MuleSoft from Salesforce

Mumbai MuleSoft Meetup (In Collaboration with Guwahati Meetup Group)

MuleSoft Training for Salesforce Developers and Beginners - Module 6

Date: 22nd Nov 2025
Time: 11 AM to 1 PM

Safe Harbour Statement

MuleSoft from Salesforce

- Both the speaker and the host are organizing this meet-up in individual capacity only. We are not representing our companies here.

- This presentation is strictly for learning purposes only. Organizer/Presenter do not hold any responsibility that same solution will work for your business requirements.

- This presentation is not meant for any promotional activities.

Housekeeping

A recording of this meetup will be uploaded to events page within 24 hours.

Questions can be submitted/asked at any time in the Chat/Questions & Answers Tab.

Make it more **Interactive!!!**

Give us feedback! Rate this meetup session by filling feedback form at the end of the day.

We Love Feedbacks!!! Its Bread & Butter for Meetup**.**

Organizers/Speakers

Jitendra Bafna

Senior Solution Architect
EPAM Systems

Moderators

Jitendra Bafna

Senior Solution Architect
EPAM Systems

Abhishek Bathwal
Technical Architect
NeuraFlash

What will we cover in Training?

Module	Topic	What will we cover?	Date
Module 1	Integration & REST/HTTP Basics for Beginners & Salesforce	Integration, P2P, REST APIs, MuleSoft, Anypoint Platform	1st Nov 2025
Module 2	API Design with RAML for Beginners & Salesforce Developers	API Design with RAML, Publishing APIs to Exchange, Resources	2nd Nov 2025
Module 3	Anypoint Studio & Mule Basics for Beginners & Salesforce	API Implementation and Deploying API to CloudHub	8th Nov 2025
Module 4	Core Components, DataWeave & Error Handling Essentials	Dataweave, Error Handling, Core Components	9th Nov 2025
Module 5	Flow Control & Batch Processing for Scalable Integrations	Batch Processing, For Each, Parallel For Each	15th Nov 2025
Module 6	HTTP Connector – Listener, Requestor & Payload Handling	HTTP Connector, OAuth Module	16th Nov 2025
Module 7	Database Connector for CRUD Operations	Database connector to perform query and call store procedure	22nd Nov 2025

What will we cover in Training?

Module	Topic	What will we cover?	Date
Module 8	Salesforce Connector for Seamless CRM Integration	Deep Dive into Salesforce Connector	23rd Nov 2025
Module 9	Hosting Options & Deploying Applications to CloudHub	ClodHub 1.0 and CloudHub 2.0	6th Dec 2025
Module 10	Managing & Securing APIs with API Manager & API Gateway	API Security, API Policies	7th Dec 2025
Module 11	MuleSoft Demo Project	Database and Salesforce related project	13th Dec 2025

What have we learned on Day 1?

- What is Point-To-Point Integration?
- What is Integration?
- What is REST APIs?
- What is MuleSoft and Anypoint Platform?
- Walkthrough of Anypoint Platform.
- Understanding the API Lifecycle Management.
- Design the RAML to create and fetch Account and Contacts from Salesforce.
- Published API to Anypoint Exchange.

What have we learned on Day 2?

- What is RAML?
- Reusability of RAML using Traits, Library, Security Schemes.
- OAS (Open API Specification)
- API Governance
- Overview of Anypoint Studio
- Start with API Implementation.

What have we learned on Day 3?

- What Are Connectors
- Salesforce Connector
- Implementing Salesforce System API
- Implementing Properties and Secure Properties
- Deploying MuleSoft Application to CloudHub

What have we learned on Day 4?

- Understanding API Manager and API Gateway
- Creating API Proxy
- Implementing API Auto-Discovery
- Applying Basic Authentication Policy
- Applying Client ID Enforcement Policy

What have we learned on Day 5?

- MuleSoft Batch Processing
- For Each
- Parallel For Each
- Implementing Process API

What will we learn on Day 6?

- MuleSoft Batch Processing
- Error Handling in Batch Processing
- Scatter Gather
- Orchestrating Process API

Batch Processing

What is Batch Processing

Batch processing in MuleSoft is a way to handle many items (records) one by one in an efficient and organized way.

MuleSoft automatically:

- Takes your big data list
- Breaks it into individual records
- Processes each record in small groups (called batches)
- Finishes all of them even if some fail

It helps when you have lots of data (thousands or millions).

