hecho en Java y Spring Boot, de modo pueda ser exportado en un archivo Jar para ser ejecutado desde cualquier línea de comandos o servidor Backend.

II. OBJETIVOS

A. Objetivo general

Desarrollar un sistema de carga y validación de datos (ETL) con Java y Spring, para la gestión de los datos que se suben a los CRM.

B. Objetivos específicos

- Definir la estructura básica del flujo que tendrán los datos.
- Definir las reglas de validación generales.
- Definir las reglas de validación específicas para cada proyecto.
- Implementar los modelos de carga para los diferentes proyectos.
- Implementar un módulo que permita visualizar los resultados de los procesos de validación y carga.

III. MARCO TEÓRICO

Se requiere definir términos relacionados con la propuesta, para así tener total claridad de las herramientas con las cuales se trabajó:

- **ETL:** Es un tipo de integración de datos que hace referencia a los tres pasos (extraer, transformar, cargar) que se utilizan para mezclar datos de múltiples fuentes. Se utiliza a menudo para construir un almacén de datos. Durante este proceso, los datos se toman (extraen) de un sistema de origen, se convierten (transforman) en un formato que se puede almacenar y se almacenan en un data warehouse u otro sistema como bases de datos[1].

- **Java:** Java es un lenguaje de programación utilizado para el Backend de diversas soluciones de software.

- **Jar:** Un archivo JAR (por sus siglas en inglés, Java Archive) es un tipo de archivo que permite ejecutar aplicaciones y herramientas escritas en el lenguaje Java. Las siglas están deliberadamente escogidas para que coincidan con la palabra inglesa "jar" (tarro). Los archivos JAR están comprimidos con el formato ZIP y cambiada su extensión a .jar[5].

- **Spring Batch:** Un marco de trabajo por lotes ligero y completo diseñado para permitir el desarrollo de aplicaciones por lotes sólidos para las operaciones diarias de los sistemas empresariales.Spring Batch proporciona funciones reutilizables que son esenciales en el procesamiento de grandes volúmenes de registros, incluido el registro/rastreo, la gestión de transacciones, las estadísticas de procesamiento de trabajos, el reinicio de trabajos, la omisión y la gestión de recursos. También proporciona funciones y servicios técnicos más avanzados que permitirán trabajos por lotes de alto rendimiento y volumen extremadamente alto a través de técnicas de optimización y partición. Los trabajos por lotes de gran volumen, tanto simples como complejos, pueden aprovechar el marco de una manera altamente escalable para procesar volúmenes significativos de información[6].

- **Spring Boot:** es una herramienta que busca simplificar el trabajo con el framework Spring de Java. Ayuda a que la complejidad del desarrollo de nuevos proyectos se vea reducida, ya que dicha herramienta incluye pautas y bibliotecas relevantes para una amplia gama de aplicaciones[2].

- **Base de datos relacional - SQL:** Sistema de gestión de bases de datos relacionales de código abierto, en donde la información persiste en tablas relacionadas, las cuales están conformadas por filas y columnas. Las columnas son las variables que representan a la tabla, mientras que las filas son la información existente en dicha tabla [3].

- **SCRUM:** Es un proceso o metodología donde se realizan constantemente un conjunto de actividades para trabajar en equipo y obtener mejores resultados en el desarrollo de un proyecto[4].

IV. METODOLOGÍA

La metodología que se usó en el desarrollo del proyecto fue la metodología SCRUM (ver figura 1), estos fueron los pasos realizados para el desarrollo del trabajo:

- **Contextualización del problema:** El equipo de desarrollo se familiariza con el proyecto, esto se realizó con una reunión con el equipo anterior encargado y el profesor, ellos explicaron la situación problemática y el funcionamiento del proyecto.

- **Product backlog:** Se realizó una lista con las tareas de los módulos faltantes del proyecto.
- Sprint Backlog: Se seleccionaron los objetivos con mayor prioridad y se elaboró una lista con las tareas que se desarrollaron en los sprints.

- **Sprint:** Ciclo de trabajo donde se llevó a cabo el desarrollo de las tareas. Tuvieron una duración de 1 a 2 semanas.

- **Weekly meeting:** Reunión semanal para asesorías y avances del sprint.

