TGH

Making Integrations Simpler

A Guide to Batch Processing in MuleSoft

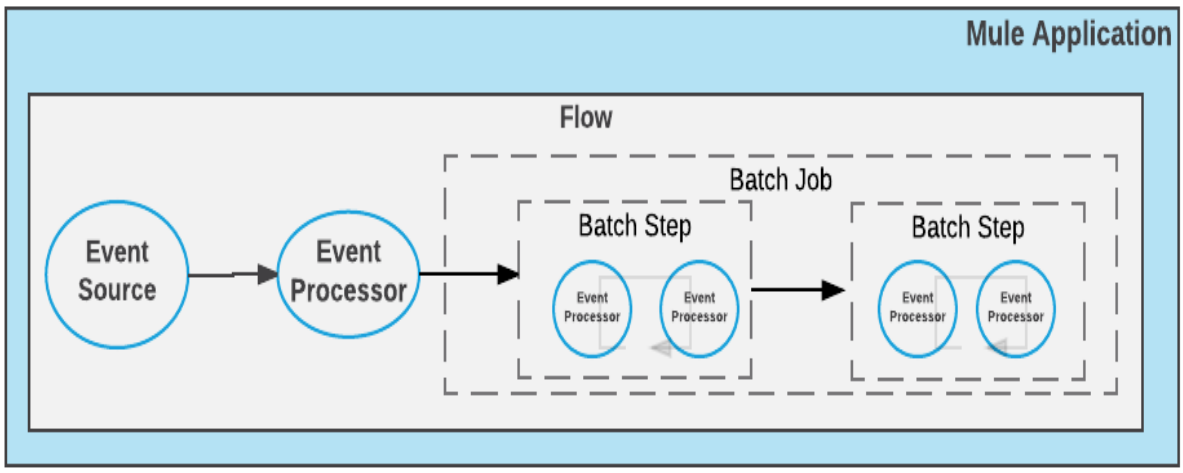
Author
Teja Dannina

Contents

Batch Processing in MuleSoft

Introduction:

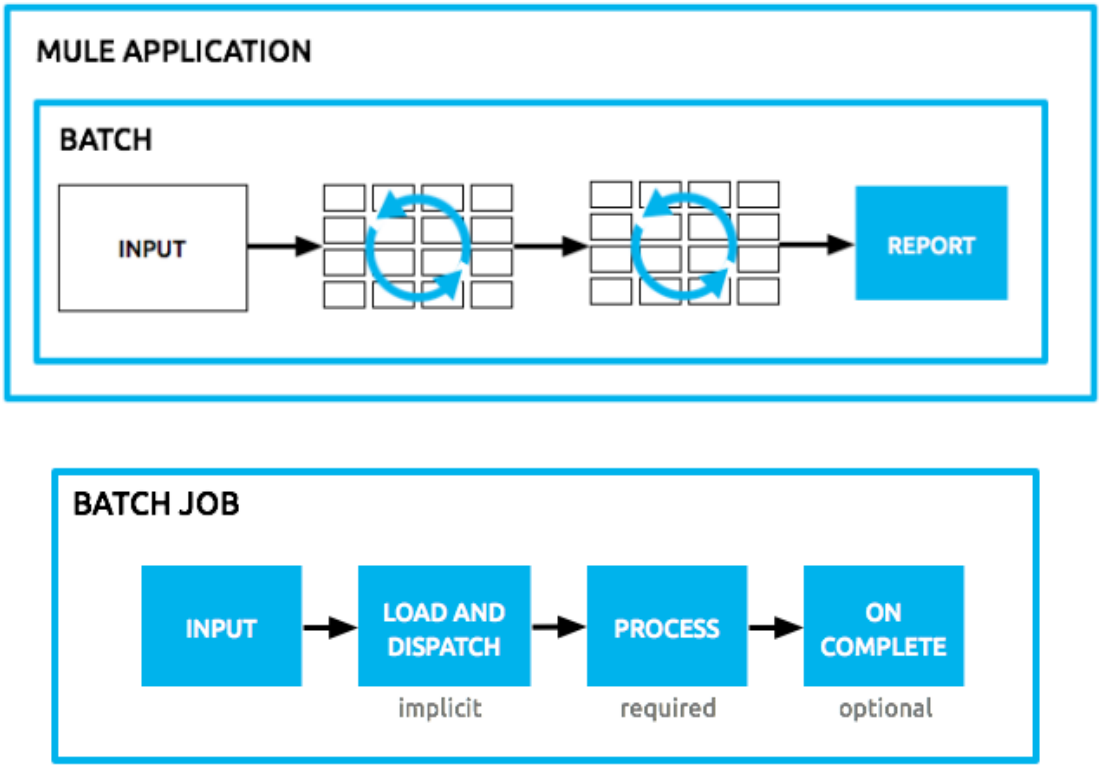
- Batch processing in MuleSoft refers to the capability of processing large volumes of data in batches rather than individually. This is particularly useful when dealing with large datasets or when interacting with systems that have limitations on the number of transactions they can handle at once.
- Mule batch processing components are designed for reliable, asynchronous processing of larger-than-memory data sets.
- this feature allows you to process large amounts of data by breaking them into individual records and processing each record asynchronously.
- Batch processing is commonly used in scenarios such as bulk data uploads, data synchronization between systems, data migration, and batch updates.
- In MuleSoft, a batch job is a sequence of steps that define the processing logic for each record in a batch. These steps typically include reading data, processing it, and optionally writing it to a destination.
- A batch job is a top-level element in Mule that exists outside all Mule flows. Batch jobs split large messages into records which Mule processes asynchronously in a batch job.
- In case of an application crash, it will resume from the same point where it stopped because the records are stored in the persistent queue.