Batch Job Phases

1. Input Phase

- 🔷 Mule takes the big payload (e.g., list of 10,000 records).
- 🔷 It splits the payload into individual records.

Example:

- A list of 100 customers → Mule creates 100 records.

2. Load and Dispatch Phase

- 🔷 Mule organizes the records into groups/chunks.
- 🔷 These groups are sent to workers/threads for processing.

Example:

- It may create batches of 100 records at a time and send them to be processed in parallel.

Batch Job Phases

3. Process Phase

- 🔷 This is where your batch steps run.
- 🔷 Each record goes through the steps one by one.

Example steps:

- Step 1 → Validate record
- Step 2 → Transform record
- Step 3 → Save to database

Each record moves through these steps.

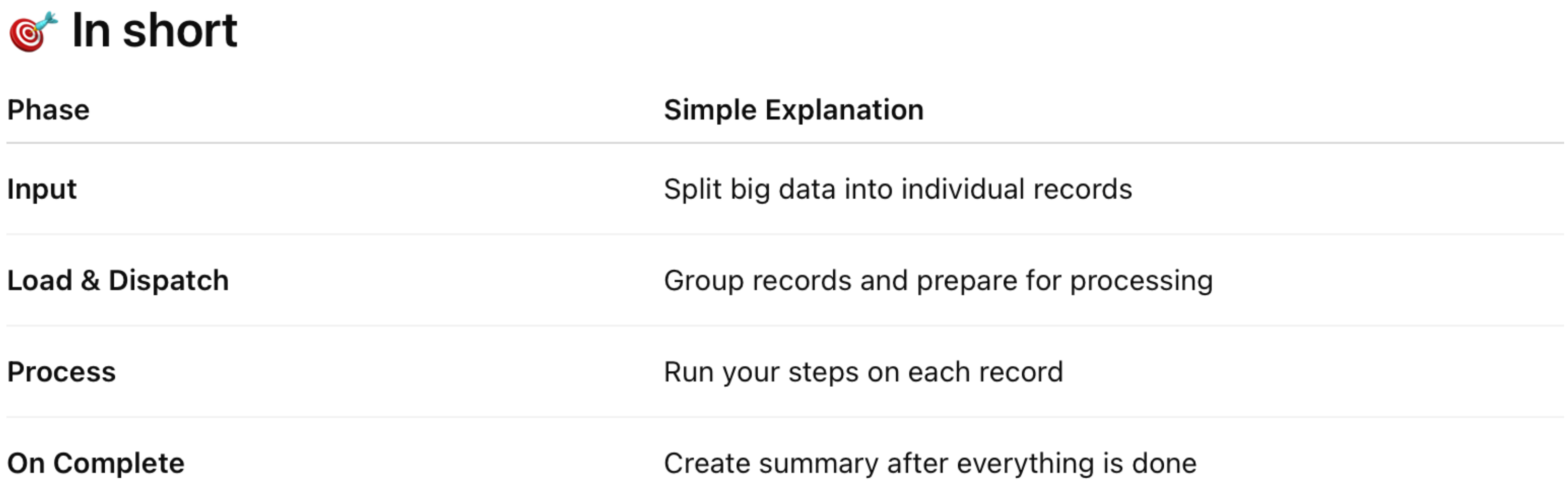
4. On Complete Phase

- 🔷 Runs after all records are processed.
- 🔷 Used to summarize results.

Example:

“Out of 100 records: 90 succeeded, 10 failed.”

Batch Job Phases



🎯 In short

Phase	Simple Explanation
Input	Split big data into individual records
Load & Dispatch	Group records and prepare for processing
Process	Run your steps on each record
On Complete	Create summary after everything is done

Key Batch Processing Components in MuleSoft

1️⃣ Batch Job

This is the main container for everything related to batch processing.

It defines:

- The batch name
- What steps should run
- How records are processed
- Error handling behavior

👉 Think of it as the box that holds the entire batch process.

2️⃣ Batch Job Input

This is part of the batch job where Mule:

- Accepts the incoming payload
- Splits it into records (one record = one item to process)

👉 Example: If the input is a list of 500 users, Mule creates 500 records.

Key Batch Processing Components in MuleSoft

3️⃣ Batch Steps

These are individual blocks of logic that apply to each record.

