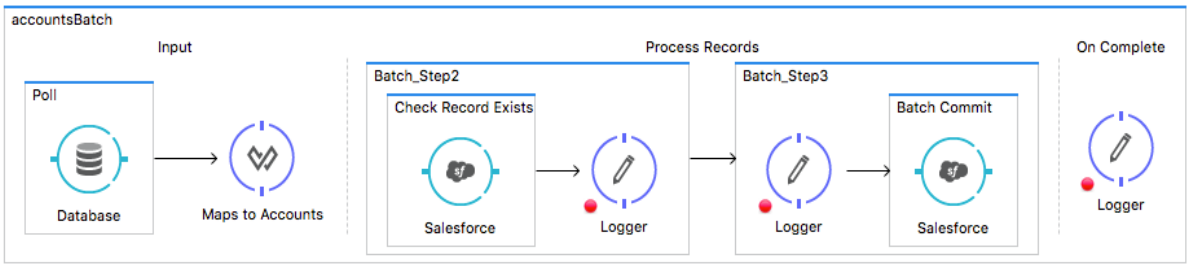
Module 13: Processing Records

Goal



At the end of this module, you should be able to

MuleSoft

- Use the For Each scope to process items in a collection individually
- Use the batch job element (EE) to process individual records
- Trigger a batch job using a poll
- Use a batch job to synchronize data from a legacy database to a SaaS application

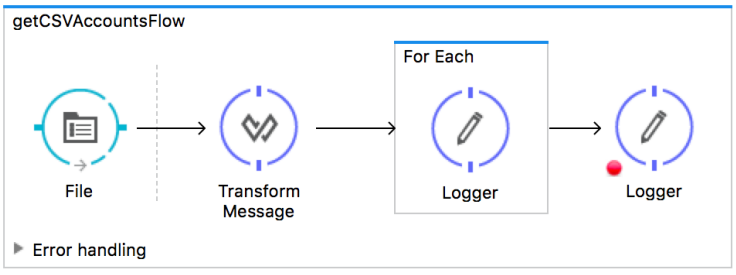
Processing items in a collection

Processing items in a collection

- Create a flow that uses
  - A splitter-aggregator pair
    - One flow control splits the collection into individual elements, which the flow processes iteratively
    - Another flow control is used to re-aggregate the elements into a new collection so they can be passed out of the flow
  - A For Each scope
    - Splits a message collection and processes the individual elements and then returns the original message
    - More versatile and convenient than splitter/aggregator pairs
- Use a batch job (enterprise edition only)
  - Created especially for processing data sets
  - Not a flow, but another top level element

Walkthrough 13-1: Process items in a collection individually

- Use the For Each scope element to process each item in a collection individually
- Look at the thread used to process each record

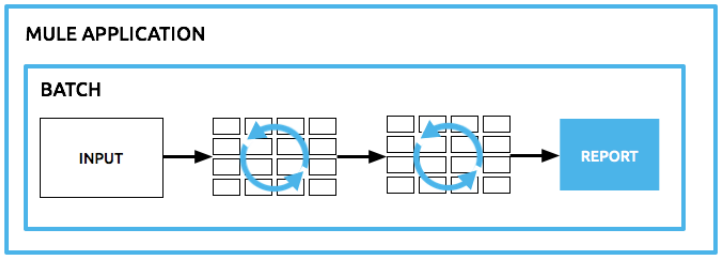


Processing records with the batch job element

Batch processing with the batch job element

MuleSoft

- Is an alternative to standard flows
- Stands on its own as an independent block of code
- Provides ability to split large messages into records that are processed asynchronously in a batch job
- Provides ability to process messages in batches
- Is exclusive to Mule EE runtimes



8

Example use cases

MuleSoft

- Integrating data sets to parallel process records
  - Small or large data sets, streaming or not
- Engineering "near real-time" data integration
  - Synchronizing data sets between business applications
  - Like syncing contacts between NetSuite and Salesforce
- Extracting, transforming and loading (ETL) information into a target system
  - Like uploading data from a flat file (CSV) to Hadoop
- Handling large quantities of incoming data from an API into a legacy system