Batch Job Lifecycle:

It has three phases: -

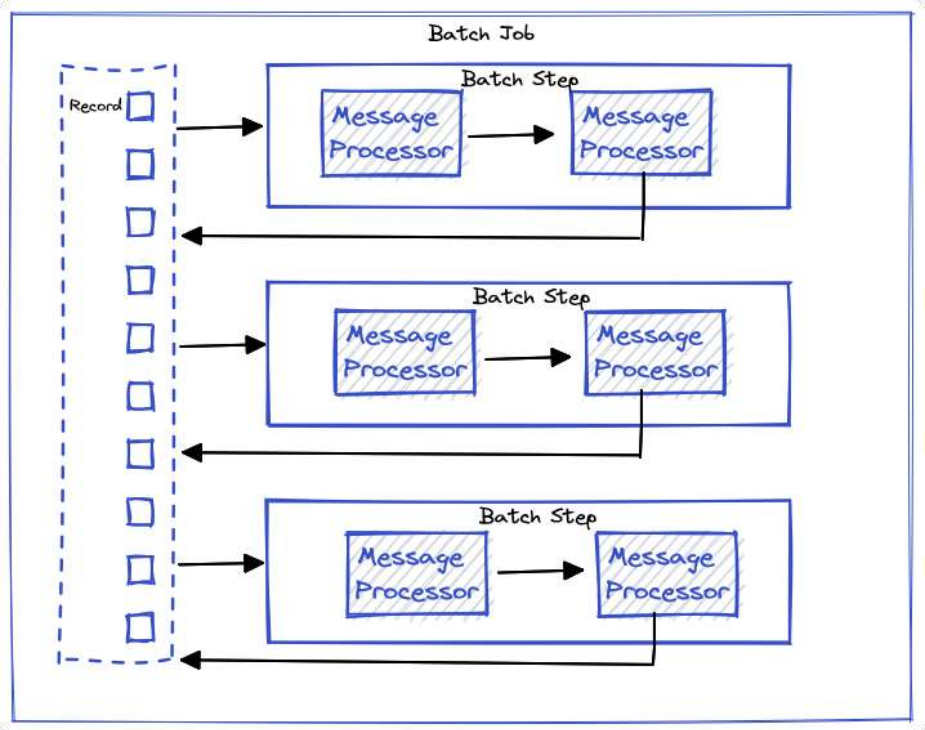
- Batch Job has three phases: -
    - Load and dispatch phase
    - Process phase
    - On complete phase



- **Load and dispatch phase**: -
- The "Load and Dispatch" phase serves as the initial step in the lifecycle of a batch job.
- When the Batch Job component receives a message in the flow, the Load and Dispatch phase begins.
- In this phase, the component prepares the input for processing as records.
- It will split the payload into a collection of records and stores in a persistent queue, and associate it with the new batch job instance.
- When a batch job is created, it is assigned a unique instance ID which is in the form of a UUID. and can be retrieved using the variable batchJobInstanceId.
- with all its queued-up records, the process phase will begin.