You can have multiple steps, such as:

- Validate record
- Transform record
- Call API
- Insert into database

Each record goes through the steps in order.

Final Recommended Approach (Simple and Accurate)

Step 1: Measure Avg Record Size

Small (<5 KB), Medium (5–20 KB), Large (>20 KB)

Step 2: Identify CPU cores

E.g., 4 vCores → expect 4–8 concurrency

Step 3: Choose batch size

Based on record size.

Step 4: Load test

Start with:

batch size = 100

max concurrency = CPU cores

Adjust based on memory/CPU usage.

Key Batch Processing Components in MuleSoft

4️⃣ Record

A record is a single item from the original input list.

Examples:

- One customer
- One order
- One product

Every record has:

- Its own data
- Its own status (success or fail)

👉 This helps ensure that one failed record does NOT stop the whole batch.

Key Batch Processing Components in MuleSoft

5️⃣ Batch Aggregator (Optional)

This is used inside a batch step when you want to:

- Combine multiple records
- Process records together as a group

Example use case:

Insert 100 records at a time into a database → better performance.

👉 Think of it as a grouping tool for batch records.

Key Batch Processing Components in MuleSoft

5️⃣ Batch Aggregator (Optional)

This is used inside a batch step when you want to:

- Combine multiple records
- Process records together as a group

Example use case:

Insert 100 records at a time into a database → better performance.

👉 Think of it as a grouping tool for batch records.

Key Batch Processing Components in MuleSoft

6️⃣ On Complete Phase

Runs after all records finish.

Used for:

- Summary logs
- Sending notification emails
- Cleanup
- Final reporting

👉 Think of it as the "wrap up" part of the job.

Key Factors That Affect Batch Size & Concurrency

A. Number of records

- More records → smaller batch size + higher concurrency
- Fewer records → larger batch size + lower concurrency

B. Record size (payload size)

- Large records (e.g., 100 KB each) → smaller batch size
- Small records (e.g., < 5 KB each) → larger batch size

C. Available CPU cores

- Concurrency should not exceed CPU cores
  (E.g., 8 cores → concurrency between 4–8)

D. Available heap memory

- Larger batch size = more memory used
- Small heap → smaller batch size

How MuleSoft Actually Processes Batches

- Each batch job instance is broken into "Batch Blocks" (default 100 records per block).
- Each block is processed using a thread from the Batch Engine.
- Max concurrency = number of threads that process blocks in parallel.

How MuleSoft Actually Processes Batches

- Each batch job instance is broken into "Batch Blocks" (default 100 records per block).
- Each block is processed using a thread from the Batch Engine.
- Max concurrency = number of threads that process blocks in parallel.

How MuleSoft Actually Processes Batches

MuleSoft from Salesforce

Batch Size (Number of records per block)

A simple rule:

Batch Size = Total Records / (CPU Cores × 2)

Example:

- 10,000 records
- 4 CPU cores
- Batch size = 10000 / (4×2) = 1250 per block

But MuleSoft default block size is 100 records, which is usually optimal.

! In practice, you rarely change block size unless you need high throughput for small records.

Max Concurrency (Batch Threads)

Max Concurrency ≈ 2 * CPU Cores

Recommended Settings Based on Payload Size

🚦 **A. Small records (< 5 KB each)**

- ○ Default batch size (100) is good
- ○ Max concurrency: 4–8 threads per vCore

✔ Works well for API integrations, simple DB loads.

🚦 **B. Medium records (5–20 KB each)**

- ○ Reduce batch size: 50–100
- ○ Max concurrency: 2–4 threads per vCore

✔ Best for medium-sized JSON objects, CSV rows.

Recommended Settings Based on Payload Size

🚦 A. Small records (< 5 KB each)

- ○ Default batch size (100) is good
- ○ Max concurrency: 4–8 threads per vCore

✔ Works well for API integrations, simple DB loads.

🚦 B. Medium records (5–20 KB each)

- ○ Reduce batch size: 50–100
- ○ Max concurrency: 2–4 threads per vCore

✔ Best for medium-sized JSON objects, CSV rows.