Batch jobs

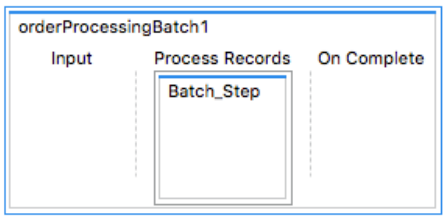
MuleSoft

- Accept data from an external resource
  - May poll for the input
- Split messages into individual records and perform actions upon each record
  - Can use record-level variables to enrich, route, or otherwise act upon records
  - Handle record level failures that occur so batch job is not aborted
- Report on the results and potentially push output to other systems or queues

Creating batch jobs

MuleSoft

- Batch jobs are top-level elements that exists outside the context of any regular Mule flow
- To create, drag a Batch scope element to the canvas

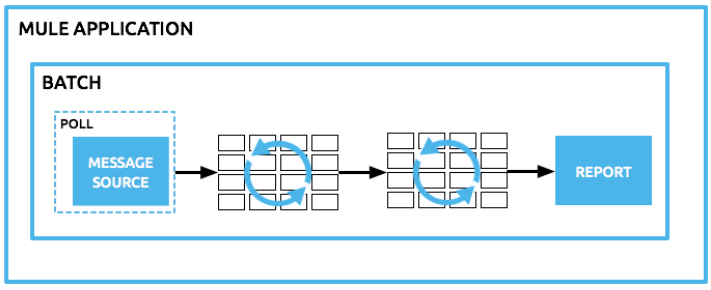



```
<batch:job
name="orderprocessingBatch1">
  <batch:process-records>
    <batch:step name="Step1"/>
  </batch:process-records>
</batch:job>
```


Triggering batch jobs: Option 1

MuleSoft

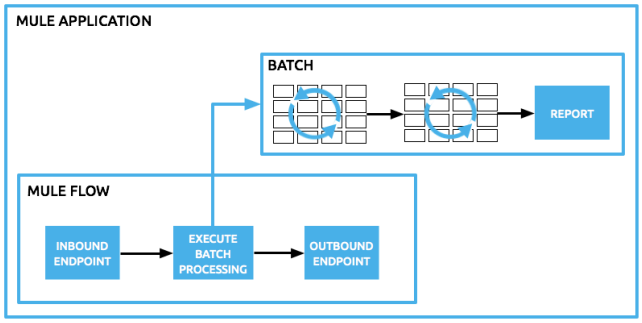
- Place an inbound, one-way message source at the beginning of the batch job
  - It cannot be a request-response inbound message source



Triggering batch jobs: Option 2

MuleSoft

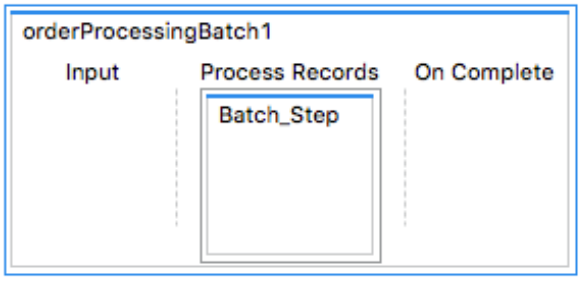
- Use a Batch Execute message processor to reference the batch job from within a Mule flow in the same application



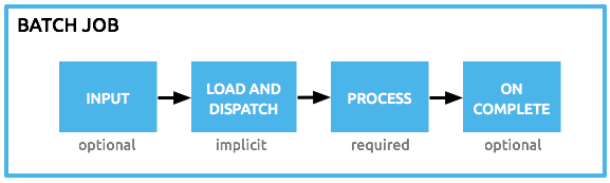
Batch phases in the canvas

MuleSoft

- When you add a Batch scope element to the canvas, multiple phases are shown
  - Input, process records, and on complete

