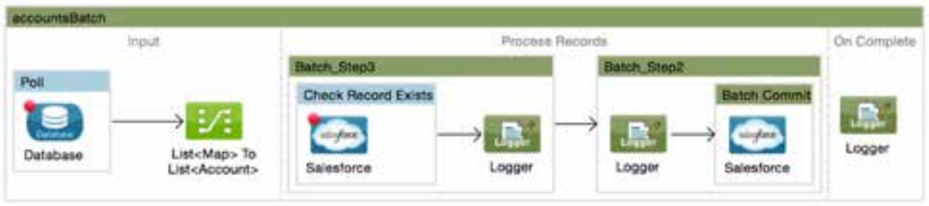
Module 9: Processing Records

Goal



Objectives

- In this module, you will learn:
  - To process items in a collection individually
  - Process items in a collection individually
  - Use DataWeave with CSV files
  - Use the Batch Job element (EE) to process individual records
  - Synchronize data from a CSV file to a SaaS application
  - Synchronize data from a legacy database to a SaaS application

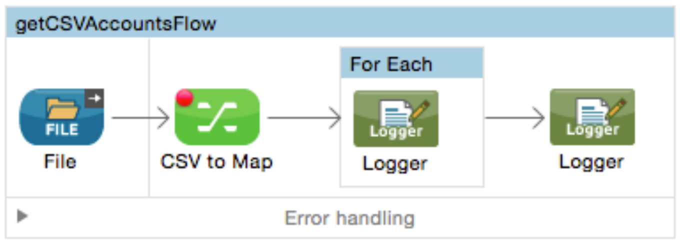
Processing items in a collection

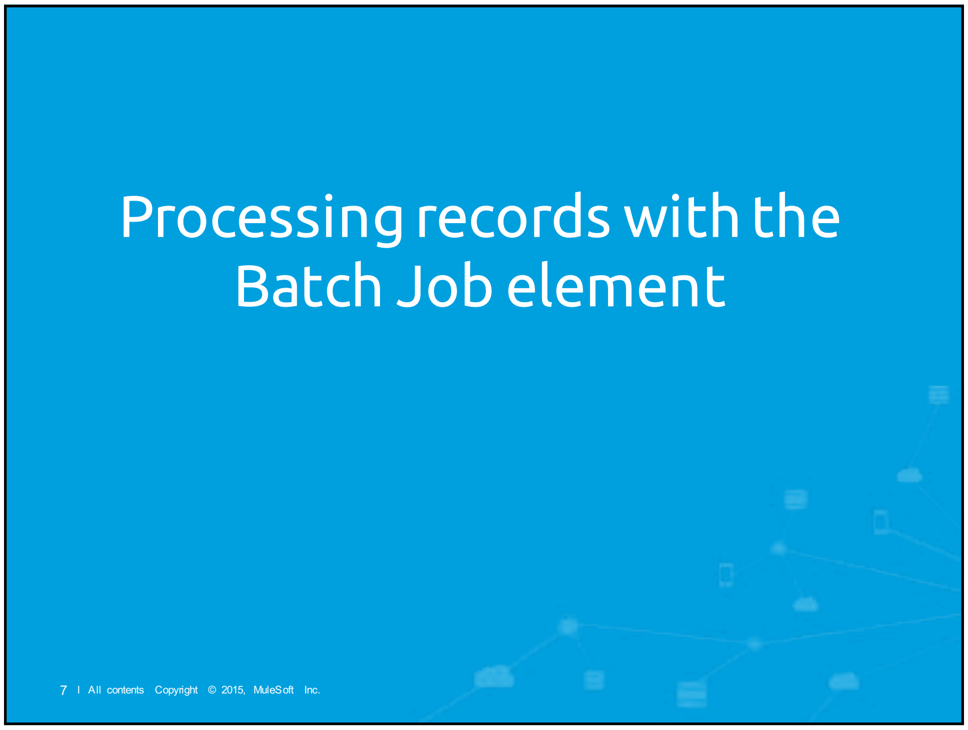
Processing items in a collection

- Create a flow that uses
  - A splitter-aggregator pairs
    - One flow control splits the collection into individual elements, which the flow processes iteratively, then 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 that splitter/aggregator pairs
- Use a batch job (EE)
  - Created especially for processing data sets
  - Not a flow, but another top level element

