2. Spring Batch.

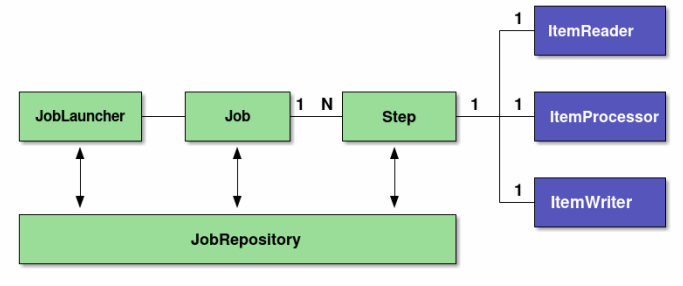
2.1 Datos Generales.

Es un aplicativo con características específicas que busca procesar grandes de cantidades de datos con diferentes *Data Sources* donde se debe disparar para comenzar el procesamiento. Dicho proceso suele demorar por la gran cantidad de información que suelen procesar de forma automática. A diferencia del herramientas de procesamiento para *Big Data*, *Spring Batch* el diseño y de esta solución no se preocupa por el tiempo sino por el correcto procesamiento.

- Los datos pueden tener diferentes formatos, tales como: XML, CSV, SQL, NoSQL, entre muchos otros.
- Un proyecto *Batch* puede: Importar o exportar archivos, procesar registros de BB.DD., generar reportes y facturas, migrar datos.
- Soporta transacciones, reintentos, saltos, reinicios y lectura/escritura en chunks (segmentación de registros).

2.1 Arquitectura.

La arquitectura consta de ***Config, Controller, Entity y Repository*** que tendrá distribuida los siguientes 4 elementos clave *JobLauncher, Job, JobRepository* y *Step* que a su vez constará de 3 *métodos* para lograr el procesamiento:



Config

Concentra todos las *Clases* y *métodos* necesarios para configurar el *elemento Job* dentro de la arquitectura *Batch:*

- ***CustomerProcessor:*** *Clase* que implementa la *Interfaz* ***ItemProcessesor<In, Out>*** y sobre escribe su único *método* que permitirá dictar condiciones sobre el procesamiento de los registros obtenidos de la fuente *JobRepository.*
- ***SpringBatchConfig:*** *Clase* clave que contendrá los *Annotations* ***@Configuration*** y ***@EnableBatchProcessing*** para ser declarada como el elemento de configuración *Batch*. Dentro de su estructura recibirá por inyección de dependencia a ***JobBuilderRepository, StepBuilderRepository*** *(ambas Clases de batch.core)* y ***CustomerRepository***.
  En resumen, contendrá los *métodos* de configuración de cada *Step* y *métodos reader(), processor(), writer(),* entre otros:
  - ***reader():*** Crea una instancia de la *Clase* que permita leer la fuente de datos, en el caso del ejemplo es *CSV,* por lo tanto, ***FlatFileItemReader<Type>****, Objeto* que permitirá leer la fuente de datos para retornar los datos como *Objeto*.
  - ***lineMapper():*** *Método* que permite complementar la lectura de los datos por procesar donde se inserta un delimitante sobre la lectura y clasificación de los datos por campos de cada registro.
  - ***proccesor():*** *Método encargado de crear la instancia de la Clase CustomerProcessor*.
  - ***writer():*** El *método* crea una instancia de la *Clase* ***RepositoryItemWriter<Type>*** que recibirá usará la inyección de *CustomerRepository* para insertar la información en la fuente de destino.
  - ***taskExecutor():*** El *método* crea una instancia de la *Clase* ***SimpleAsyncTaskExecutor*** que permitirá añadir características de ejecución.
  - ***step():*** Será el *método* que conjunte los *Objetos* generados por los *métodos* anteriormente mencionados para construir un solo *Objeto*. Además, podrá añadir ***chunks()*** que permitan modular el procesamiento de los datos por bloques para evitar saturación de memoria.
  - ***runJob():*** Configura la dependencia *Job* que será insertada en el *JobController*, recibe la cantidad de *Objetos step* generados, marca el final del *Job* y al final de los *métodos* añade *build()* que creará un solo *Objeto* para retornarlo.

Controller.

- ***JobController:*** *Clase* que recibe por inyección de dependencias a las *Interfaces* ***JobLauncher*** y ***Job****;* de *JobLauncher* se implementa el único *método* ***run()*** que recibe 2 argumentos: la implementación de *Job* y la implementación de la *Clase* ***JobParameter***, que recibe parámetros de inicialización.
- ***Job:*** Esta dependencia se configura en la *Clase SpringBatchConfig,* a través del *método runJob()*.
  El *método run()* tiene la declaración de 4 *Exceptions* las cuales todas *extiende* de *Exception (Checked),* por lo que es necesario rodearlo con *try/catch* para tratar cualquiera de los *Exceptions*.
- La *Clase* contiene el *endpoint* y el *verbo HTTP* con el cual pone a disposición la ejecución del *Batch* como *servicio*, sin embargo, es posible modificarlo para que sea ejecutado de forma automática y periódica.

Entity.

- ***Customer:*** *Clase POJO* que mapeará la tabla o fuente destino donde serán insertados los datos procesados.
- ***ProssesingReport:*** *Clase* que permitirá generar reportes sobre la información procesada dentro de la ejecución del *Batch*.

Repository.

- ***CustomerRepository:*** *Interfaz* que *extiende* de *JpaRepository<Type, ID>* que servirá como inyección de dependencia para proveer *métodos* a ***ReportItemReader*** y ***SpringBatchConfig***.

Ejecución.

La ejecución puede programarse para que se realice de forma automática sin peticiones, sin embargo, para el ejemplo realizado es diseñado como un servicio *REST POST* y se consume como servicio a través de un *Cliente* externo.