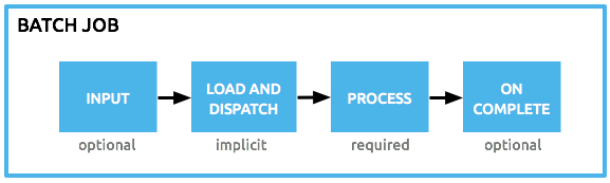


Phases of a batch job



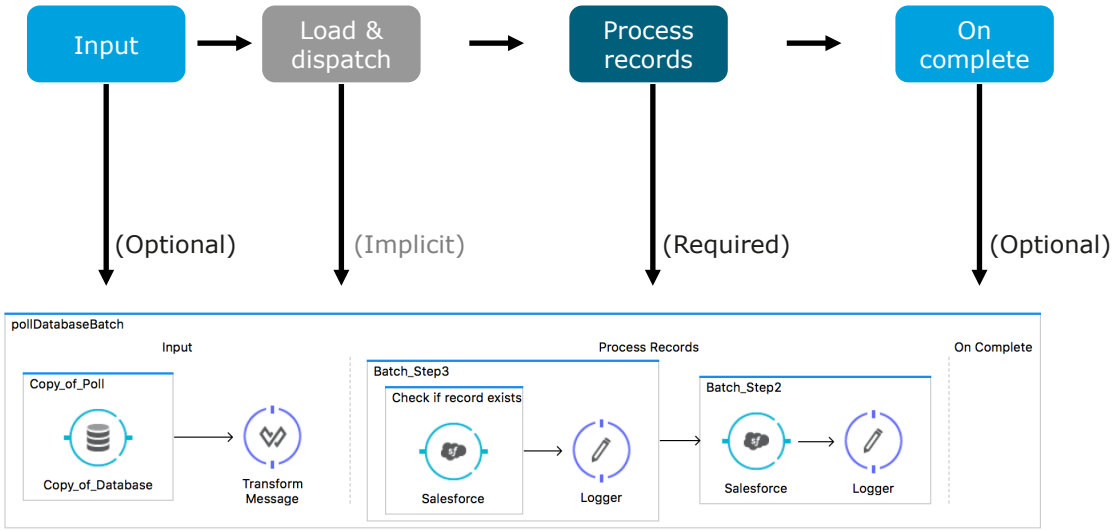
- Input (optional)
  - Triggers the processing via an inbound endpoint
  - Modifies the payload as needed before batch processing
- Load and dispatch (implicit)
  - Performs "behind-the-scene" work
  - Splits payload into a collection of records and creates a queue

Phases of a batch job



- Process (required)
  - Asynchronously processes the records
  - Contains one or more batch steps
- On complete (optional)
  - Report summary of records processed
  - Get insight into which records failed so can address issues

A batch job example



17

How record processing works

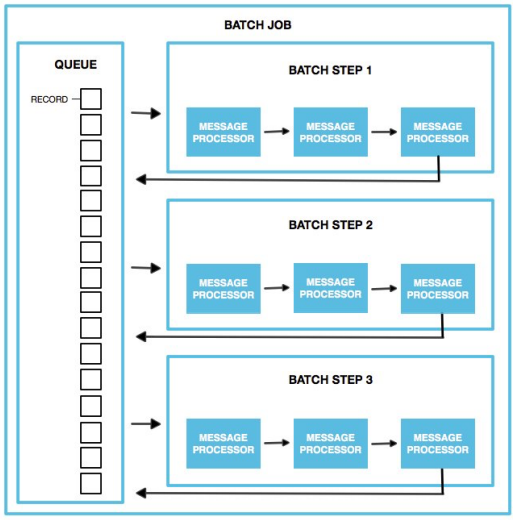
- Only one queue exists and records are picked out of it for each batch step, processed, and then sent back to it
- Each record keeps track of what stages it has been processed through while it sits on this queue
- A batch job instance does not wait for all its queued records to finish processing in one batch step before pushing any of them to the next batch step

18

How record processing works

MuleSoft

- Each record
  - Moves through the processors in the first batch step
  - Is sent back to the queue
  - Waits to be processed by the second batch step
- This repeats until each record has passed through every batch step

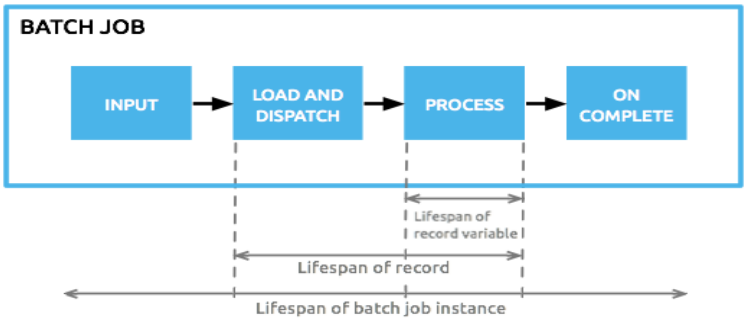


How record processing works

MuleSoft

- To store record-specific information, use record variables
  - Are stored in the recordVars scope
- Persist across all batch steps in the processing phase
  - A flow variable only persists in a single batch step
- Commonly used to capture whether or not a record already exists in a database

