International Journal for Research in Engineering Application & Management (IJREAM)
ISSN : 2454-9150 Vol-07, Issue-05, AUG 2021

Integration of Salesforce through MuleSoft Using Batch Processing

[*]Prakruthi P S, [#]Sunil Kumar R

[*,#]MuleSoft Certified Developer – Software Engineer, Apisero, Inc. Bangalore & India,

[*]prakruthishekar29@gmail.com, [#]sunilrudrakumar@gmail.com

Abstract: MuleSoft provides the ability to batch process messages. It breaks down big messages into individual records, which are then handled asynchronously in batch tasks. Large volumes of incoming data from any upstream system may be extracted, transformed, and loaded (ETL) into any destination system in real time using the batch method. Facebook Graph API is utilized as the upstream system in this article (The Facebook Graph API is the primary way for apps to read and write to the Facebook social graph), while Salesforce is used extensively as the destination system. Salesforce is a cloud computing platform that uses data objects to store data. This article discusses the problems that arise when upstream systems have complicated data storage formats, as well as the transformations that are required to accomplish efficient data transfers. Batch Processing may be used to combine big or small databases and process records in a parallel manner. Additionally, individual records can have variables added or removed so that Mule can route or act on records in a batch based on a record variable during batch processing. This article addresses numerous components that are highly unique to batch processing and may be utilized to build business logic, as well as certain generic situations that form the core of every batch flow, to assist offer a better understanding. Uses Cases where up to 6 million entries were successfully fetched from a database, converted, and upserted to Salesforce, as well as suitable error handling methods, are presented.

Keywords — MuleSoft, Batch Processing, Error Handling, Salesforce, ETL, Object Store, Send Module

I. INTRODUCTION

Data-driven decision-making is extremely important in today's organizations as it is the most important aspect of the organization. Data-driven decision making is critical because it allows us to analyze real - time data and derive predictive insights. It enables you to conduct research and determine what is or is not working for your company.

Organizations must gather data from a variety of sources and change it into a format that can be used for downstream analysis. Given the range of data repositories and complex formats used in businesses, the design of systems that can achieve these goals can quickly become complicated to build and maintain.

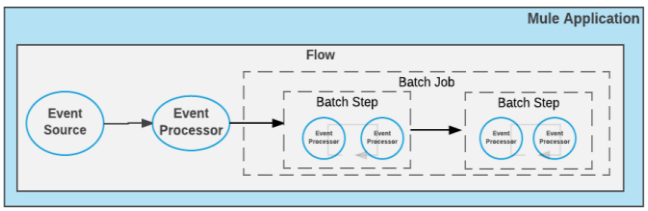
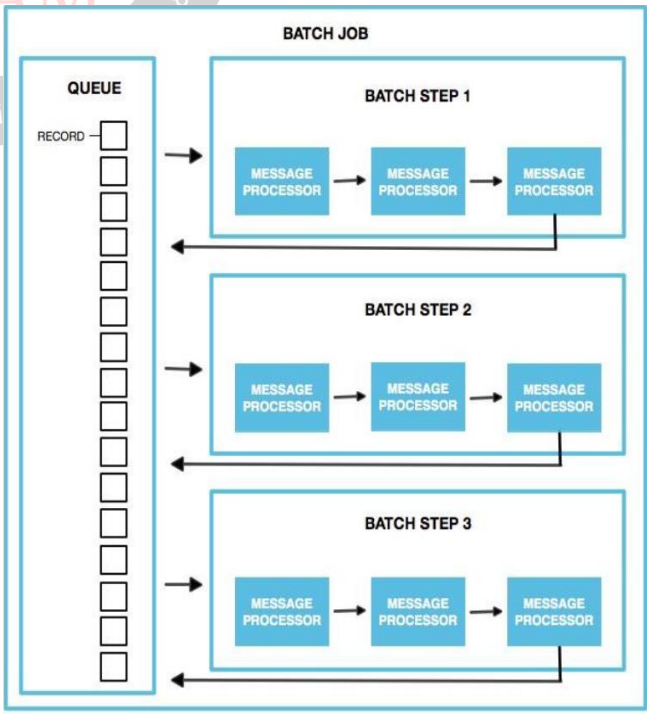


Figure 1 shows the Architecture of batch job.

