**Spring Batch 4.2**

Batch Processing

* Processing large volumes of data is the primary use case for batch processing applications.
* It is a crucial task in most organizations but is often done using customized solutions which are hard to develop & maintain and are usually not optimized.
* Spring Batch addresses all of these limitations & makes batch processing a whole lot easier.
* Spring Batch is a module of Spring framework which makes it very easy to develop & maintain large – scale batch processing applications.

**Batch Processing:** It is the execution of **repetitive tasks** on a **large volume of data** with **little or no human interaction**.

For e.g., reading 1 million lines from a CSV file & to process each of the lines with same set of tasks need to be carried out.

**Examples of Batch Processing**

e.g.,1: Loading a CSV file’s data into a relational database

e.g.,2: Reporting trades on an hourly basis to a regulator

e.g.,3: Recording user activity in an analytics platform

e.g.,4: Compiling monthly usage to generate a phone bill

**Common Batch Processing Steps**

Step 1: **Extract**/read data from an input source

Step 2: **Transform** data to a required format

Step 3: **Load**/write data to an output location

* These steps look familiar to us because they form part of an ETL process i.e., Extract, Transform & Load & it’s within the broad family of ETL processes where **Batch Processing** often plays a crucial role.

**Batch Processing Properties**

1. Extract-Transform-Load (ETL) is a common type of operation.
2. Job are typically scheduled (as opposed to real-time processing)
3. Ca be set to run when resource utilization is low
4. Large volume of data means it is better processed in batches.

**Problems with Batch Processing (Especially when customized solutions are built by developers)**

1. Many teams build bespoke solutions for their own input/processing/output needs.
2. Such solutions can’t be reused for new batch processing jobs.
3. New solutions need to be built for each processing task.

* The drawbacks here involve not just a lot of added development work, but also the fact that there are many more applications & components to maintain, which increases the likelihood of bugs in the applications.

1. Processing jobs at scale requires complex optimizations.
2. Features such as logging need to be integrated to simplify debugging.
3. Execution metrics need to be generated & maintained for analytics.

* While Traditional batch processing applications come with a lot of drawbacks, Spring batch tackles most of these.
* Spring batch makes the development as well as maintenance of batch processing applications a whole lot simpler than the traditional bespoke solutions adopted.

Official documentation: <https://docs.spring.io/spring-batch/docs/current/reference/html/spring-batch-intro.html#spring-batch-intro>

**Spring Batch (Acc. To official Documentation)**

* Spring Batch is a comprehensive, light-weight framework for the development of robust batch applications for use in enterprise systems.

**Spring Batch Features**

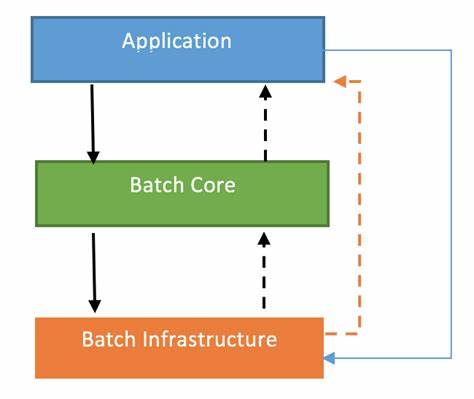
1. Spring Batch is an extension of the Spring framework specifically target at batch processing applications. (E.g., uses beans)
2. Spring Batch makes development a lot easier by providing a lot of **reusable functions & components** for common data processing tasks. For e.g., Loading data from a CSV or JSON file.
3. Keep in mind that Spring Batch is not a scheduling framework & is meant to work alongside one i.e., we can use Spring batch to develop a batch processing application & then schedule it using a different scheduler tool.

**Reusable Components in Spring Batch**

* One of the benefits of Spring Batch is that we don’t need to build components from scratch which are meant to perform common tasks. For e.g., Read from / write to JSON, XML or CSV files.
* We can make use of the pre-built components that are already present in Spring Batch & then make minor tweaks in their configurations for each specific file.

1. Readers/Writers for common file formats – JSON, XML, CSV
2. Readers/Writers for relational & NoSQL databases.
3. Logging & Transaction management APIs
4. Adapters to translate POJOs to specific formats. For e.g., converting Java objects to XML or JSON.