Record Variable



Reporting in the on complete phase

- Payload is a BatchJobResult
  - Has properties for processing statistics including
    - loadedRecords
    - processedRecords
    - successfulRecords
    - failedRecords
    - totalRecords

Handling record-level errors during processing

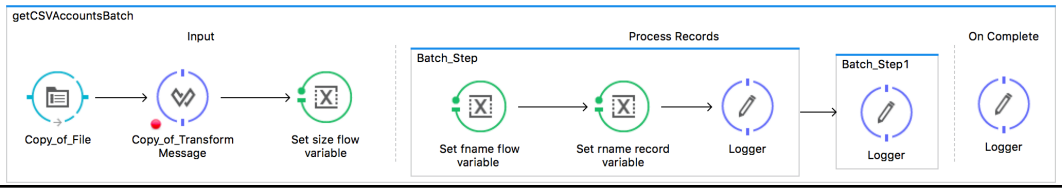
- If a message processor in a batch step cannot process a record (corrupt or incomplete data) there are 3 options

```
<batch:job name="Batch1" max-failed-records="0”>
```

  - 0: Stop processing the entire batch (default)
    - Any remaining batch steps are skipped and all records are passed to the on complete phase
  - -1: Continue processing the batch
    - You need to use filters to instruct subsequent batch steps how to handle failed records
  - {integer}: Continue processing the batch until a max number of failed records is reached
    - All records are then passed to the on complete phase

Walkthrough 13-2: Create a batch job for records in a file

- Create a batch job
- Explore flow & record variable persistence across batch steps & phases
- In the input phase, check for CSV files every second and convert them to a collection of objects
- In the process records phase, create two batch steps for setting and tracking variables
- In the on complete phase, look at the # of records processed and failed
- Look at the thread used to process each record in each step



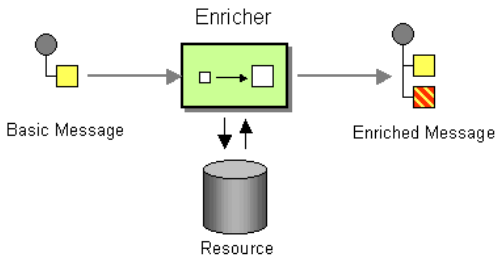
23

Using a batch job to synchronize data

Checking for duplicate records

MuleSoft

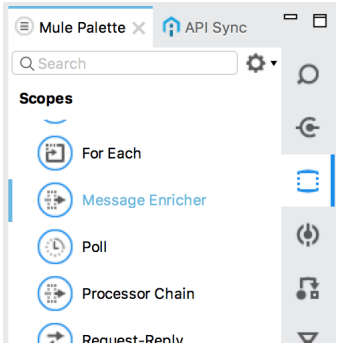
- When synchronizing data between data sources, you often check to see if a record already exists in the target resource
- If you simply add an endpoint to query the target resource first before adding it, the response would become the payload
  - This is not what you want
- You want the external call to act as an enrichment of the existing message with the original payload retained



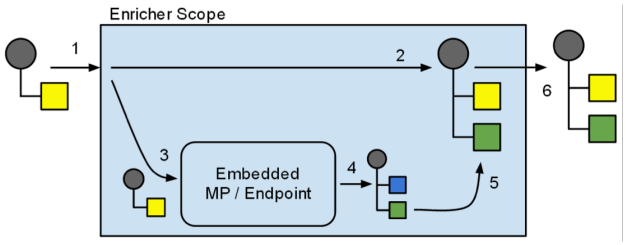
Using a message enricher

MuleSoft

- Add a Message Enricher scope to a flow
- Add message processor(s) to the scope
- Specify the message enricher source and target
  - The target specifies what part of the message to modify
  - The source specifies what to set the target to
    - By default, is equal to payload



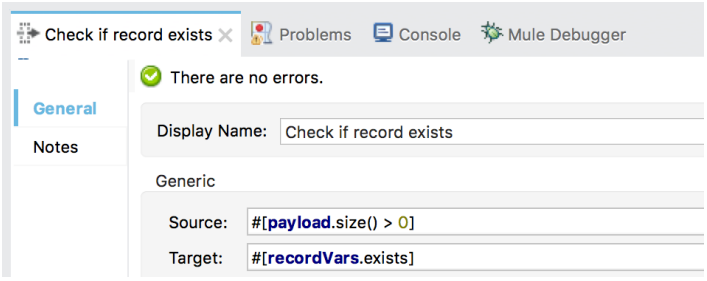
How the message enricher works



1. Enricher sends a copy of the original message into the processor
2. The original message waits
3. The copy is processed
4. The copy's response is a message
5. Part(s) of the response are added to part(s) of the original message
6. The enriched message moves forward