It's usually tough to process each individual record separately when loading vast amounts of data from one system to another. As a result, processing data in batches is always advised; most systems also support this.



Figures 2 shows how MuleSoft's batch process phase works

 DOI : 10.35291/2454-9150.2021.0456

MuleSoft, on the other hand, has a number of built-in interfaces that help reduce time to market by allowing batch processes to run asynchronously and in multithreading mode. We hope to address this complicated data processing mechanism in this work, as well as rapid real-time procedures for transmitting millions of entries. This paper also includes a generic method for dealing with errored records that may be applied to any batch flow.

II. Batch Job Processing Phases

Each batch job contains three different phases:

1. Load and Dispatch.
2. Process.
3. On Complete.

Load and Dispatch: This initial stage is self-evident. The runtime does all of the "behind the scenes" work to establish a batch job instance at this step. This is the stage at which Mule converts a serialized message payload into a collection of records that can be processed in a batch and it does not require any configuration, however it is helpful to understand the tasks Mule completes during this phase.

1.Mule uses Data weave to separate the message. A new batch task instance is created in the first step. The batchJobInstanceId variable in Mule exposes the batch job instance ID. This variable is present in each step as well as the on-complete phase.

2.Mule produces a record for each item generated by the splitter and places it in the queue. This activity is "all or nothing" – Mule either generates and queues a record for each item correctly, or the entire message fails during this phase.

3.Mule sends the batch job instance to the first batch step for processing, along with all of its queued records.

After this phase is over, the flow continues without having to wait for the batch job to finish processing all of the records. Because the next phase, Process, is asynchronous, this behavior occurs.

Process: Mule completes the following steps during this phase.

1.Mule begins extracting records from the stepping queue during the Process phase to create record blocks of the configured batch block size. Mule then delivers the record blocks to the batch step that corresponds to them and processes them asynchronously.

2.Each batch step processes numerous record blocks in parallel, but the records within each block are processed sequentially.

After processing all of the data in a block, a batch step transfers the records to the stepping queue, where they await processing by the next batch step (each record keeps track of the steps it completed).

This process continues until every record in the batch job instance has passed through all of the batch steps.

It's worth noting that a batch job instance doesn't wait for all of its queued records to complete processing in one batch step before moving on to the next. If you configure an aggregator, however, the batch job step's behavior while processing records changes depending on how the aggregator is configured.

On Complete: There are two ways to work with the output.

1. Create a report using Data Weave in the On Complete phase, including the number of failed records and successfully processed records, as well as where any mistakes occurred.
2. To collect and use batch metadata, such as the number of records that failed to process in a given batch job instance, use the batch job result object elsewhere in the Mule application.

If you leave the On Complete phase empty and do not reference the batch job result object elsewhere in your application, the batch job simply completes, whether failed or successful.

The following parameters for the batch connector should be determined based on the input and its size:

1.Block Size:

To figure out how big the block that will be used for the job instances should be. Instead of queuing each record separately, they are queued in blocks of 100 records, which reduces I/O overhead but increases working memory requirements. There is a block: 1 relationship between each record and a working thread instead of a 1:1 relationship. Each job has 16 threads by default (this can be changed using threading-profile element)

- Each of these threads is assigned a 100-record block.
- Each thread goes over the block one by one, processing each record.
- Each block is returned to the queue, and the process continues.
- Default value: 100

2.Max Failed Records:

- Value 0 accepts no errors and instantly stops the batch operation.

 DOI : 10.35291/2454-9150.2021.0456

- Value -1 accepts all failures and never halts processing due to a failed record.
- Any integer value specifies the maximum number of unsuccessful records that a batch can process before ending.

3.Batch Step Filters:

Accept Expressions limits the number of records that can be processed depending on a condition.