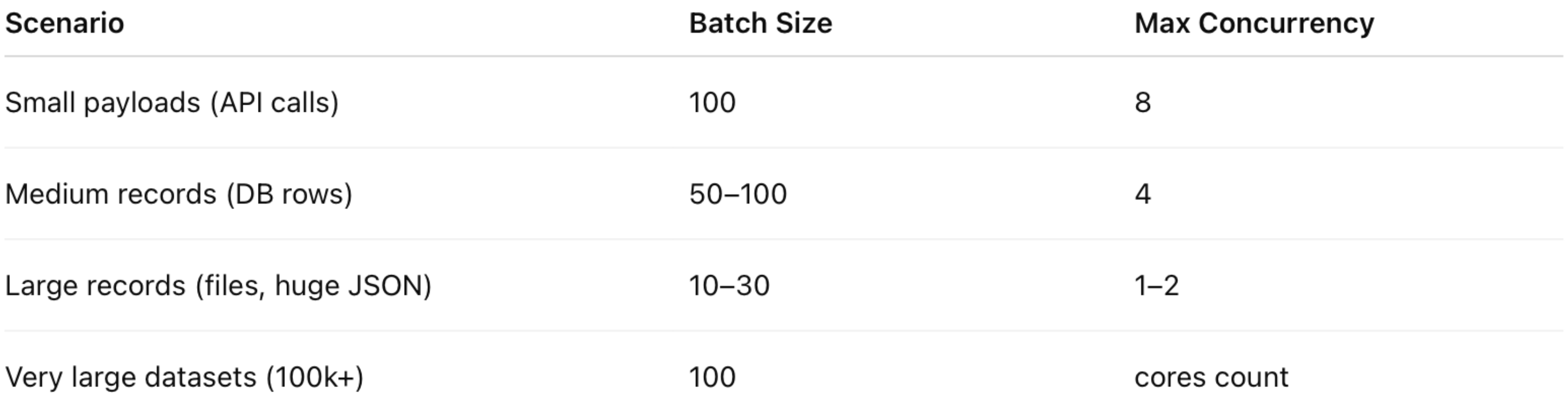
🚦 C. Large records (> 20 KB each)

Reduce batch size: 10–50

Reduce concurrency: 1–2 threads per vCore

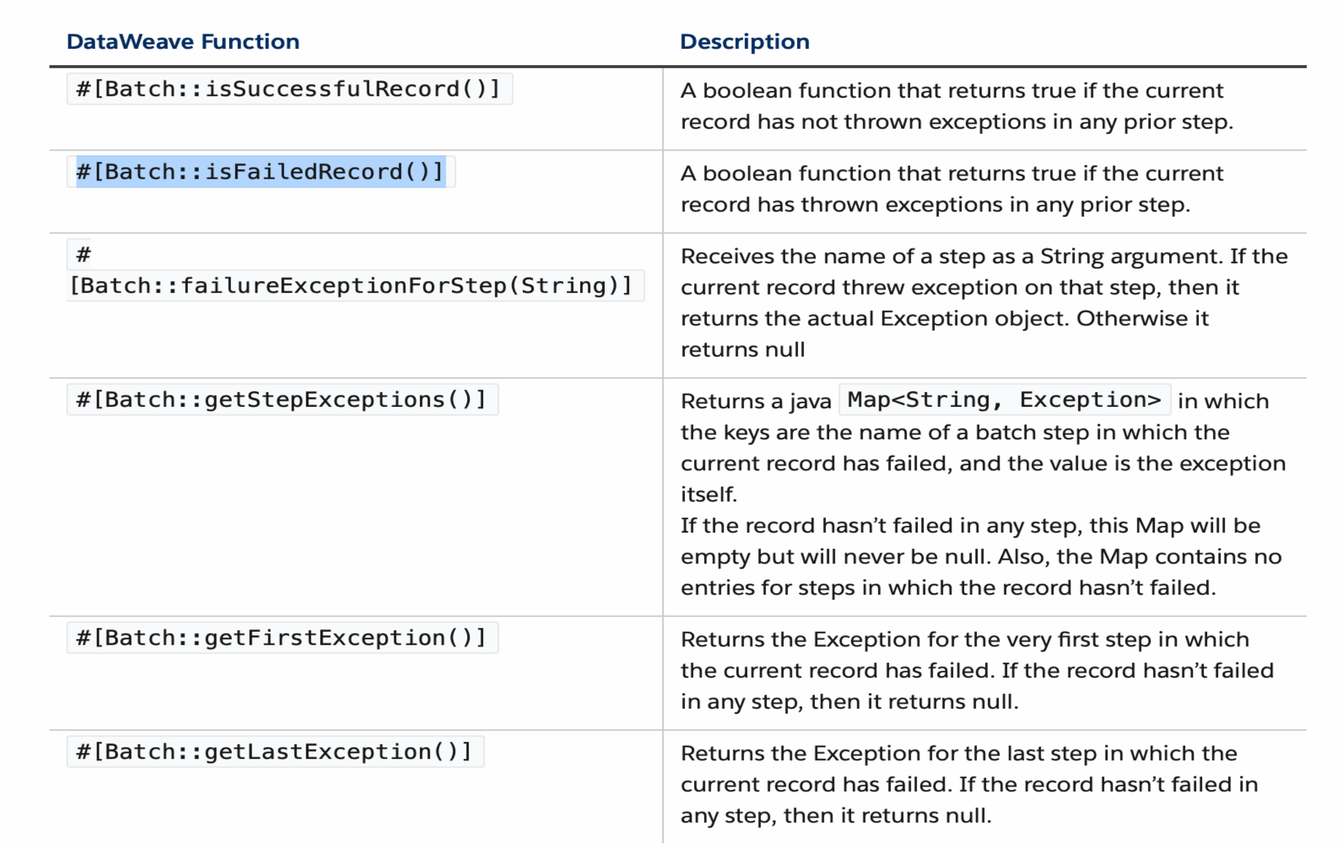
✔ Prevents heap memory pressure and OOM crashes.