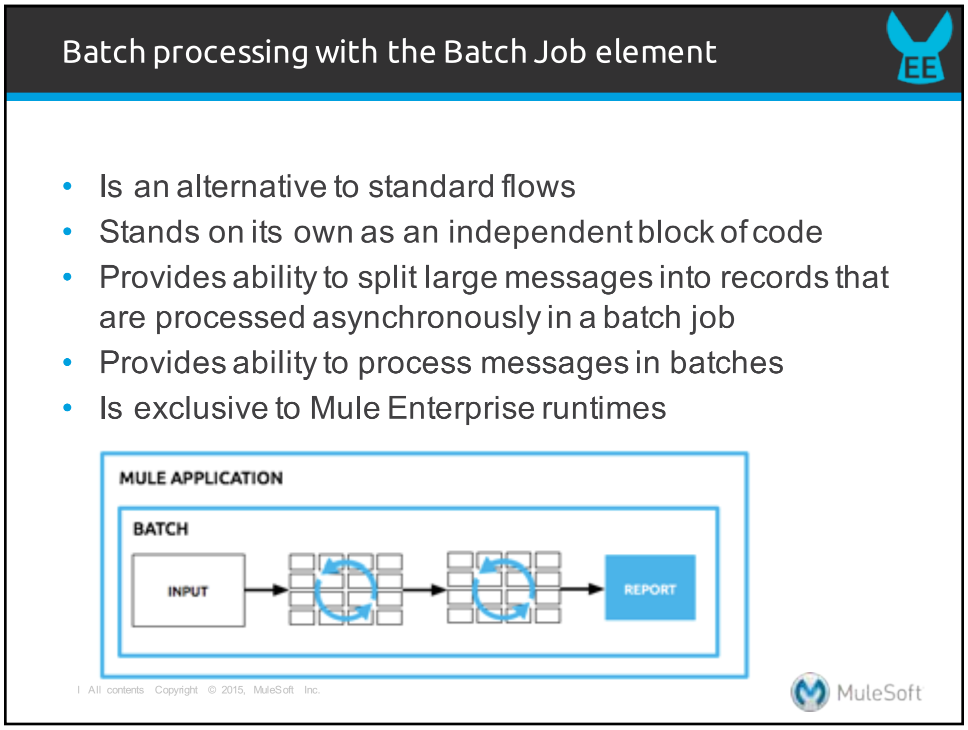
Walkthrough 9-1: Process items in a collection individually

- In this walkthrough, you will:
  - Add metatdata to a File endpoint
  - Read a CSV file and use DataWeave to convert it to a collection of objects
  - Use the For Each scope element to process each item in a collection individually