Accept the following policies for each step:

- ALL takes all records
- NO_FAILURES takes only successful records
- FAILURES_ONLY takes only failed records

4. Batch Commit:

The number of records that can be committed to the destination at one time.

- **Fixed-sized commit:**

Commit blocks are used to combine a bunch of records together so that bulk activities can be performed. Inserting records one by one would be inefficient due to I/O costs (network overhead, disc overhead, etc.).

As a result, it's preferable to use batch commit to conduct only one bulk operation that aggregates a subset of records within a batch.

a) Salesforce Upsert Operation: Max value 200.

b) Salesforce Bulk Upsert Operation: Max value 10,000.

- **Streaming commit:**

The streaming feature – one-read, forward-only iterator is returned – assures that all records received are the records in a batch job without running out of memory, rather than a list of elements received with a fixed-size batch commit.

III. USE CASES

a) Loading large amount of data:

We're pulling a huge number of records from the Graph API, converting them to Salesforce format, then upserting them with a batch commit size of 200.

Values are kept in properties file as-

- ${batch.failedRecords} : -1
- ${batch.size} : 200

There are two upsert possibilities in Salesforce:

(Shown in Fig3)

- **Upsert:** a maximum of 200 records can be pushed to a salesforce object at once.
- **Bulk Upsert:** a maximum of 10,000 records are provided to be upserted.

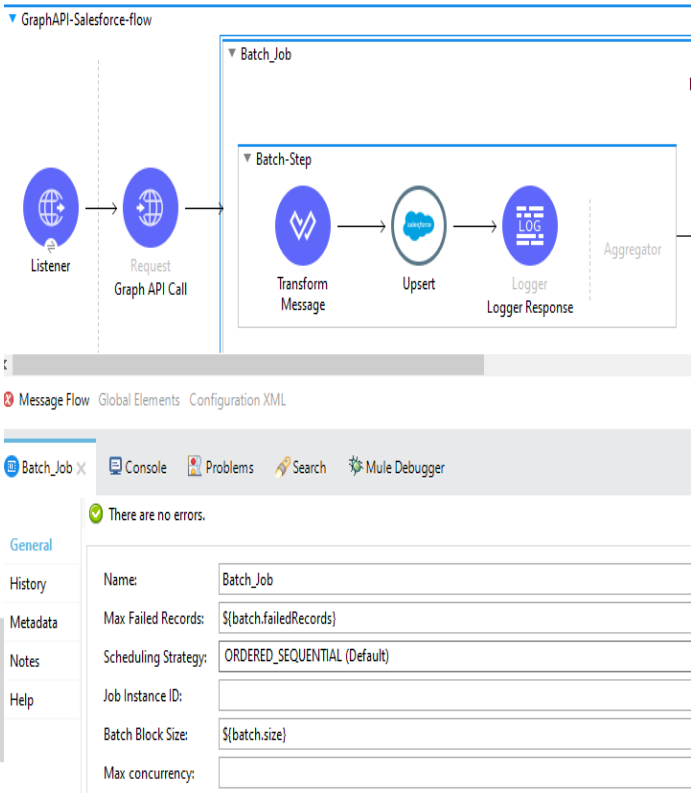


Figure 3 shows the batch job configuration.

Input Data :


```
{
    "id": "1079",
    "name": "Pre-cabling",
    "action_number": 19422,
    "owner": "Harish Sharma",
    "status": "Closed",
    "locations": [
        "DEN3"
    ],
    "start_date": "2021-03-19T07:00:00+0000",
    "end_date": "2021-03-22T07:00:00+0000",
    "deployment_engineer": "Igor Assis",
    "project_coordinator": "Grace Jack",
    "project_manager": "Grace Jack",
    "project_name": "AMER H1 2021",
},
{
    "id": "1117",
    "name": "Execute Fallouts",
    "action_number": 16077,
    "owner": "James",
    "status": "Closed",
    "locations": [
        "SJC1"
    ],
    "start_date": "2020-12-10T08:00:00+0000",
    "end_date": "2020-12-11T08:00:00+0000",
    "project_coordinator": "Mohamed Feroz",
    "project_manager": "Nimisha",
    "project_name": "AMER ",
},
{
    "id": "112429174256573",
    "name": "Raise Site Access - Audit Lite",
    "action_number": 22310,
    "owner": "Lou Noble",
    "status": "Closed",
    "locations": [
        "SEA1 70500"
    ],
    "start_date": "2021-05-19T07:00:00+0000",
```


Figure 4 shows Example response from Graph API.