Most Common Real-World Values (Used by Mule Architects)



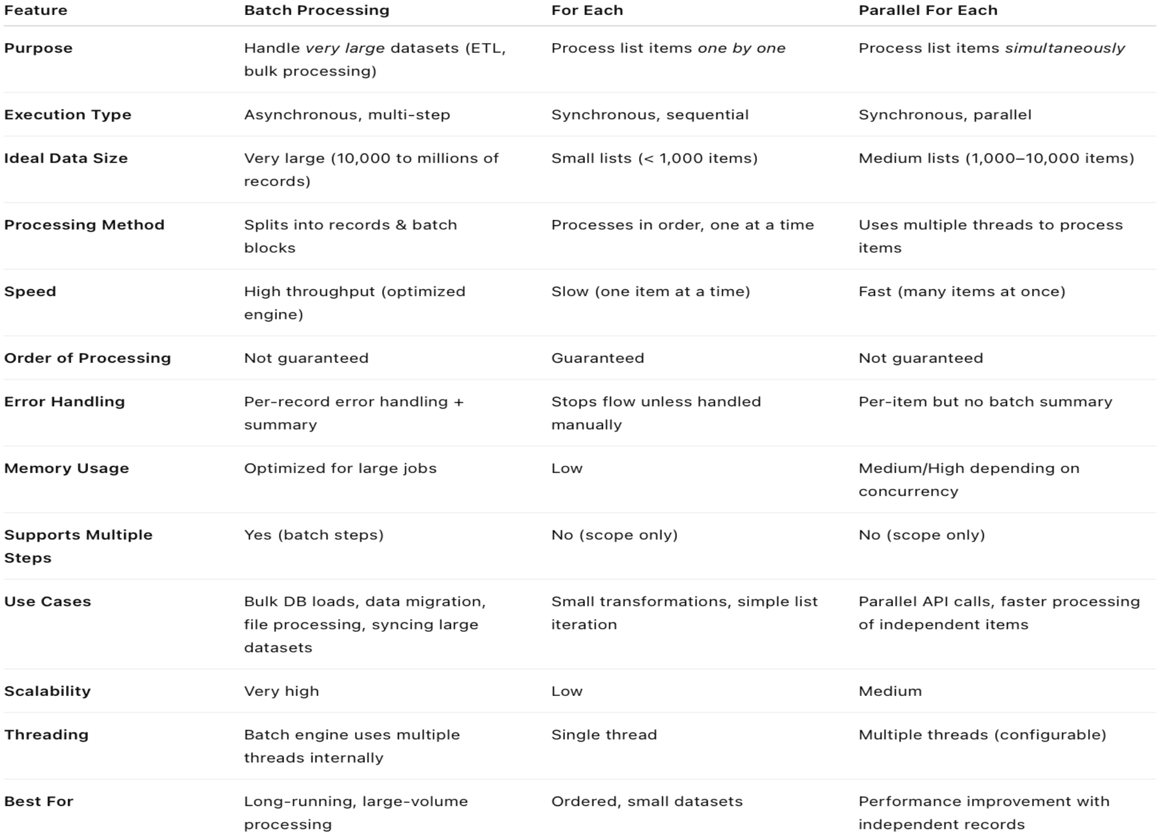
Scenario	Batch Size	Max Concurrency
Small payloads (API calls)	100	8
Medium records (DB rows)	50–100	4
Large records (files, huge JSON)	10–30	1–2
Very large datasets (100k+)	100	cores count