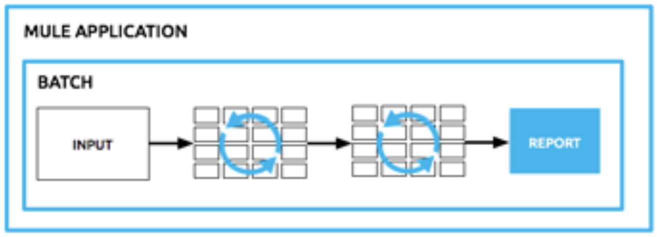


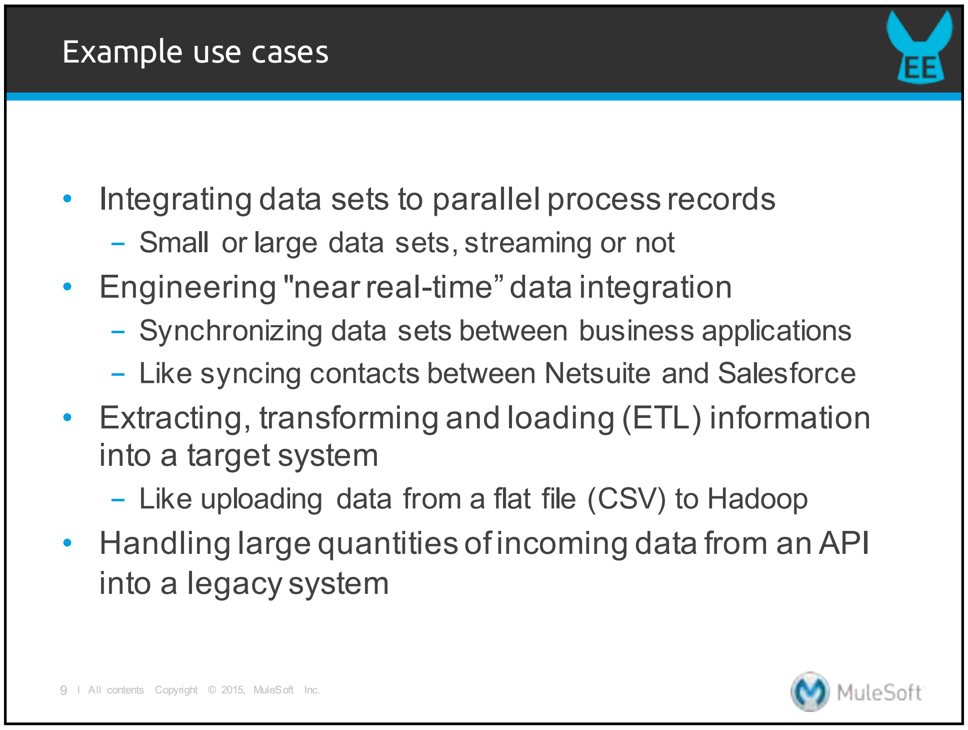
Processing records with the Batch Job element



Batch processing with the Batch Job element

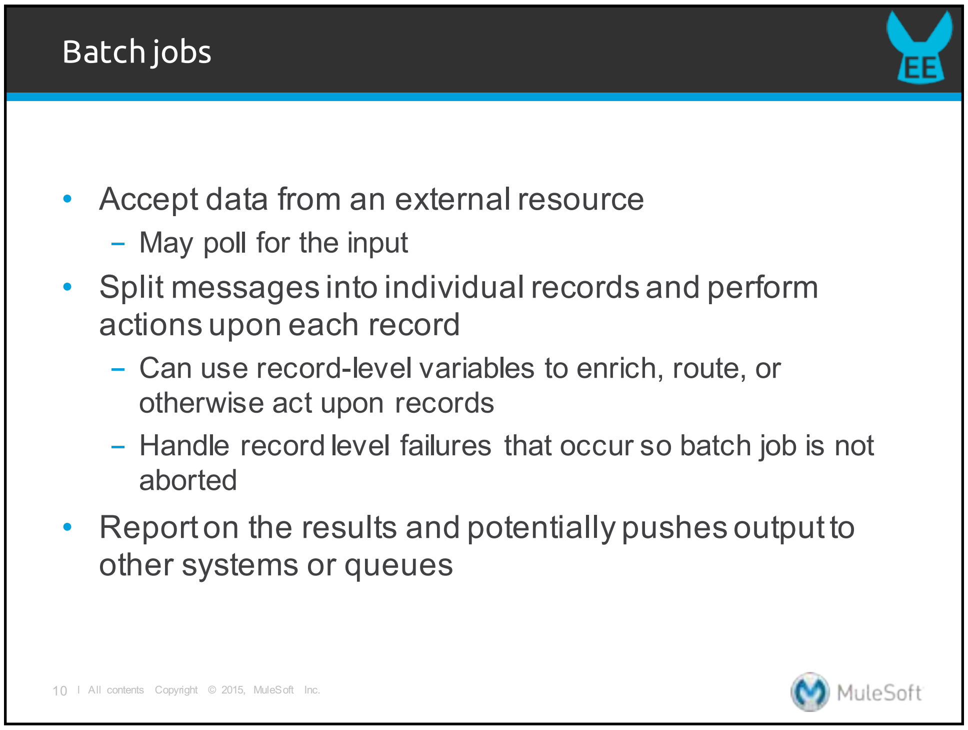
- Is an alternative to standard flows
- Stands on its own as an independent block of code
- Provides ability to split large messages into records that are processed asynchronously in a batch job
- Provides ability to process messages in batches
- Is exclusive to Mule Enterprise runtimes