- **Product preview presentation:** Reunión de revisión donde se entregó al encargado las tareas completadas durante el sprint, en ésta el tutor realiza las adaptaciones si las cree necesarias. Estas reuniones se realizaron al mismo tiempo que las del Weekly Meeting.

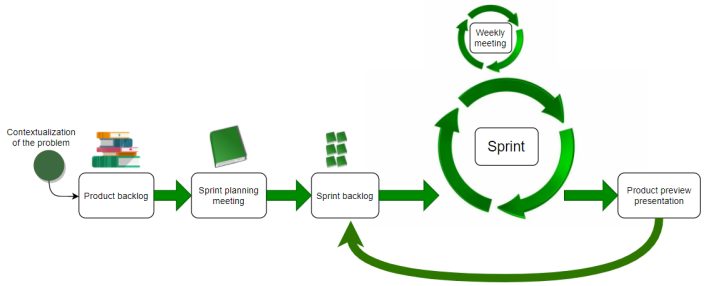

Figura 1. Ciclo de Scrum.

V. RESULTADOS

Configuración del proyecto

Para la creación del proyecto se usó spring initializer, como configuración del proyecto se escoge el lenguaje Java, hecho con gradle, el empaquetado en JAR y la versión de java es Java 11.

También se usa el tutorial que hay en la página oficial de spring[7], el cual nos brinda librerías con clases como: processor (para procesar los datos), listener (para realizar acciones antes de comenzar el proceso de carga), writer (para escribir los datos) y reader (para leer los archivos de los cuales se sacan los datos). Todas las anteriores clases se importaron en una sola clase de configuración, en esta se crean métodos que contienen todo el flujo que debe seguir el proceso ETL (ver figura 2).

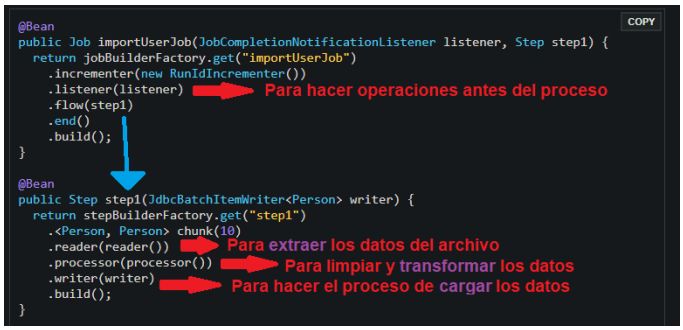

Figura 2. Métodos que contienen el flujo de la aplicación ubicados en la clase de configuración.

Ejecución de múltiples procesos batch

Una vez construido un proyecto base funcional, se busca que este funcione para varios procesos batch. El primer paso fue copiar múltiples veces toda la estructura del proyecto (ver figura 3 y 4), dándoles ahora nombre propio a las clases para identificar a qué proceso pertenece, esto técnicamente hace que tengamos varios proyectos ETL en un solo.

Figura 3. Carpetas de los procesos ETL que contiene la aplicación.

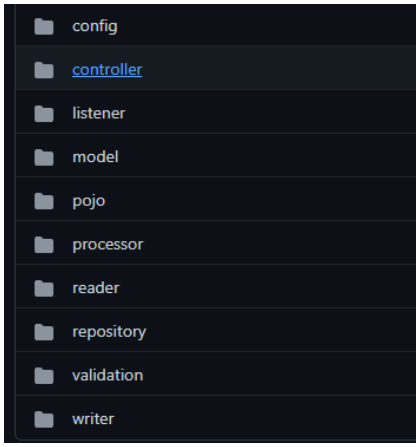

Figura 4. Carpetas que contienen cada proceso ETL.