**Spring Batch Architecture**



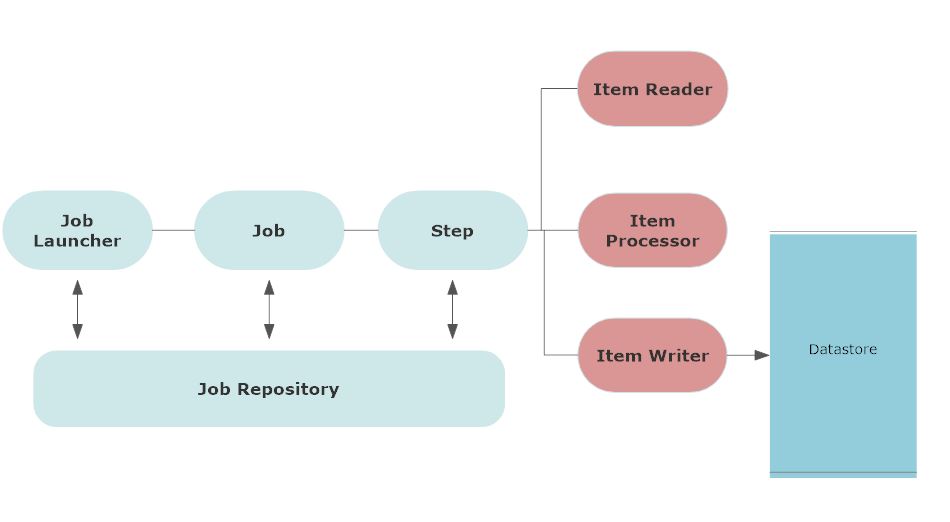
Spring Batch have 3 different layers: an Application layer, a Batch core & a Batch infrastructure layer.

1. Application layer: The application layer contains all batch jobs & code written by developers using Spring Batch i.e., where we will stitch together various Spring Batch components & then invoke them when necessary.

2. Batch Core layer: The Batch core contains the required run-time classes – Job, Step, JobLauncher etc. Here we have interfaces which enable the execution of batch processing tasks.

3. Batch Infrastructure: The Batch infrastructure includes common readers/writers and services used by the developers & the core layer.

**Spring Batch Core Components**



**1. JobLauncher:** A JobLauncher launches/starts a spring batch job at its scheduled time.

**2. Job:** A Job may have parameters & tracks its status. It is a Job which takes care of the batch process. For e.g., it is a job which may encapsulate the process of transforming an input CSV file into a collection of rows which are then written to a database table to perform all of these tasks.

**3. Step:** A Job may need to be broken up into a sequence of steps. A step can be thought of as an independent phase of a Job & can track its own status.

So, the JobLauncher, Job & Step are different objects available in the Spring Batch core & we can monitor the status of a Job & individual Step as they are being executed or also after they are done running using the corresponding objects.

* A Step, in turn, contains a number of different components of its own, enabling it to perform a batch operation & these are the **ItemReader**, **ItemProcessor** & **ItemWriter**.

**4. Job Repository:** The Job Repository keeps all the metadata with regards to Jobs, Steps & the JobLauncher.

* For e.g., this maintains records of when a Job or Step was launched, How long it took to execute & what the execution status was, after it was done running.
* We can use all of this information in order to analyze each of our batch processing applications.

**Step Components**

**1. ItemReader:**

* ItemReader reads the input data.
* It is capable of reading input data & this could be from a file or database table & this data is typically translated into Java objects (one java object for each item being read).
* So, an item may represent a single row in a CSV file or a database table or a single json object.

**2. ItemProcessor**

* ItemProcessor transforms the input data.
* Once we have different java objects for each of the input items read in, We can apply the ItemProcessor in order to transform the data.
* For e.g., Currency conversion, Date format conversion, may even split up each item object into a number of other objects.
* Out of the objects produced by the ItemProcessor will be handed over to the ItemWriter.

**3. ItemWriter**

* ItemWriter writes/loads the transformed data.
* ItemWriter is tasked with the responsibility of translating those objects to a form which can be written to the output.
* For e.g., Translating each of the objects fed in by the ItemProcessor into XML elements & then writing them to an XML file.

**Different Approaches to using Spring Batch**

1. Configurations can be setup using XML files or Java Configurations.

2. Processing can be done online & offline.

3. Multiple batch processes can run in parallel.

4. Performance can be further boosted by partitioning the input data when possible. Assuming each of the partition is independent of one another.