❖ **Process phase**: -

- All record processing takes place during this phase. Each Batch Step component contains one or more processors that act upon a record to transform, route, enrich, or modify data in the records.
- in this phase where all the records are processed asynchronously. Each record is picked from the queue by batch step and processors within the batch step are executed on the record. Once batch step processing is complete the record is again sent back to the Queue where it's picked by the next Batch Step.



❖ **On complete phase**: -

- This will execute last when all the records are processed.
- In this On Complete phase, it creates a report, using MEL (Mule Expression Language) expressions to capture the number of failed records and successfully processed records, and in which step any errors might have occurred.

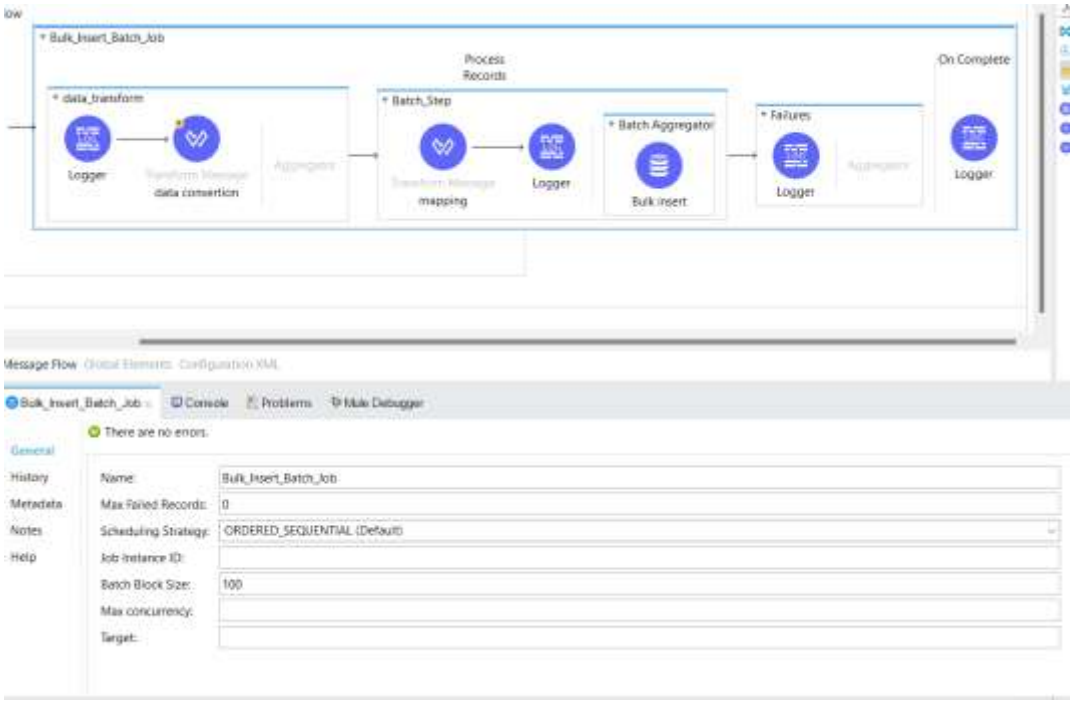
Batch Job Components: -

- The components are the Batch Job, Batch Step, and Batch Aggregator.

❖ **Batch Job**: -

- If we drag and drop any Batch job activity then it will automatically create a flow for the batch job.

Configuration: -



Name: - Batch job name

Max Failed Records: - this value defines how many failed records we should stop the Batch job processing. We can configure this by: -

- 0(Default): - it will stop processing when a failed record is found
- -1: - it will stop processing when a failed record is found
- >0: - it will continue processing until reaching the maximum number of failed records and if it reaches the maximum number, it will stop processing.

Note: - When a batch job accumulates enough failed records to cross the maxFailedRecords threshold, Mule aborts processing for any remaining batch steps, skipping directly to the On Complete phase.

Scheduling Strategy: - Batch jobs execution

- ORDERED_SQUENTIAL (the default): If several job instances are in an executable state at the same time, the instances execute one at a time based on their creation timestamp.
- ROUND_ROBIN: This setting attempts to execute all available instances of a batch job using a round-robin algorithm to assign the available resources.

Job Instance ID: - We can use a DataWeave expression to specify a unique identifier for each batch job by passing a Job Instance Id attribute that takes a DW expression. Note that constant values are not allowed for batch jobs and if the DW expression returns a previously seen value, Mule throws an exception and does not create the job instance. If you don't set a job instance ID, Mule auto-generates a UUID to assign to your instance.