Response from Salesforce in logger:

 DOI : 10.35291/2454-9150.2021.0456

```
{
  "id": null,
  "items": [
    {
      "exception": null,
      "message": null,
      "payload": {
        "success": true,
        "id": "aCk170000004p5pCAA",
        "errors": [

        ]
      },
      "id": "aCk170000004p5pCAA",
      "statusCode": null,
      "successful": true
    }
  ],
  "successful": true
}
```


Figure 5 shows Example Response from Salesforce after upserting a record.

Result after the execution in batch job in On-Complete phase:


```
{
"onCompletePhaseException":null,
   "loadingPhaseException":null,
   "totalRecords": 60,00,000,
   "elapsedTimeInMillis":69000,
   "failedOnCompletePhase":false,
   "failedRecords":0,
   "loadedRecords": 60,00,000,
   "failedOnInputPhase":false,
   "successfulRecords": 60,00,000,
"inputPhaseException":null,
   "processedRecords": 60,00,000,
   "failedOnLoadingPhase":false,

"batchJobInstanceId":"6ega3c10-a62a-11da-8a1a-9aaf65ed66
d8"
```


Figure 6 shows sample response after completion of batch job

This flow upserts records in Salesforce and logs the response that is sent by salesforce as the success response which will have Salesforce ID for the records created.

b) Handling failed records:

Accept policy as "ONLY FAILURES" in batch step failed records configuration settings to know all records that have failed in previous batch stages. This will just show you the records that failed, not the actual error trail. The actual error trail will be saved in log files that can be examined later.

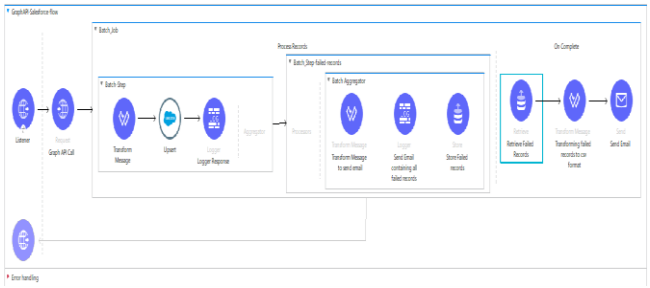


Figure 7 shows overall mule flow.

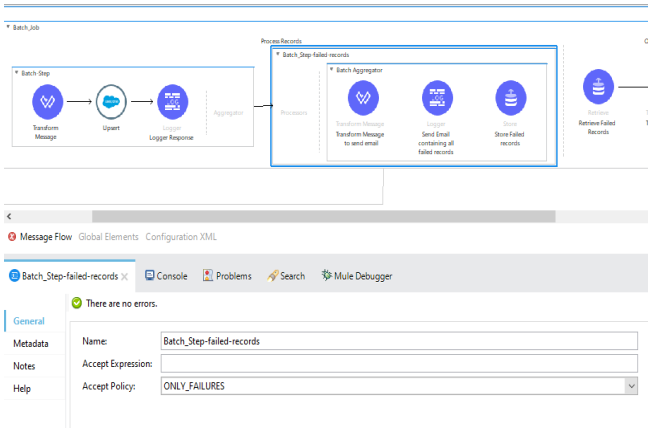


Figure 8 shows Configuration of batch step for failed records

- All the failed records payload will be fetched in the seconds batch step which is Batch Step-failed-records.
- The variables or payload of batch step are not accessible in the on complete phase of batch Job. Here comes the need of using Object store.