Cuando se copió la clase de configuración con los métodos bean, se procedió a nombrar estos métodos, los nombres de estos representan los procesos ETL del proyecto (ver figura 5).

```
72      @Bean(name="originationStrategyJob")
73      public Job jobOriginationStrategy() throws Exception {
74              return jobBuilderFactory.get("originationStrategyJob")
75                              .incrementer(new RunIdIncrementer())
76                              .listener(completionListener)
77                              .flow(stepStrategyOrigination())
78                              .end()
79                              .build();
80      }
81
82      /**
83       * Step strategy origination.
84       *
85       * @return the step
86       * @throws Exception the exception
87       */
88      @Bean
89      public Step stepStrategyOrigination() throws Exception {
90              return stepBuilderFactory.get("Multithreaded: read -> Process->write ")
91                              .<RecordOriginationStrategy, OriginationStrategy>chunk(10000)
92                              .reader(readerOriginStrategy.read())
93                              .processor(processorOriginationStrategy())
94                              .writer(originationStrategyWriter)
95                              .taskExecutor(configJobs.taskExecutor())
96                              .build();
97      }
```

Figura 5. Clase de configuración con el método bean nombrado .

El primer limitante con esta configuración es que, por defecto, cuando se ejecuta el programa se realiza automáticamente el primer proceso ETL que encuentre, por lo que no es posible identificar cual ejecutar.
Primeramente para solucionar esto, se colocó en las propiedades del proyecto una configuración que hace que cuando sea ejecutado no comience ningún proceso batch, por lo cual ahora el proyecto no realiza nada en su ejecución (ver figura 6).

```
14    spring.batch.job.enabled=false
```

Figura 6. Propiedad para evitar una ejecución automática.

Ahora se necesita que cuando se ejecute el programa por una consola de comandos, se identifique el proceso ETL y se ejecute. Por lo anterior, se creó una clase que extrae los parámetros que vienen en el comando para identificar cuál proceso ejecutar.

```java
@SpringBootApplication
@EnableBatchProcessing
@EnableTransactionManagement
@EnableCaching
public class BancolombiaApplication implements ApplicationRunner {

	@Autowired
	private CommandLineProcessor commandLineReader;

	public static void main(String[] args) {
		SpringApplication.run(com.konecta.bancolombia.BancolombiaApplication.class, args);
	}

	public void run(ApplicationArguments arguments) throws Exception {
		this.commandLineReader.readLine(arguments.getSourceArgs());
	}
}
```

Figura 7. Clase principal llamando a la clase que identifica los procesos ETL.

Como puede verse en la figura 7, se tiene un método llamado “run” en la clase principal, el cual extrae los parámetros del comando y llama a la clase que se encarga de ejecutar el proceso ETL.

Posteriormente se determinó que él parámetro que viene desde el comando, debe ser el nombre del bean del proceso que se va a ejecutar, de esta manera usando la clase JobLauncher (la cual nos permite ejecutar un bean a partir de su nombre) podemos darle inicio al proceso en específico (ver figura 8).

```java
Job job1 = (Job) this.appContext.getBean(args[0]);
JobLauncher jobLauncher = (JobLauncher) this.appContext.getBean("jobLauncher");
JobParameters jobParameters = (new JobParametersBuilder())
		.addString("JobID", String.valueOf(System.currentTimeMillis())).addString("idProject", idProject)
		.toJobParameters();
JobExecution job = jobLauncher.run(job1, jobParameters);
```

Figura 8. Código que ejecuta el bean con el nombre del proceso ETL.

Con esta configuración ya tenemos el flujo principal de la aplicación, con diferentes procesos batch pudiendo ser ejecutados desde una consola de comandos.

Modelos de carga

Los modelos de carga son las estructuras que tienen los archivos planos y las clases POJO, estos archivos normalmente son generados desde un excel por una persona encargada y contienen toda la información que se carga en el proceso ETL.

En el caso de este aplicativo los archivos planos son archivos CSV separados por punto y coma, el tipo y la cantidad de columnas de cada proceso batch las definió el equipo de diseño de la fábrica.

```
File  Edit  Format  View  Help
ALIADO;ESTRATEGIA;CIUDAD;REGION;DEPARTAMENTO;CONSTRUCTORA;NIT CONST
KONECTA;SALA DE VENTAS;05001000;SUR;05;RUBAU;;TERRAGRATA;2375;;Paul
KONECTA;SALA DE VENTAS;05001000;SUR;05;MACANA;;NATURA;4641;;MYRIAM
KONECTA;SALA DE VENTAS;05001000;BOGOTA;ANTIOQUIA;SENDA CONSTRUCCION
KONECTA;SALA DE VENTAS;05001000;;05;;;;;1090332134;Angelica Rojas ;
KONECTA;SALA DE VENTAS;05001000;;05;;;;;1098732121;Johanna Suarez;3
KONECTA;SALA DE VENTAS;05001000;;05;;;;;1090332134;Julian Rodriguez
KONECTA;SALA DE VENTAS;05001000;;05;;;;;1090332134;Julian Rodriguez
```

Figura 9. Ejemplo de un archivo plano CSV.