Batch Block Size: - Block size defines the number of records given to each thread for execution. (default 100)

Max Concurrency: -Defining the number of threads that can participate in batch processing if you don't want the default thread count which depends on vCore size

Target: - To save Batch Stat Records

❖ **Batch Step**: -

- We can have multiple batch steps.
- Each batch step can contain a related processor to work on individual records for any type of processing like enrichment, transformation, or routing. This will help in segregating the processing logic
- We can enhance the Batch Step processing by providing the filter criteria to limit the processing to eligible records only
- A Batch Step component that uses an acceptExpression attribute applies a DataWeave expression to each record that reaches the component and accepts a record for processing within the component only if the expression evaluates to true for that record. If the record evaluates to false, the Batch Step component skips the record, and that record becomes available to the next Batch Step component within the Batch Job component if one exists.

Configuration: -

Name: - Batch Step name

Accept Expression: - An expression that remains valid for processing records (Optional)

Accept Policy: - It can only contain predefined values below.

- NO_FAILURES (Default): - the batch step processes only those records that have successfully passed through every previous processing step.
- ONLY_FAILURES: - Batch step processes only those records that encountered failures during the previous batch step.
- ALL: - The batch step processes every record, irrespective of whether it failed to process in a previous batch step.

Batch Step can contain zero or a maximum of one Batch Aggregator.

❖ **Batch Aggregator**: -

➢ Each batch step has a section called aggregation which allows us to load/upsert data in bulk to databases or external data sources.

➢ Batch Aggregation is useful for sending multiple records within an array to an external server.

➢ Within the Batch Aggregator component, you can add an operation, such as a bulk upsert, insert, or update operation, to load multiple records to a server with a single execution of an operation, instead of running an operation separately on each record.

Configuration: -

Name: - Batch Aggregator name

Aggregator Size: Size of the aggregator.

- if the size of the Batch Aggregator is set to 10, then the payload will be a collection with 10 records. For every 10 records, the processors inside the Batch Aggregator will execute.

Streaming: - It enables handling an array containing all records within the batch job instance without worrying about memory limitations, regardless of the number or size of the records.

Preserve Mime Type: - the MIME types associated with those payloads are not preserved. To preserve the record's MIME types, specify the preserveMimeTypes attribute in the Batch Aggregator component.

❖ **On complete**: -

- the On Complete scope is a part of the Batch Job that executes after all the batch steps have processed all the incoming records.
- This scope is optional and is primarily used to provide insights or reports of batch processing.
- For example, it can provide information about the total number of records processed, how many failed, and how many were successfully processed.
- If we convert that report into JSON it looks like this: -

```
3ea64f10-e513-11ee-ab43-4ae7da430a29] org.mule.runtime.core.internal.processor.LoggerMessageProcessor: {
  "onCompletePhaseException": null,
  "loadingPhaseException": null,
  "totalRecords": 100,
  "elapsedTimeInMillis": 5282,
  "failedOnCompletePhase": false,
  "failedRecords": 10,
  "loadedRecords": 100,
  "failedOnInputPhase": false,
  "successfulRecords": 90,
  "inputPhaseException": null,
  "processedRecords": 100,
  "failedOnLoadingPhase": false,
  "batchJobInstanceId": "3ee0c000-e513-11ee-ab43-4ae7da430a29"
}
```

TGH

Making Integrations Simpler

TGH Software Solutions Pvt. Ltd.

www.techygeekhub.com

At TGH, we specialize in driving digital transformation through seamless Integration Technologies.

Operating as an INTEGRATION FACTORY, we serve as a one-stop shop for all your integration needs. Our expert team is well-versed in enterprise software and legacy system integration, along with leading iPaaS technologies like Boomi, MuleSoft, Workato, OIC, and more.

We're committed to enhancing business processes and solving problems through our integration expertise.

Email address

connect@techygeekhub.com

Phone number

+ 011-40071137

+ 91-8810610395

Our offices

Noida Office

iThum
Plot No -40, Tower A,
Office No: 712,
Sector-62, Noida,
Uttar Pradesh, 201301

Hyderabad Office

Plot no: 6/3, 5th Floor,
Techno Pearl Building,
HUDA Techno Enclave,
HITEC City, Hyderabad,
Telangana 500081