In Mule, an object store is a location where objects can be stored. When Mule needs data to be saved for later retrieval, it employs object stores. This notion can be used to save data in the object store and retrieve it later in another batch step, batch phase, or even in another flow.

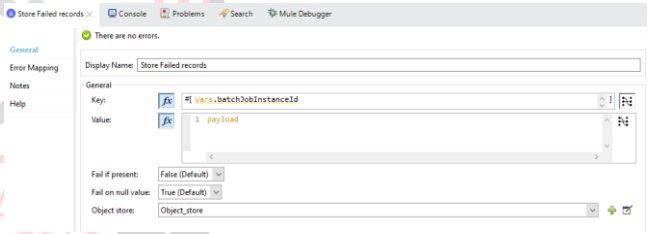


Figure 9 shows Object store configuration

Object store expects a unique key to be configured. In this case we're using BatchJobInstanceId as a unique identifier and the value is going to the failed records.

In the on complete phase retrieve module of object store connector is used to retrieve the stored failed records by taking the key.

Mule provides Email (Email Connector) to send messages over SMTP and SMTPS servers by using the Send operation. Along with the batch summary which we get in on complete phase additional attachment can be sent to the concerned person over an email as an alert if there are any failed records present.

IV. RESULTS

No of records: 6 million

Input Phase: 0 minute 2 seconds.

Load phase: 0 minute 43 seconds

 DOI : 10.35291/2454-9150.2021.0456

Process Phase: 1 minute 9 seconds

Limitation: Only 1 million records stored due to Salesforce sandbox storage limit

V. CONCLUSION

MuleSoft allows you to process messages in batches using the batch scope feature. A mule application's batch scope can split the input payload into individual records, conduct actions on those records, and then transmit the processed data to destination systems.

This paper explains the MuleSoft batch processing components in detail, including how concepts like Batch step, object stores, send email, Salesforce Connector, and others may assist in creating efficient techniques for transforming and loading massive datasets into systems like Salesforce. Because Salesforce uses complicated systems to store data, data integrity and error handling are critical. It took around a minute to load and process 6 million records. MuleSoft Batch tasks are designed for high-throughput, real-time processing of millions of records, and similar techniques can be used to any other upstream and downstream platforms not mentioned in this document.

REFERENCES

[1]. Batch Processing - Mule Runtime - MuleSoft Documentation https://docs.mulesoft.com/mule-runtime/4.3/batch-processing-concept

[2]. ERROR HANDLING IN BATCH JOB https://mulesy.com/error-handling-in-batch-job/

[3]. Batch Processing in Mule 4 - Apisero https://apisero.com/batch-processing-in-mule-4/#:~:text=Batch%20scope%20in%20a%20mule,16%20threads%20at%20a%20time.

[4]. Mule Batch Processing - Part 1: Introduction - DZone Integration https://dzone.com/articles/part-1-mule-batch-processing-introduction

[5]. Part 1: Mule Batch Processing - Introduction - {Java} Streetshttps://javastreets.com/blog/2017/9/mule-batch-processing-part1-introduction.html.

[6]. Mule ESb - https://www.tutorialspoint.com/mulesoft/mulesoft_introduction_to_mule_esb.htm

[7]. Salesforce connector - https://mulesy.com/create-records-in-salesforce/

[8]. Object Store - https://vanchiv.com/what-is-object-store-in-mule-4/

[9]. Mule Flow - https://www.baeldung.com/mule-esb-flows#:~:text=There%20are%20three%20different%20types,strategy%20of%20the%20parent%20flow&text=Asynchronous%20Flows%20%E2%80%93%20an%20asynchronous%20flow%20with%20its%20processing%20and%20exception%20handling%20strategy

[10]. Anypoint Studio - https://dzone.com/articles/introduction-to-mulesoft-anypoint-studio

[11]. Mariano Gonzalez, Tech Lead in the core Mule Runtime; https://blogs.mulesoft.com/dev/connectivity-dev/batch-improvements-in-mule-3-8-mutable-commit-blocks/

 DOI : 10.35291/2454-9150.2021.0456