Error Handling in Batch Processing



DataWeave Function	Description
`#[Batch::isSuccessfulRecord()]`	A boolean function that returns true if the current record has not thrown exceptions in any prior step.
`#[Batch::isFailedRecord()]`	A boolean function that returns true if the current record has thrown exceptions in any prior step.
`#[Batch::failureExceptionForStep(String)]`	Receives the name of a step as a String argument. If the current record threw exception on that step, then it returns the actual Exception object. Otherwise it returns null
`#[Batch::getStepExceptions()]`	Returns a java `Map<String, Exception>` in which the keys are the name of a batch step in which the current record has failed, and the value is the exception itself. If the record hasn't failed in any step, this Map will be empty but will never be null. Also, the Map contains no entries for steps in which the record hasn't failed.
`#[Batch::getFirstException()]`	Returns the Exception for the very first step in which the current record has failed. If the record hasn't failed in any step, then it returns null.
`#[Batch::getLastException()]`	Returns the Exception for the last step in which the current record has failed. If the record hasn't failed in any step, then it returns null.

Batch V/S For Each V/S Parallel For Each



Feature	Batch Processing	For Each	Parallel For Each
Purpose	Handle *very large* datasets (ETL, bulk processing)	Process list items *one by one*	Process list items *simultaneously*
Execution Type	Asynchronous, multi-step	Synchronous, sequential	Synchronous, parallel
Ideal Data Size	Very large (10,000 to millions of records)	Small lists (< 1,000 items)	Medium lists (1,000–10,000 items)
Processing Method	Splits into records & batch blocks	Processes in order, one at a time	Uses multiple threads to process items
Speed	High throughput (optimized engine)	Slow (one item at a time)	Fast (many items at once)
Order of Processing	Not guaranteed	Guaranteed	Not guaranteed
Error Handling	Per-record error handling + summary	Stops flow unless handled manually	Per-item but no batch summary
Memory Usage	Optimized for large jobs	Low	Medium/High depending on concurrency
Supports Multiple Steps	Yes (batch steps)	No (scope only)	No (scope only)
Use Cases	Bulk DB loads, data migration, file processing, syncing large datasets	Small transformations, simple list iteration	Parallel API calls, faster processing of independent items
Scalability	Very high	Low	Medium
Threading	Batch engine uses multiple threads internally	Single thread	Multiple threads (configurable)
Best For	Long-running, large-volume processing	Ordered, small datasets	Performance improvement with independent records

Scatter Gather

Scatter-Gather

MuleSoft from Salesforce

Scatter-Gather is a MuleSoft routing component that sends the same incoming message to multiple processing routes in parallel and then combines all the responses into a single output message.

In simple words:

It splits the work → runs tasks at the same time → merges the results.

Capabilities

⚙ Capabilities of Scatter-Gather

Here are the key things Scatter-Gather can do:

🔶 1. Parallel Processing

- ○ It allows multiple flows or API calls to run simultaneously, not one after another.
- ○ This improves performance and reduces response time.

🔶 2. Collects Responses

- ○ After all routes finish their tasks, it gathers all the outputs.
- ○ You get one combined response at the end.

🔶 3. Supports Multiple Routes

You can configure 2, 3, or many routes based on your integration needs.

Each route can:

- ○ Call a different API
- ○ Query a database
- ○ Transform data

Capabilities

🔶 4. Fail-Safe Response Behavior

If one route fails, you can decide:

- ○ Whether the entire scatter-gather fails
- ○ Or only that route’s output is marked as an error
- ○ Or use a Try-Catch inside routes

🔶 5. Non-blocking & Efficient

MuleSoft uses a non-blocking execution engine, meaning:

- ○ It does not waste time waiting
- ○ It efficiently uses available processing threads
- ○ This makes Scatter-Gather very fast.

🔶 6. Maintains Input Message

- The original message is not changed unless you transform it inside a route.