Adding logic to only insert new records

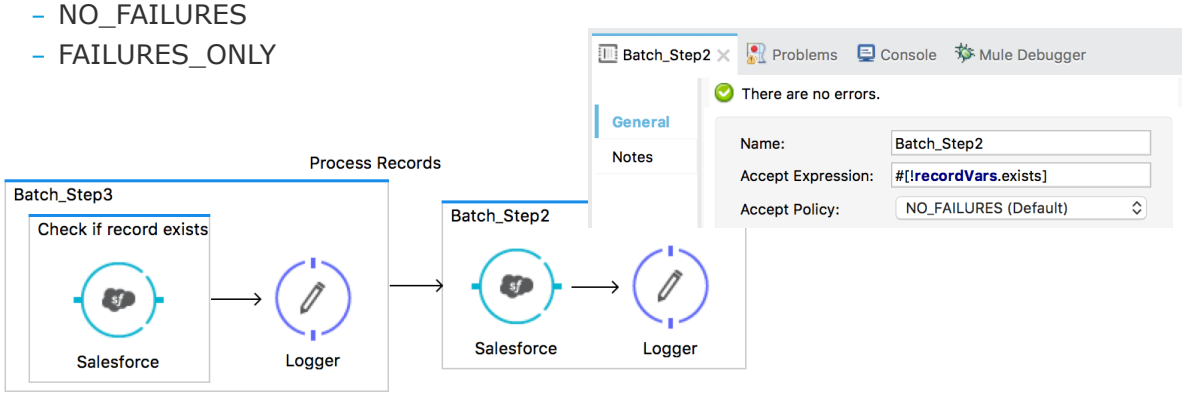
- When determining if a record exists in a target resource, it is common to store the result in a record variable



- After determining if the record exists in the target resource, you can use batch steps filters to only process qualified records

Batch step filters

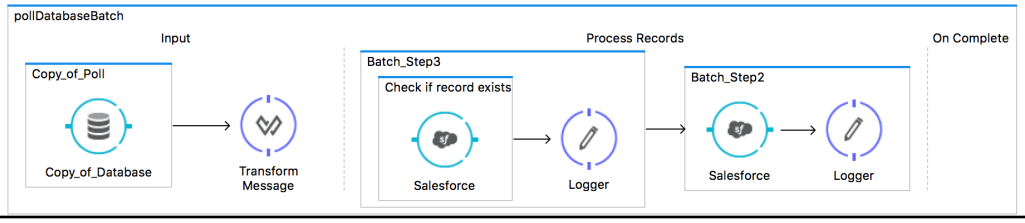
- Restricts records to be processed
- Accept policies
  - ALL
  - NO_FAILURES
  - FAILURES_ONLY



29

Walkthrough 13-3: Restrict processing using a message enricher and a batch step filter

- Create a batch job that polls a database for records with a specific postal code
- Use a message enricher to check if a record already exists in Salesforce (an account with the same Name) and stores the result in a record variable and retains the original payload
- Add a second batch step with a filter that only allows new records (records that don't already exist in Salesforce) to be added to Salesforce
- Use a batch commit scope to commit records in batches



30

Summary

Summary

MuleSoft

- Use the **For Each** scope in a flow to process individual collection elements sequentially and return the original message
- Use the **batch job** element (EE only) for complex batch jobs
  - Created especially for processing data sets
  - It is not a flow, but another top level element
  - It also splits messages into individual records and performs actions upon each record
  - But it can also use record-level variables, handle record level failures, and report on job results
  - Can have multiple batch steps and these can have filters
- A batch job is triggered via a one-way, inbound endpoint in the optional input phase (often within in a poll) or a batch execute from another flow

Summary

MuleSoft

- The implicit **load and dispatch** phase splits the payload into a collection of records and creates a queue
- The **process** phase contains processors in one or more batch steps, which can have filters to restrict which messages are processed
  - Can use record-level variables to enrich, route, or otherwise act upon records
  - Can handle record level failures so the job is not aborted
- The **on complete** phase reports on the results for insight into which records were processed or failed
- Use the **Message Enricher** scope to run nested message processors that do not modify the original payload