El tipo de clases que usa cada proceso ETL son POJO y entidades de JPA. Las clases POJO deben tener como atributos los mismos campos que vienen en los archivos planos, ya que lo que

hace el reader es convertir cada línea del archivo plano en una instancia de la clase POJO. Estas clases POJO son las que se manejan durante todo el proceso ETL,donde sus campos son limpiados, borrados y otras veces son generados.

En caso de que se use el orm JPA, también se necesitan clases entidad y repositorio. Las clases entidad representan las tablas de la base de datos, a estas clases se les introduce la información que vengan de las clases POJO, este proceso se hace en el writer. Las clases repositorio nos proveen de métodos CRUD de una entidad para realizarlos en la base de datos.

```
4
5   public class AllyInmobiliaryPojo {
6
7           private String aliado;
8
9           private String estrategia;
10
11          private String ciudad;
12
13          private String region;
14
15          private String departamento;
16
17          private String constructora;
18
19          private String nitConstructora;
20
21          private String proyecto;
22
23          private String codigo;
24
```

Figura 10. Clase POJO.

En resumen, una vez definidas las estructuras de los archivos planos, se procedió a crear esas estructuras en el código del proyecto, tanto en la entrada como lo son las clases POJO, como en la salida como lo son las clases entidad que representan las tablas de la base de datos.

Reglas de validación generales

Todos los datos que son cargados a través de un proceso ETL deben ser validados antes, con el objetivo de mantener una base de datos limpia y disminuir la cantidad de errores que se puedan presentar durante su inserción en la base de datos.

Las reglas de validación generales que se definieron fueron:

- ➔ Tipo de dato
- ➔ Obligatoriedad
- ➔ Tamaño mínimo
- ➔ Tamaño máximo
- ➔ Valores permitidos (solo en algunos casos)
- ➔ Expresión regular (solo en algunos casos)

Las anteriores reglas fueron guardadas en la base de datos en un JSON, cada campo de cada proceso de carga tiene sus validaciones definidas en estos JSON. Lo que hace el aplicativo es retornar estas validaciones desde la base de datos y aplicarlas al campo respectivo durante el proceso de carga en la clase **processor**.

Figura 11. Ejemplo de un JSON con las validaciones de cada campo.

Reglas de validación específicas

Según la necesidad y a solicitud del equipo de diseño de la fábrica, los campos tendrán validaciones específicas según sus reglas de negocio.

Algunas de las validaciones específicas que se realizaron fueron:

- ➔ **Obligatoriedad según otro campo:** Por ejemplo si el campo “aliado” contiene el valor “Konecta” entonces el campo “estrategia” es obligatorio y si el campo “estrategia” viene vacío entonces se rechaza el registro.

- ➔ **Obligatoriedad según un campo de la base de datos:** Por ejemplo el campo “estrategia” tiene su lista de valores posibles en una tabla aparte en la base de datos, ya que sus valores son más dinámicos, entonces si en el archivo plano venía un valor diferente a los permitidos se rechaza el registro.

- ➔ **Inserción en una tabla según el valor:** Por ejemplo si el campo “aliado” contiene el valor “Konecta”, entonces el registro debe insertarse en la tabla de AliadosKonecta, en caso de que tenga otro valor se inserta en la tabla Aliados.

- ➔ **Generación de un campo según el valor de otro campo:** Por ejemplo si el campo “aliado” contiene el valor “Konecta”, entonces el campo “aceptaExperiencia” debe insertarse con el valor “si” y en caso contrario de que contenga otro valor se inserta con “no”.

- ➔ **Tamaño de un campo según el valor de otro campo:** Por ejemplo si el campo "segmento" tiene el valor "10", entonces el campo "teléfono" debe tener un tamaño mayor a 10 y en caso contrario se rechaza el registro.

Visualizar los resultados de los procesos

Para cada proceso, se guarda en la base de datos la información general y los resultados de todo el proceso de carga. La información guardada contiene los siguientes campos:

- ➔ Nombre del archivo plano
- ➔ Peso del archivo plano
- ➔ Cantidad de registros del archivo plano
- ➔ Cantidad de registros cargados
- ➔ Identificador de la persona que realizó el proceso
- ➔ Estado del proceso
- ➔ Progreso

Para visualizar estos datos de manera agradable para el usuario, otros miembros del equipo de desarrollo realizaron un módulo para mostrar estos datos en uno de los proyectos ya existentes en la fábrica.

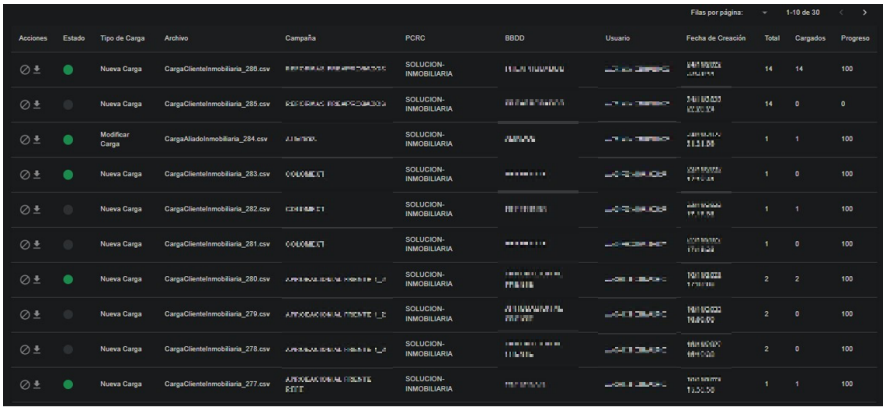

Figura 12. Visualización de resultados de los procesos de carga.

Integración con otros sistemas

Como se mencionó anteriormente, el empaquetado de este proyecto es en un JAR ejecutable, por lo cual puede ser usado fácilmente desde cualquier aplicación que tenga acceso a una consola de comandos, como lo pueden ser servidores de aplicaciones o cronjobs.
Actualmente el validador es usado desde 2 servidores de aplicaciones diferentes y desde un cronjob que ejecuta un proceso de carga todos los días.

VI. CONCLUSIONES

Fue un proceso de aprendizaje bastante útil, vivir en un entorno real con un marco de trabajo como lo es SCRUM fue bastante gratificante. Se experimentó como se pueden presentar inconvenientes y otros tipos de escenarios en los sprint, entregas y hasta el posterior mantenimiento del producto.

También fue bastante provechoso aprender a construir procesos ETL, manejar grandes cantidades de datos y estudiar cómo estos son migrados de un lado a otro. Este tipo de procesos a veces es poco conocido y normalmente realizado en grandes empresas que poseen grandes cantidades de datos, por lo que haber estado en este proyecto fue muy valioso.

REFERENCIAS

[1] "ETL: Qué es y por qué es importante" *Vista*. [Online]. Disponible en: https://www.sas.com/es_co/insights/data-management/what-is-etl.html

[2] Oblancarte, "Qué es Spring Boot y su relación con los microservicios," Oscar Blancarte - Software Architecture, 03-Jul-2020. [Online]. Disponible en: https://www.oscarblancarteblog.com/2018/07/17/spring-boot-relacion-los-microservicios /.

[3] "¿Qué es MySQL? Explicación Detallada Para Principiantes," Tutoriales Hostinger, 03-Dec-2020. [Online]. Disponible en: https://www.hostinger.co/tutoriales/que-es-mysql.

[4] "Qué es SCRUM," *Proyectos Ágiles*, 09-Oct-2018. [Online]. Disponible en: https://proyectosagiles.org/que-es-scrum/

[5] "JAR File Specification". [Online]. Disponible en: https://docs.oracle.com/javase/1.5.0/docs/guide/jar/jar.html

[6] "Spring Batch". [Online]. Disponible en: https://spring.io/projects/spring-batch

[7] "Creating a Batch Service". [Online]. Disponible en: https://spring.io/guides/gs/batch-processing/