Example use cases

- 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

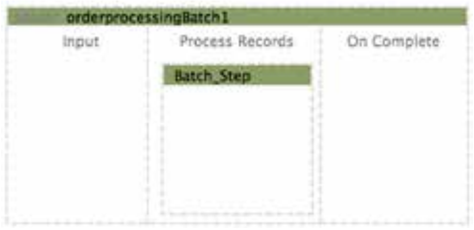


Batch jobs

- 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 pushes output to other systems or queues

Creating batch jobs

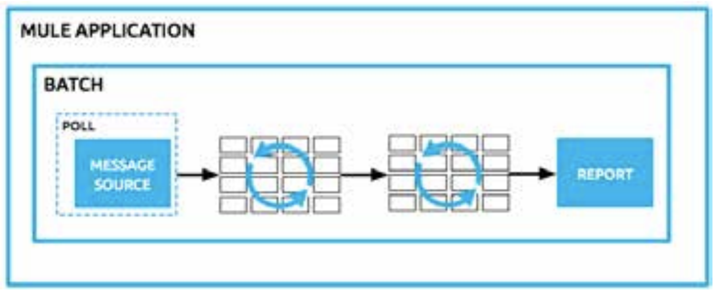
- Batch jobs are top-level elements that exists outside the context of any regular Mule flow
- To create
  - Drag a Batch scope element to the canvas or
  - Add a <batch:job> in XML



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

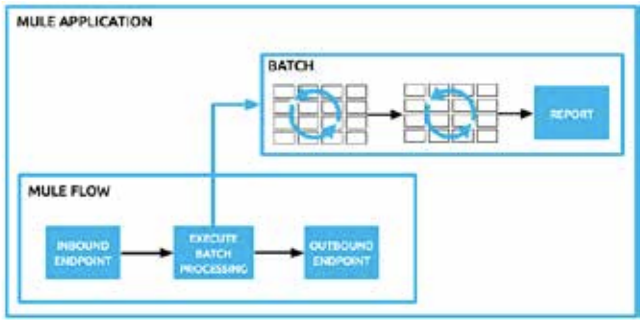
Triggering batch jobs: Option 1

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



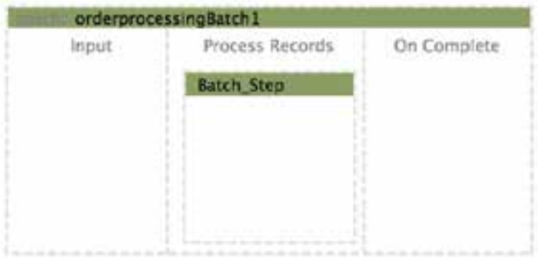
Triggering batch jobs: Option 2

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

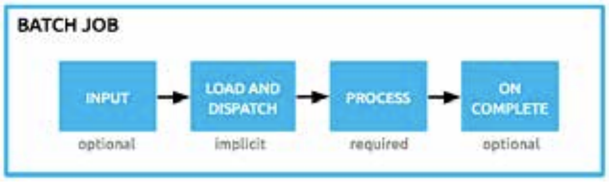


Batch phases in the canvas

- When you add a Batch scope element to the canvas, multiple phases are shown
  - Input, Process Records, and On Complete

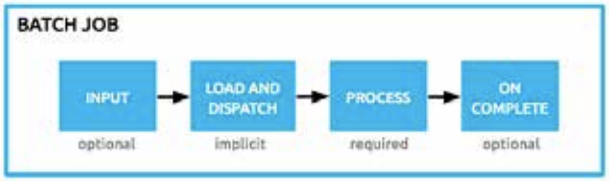


Phases of a batch job



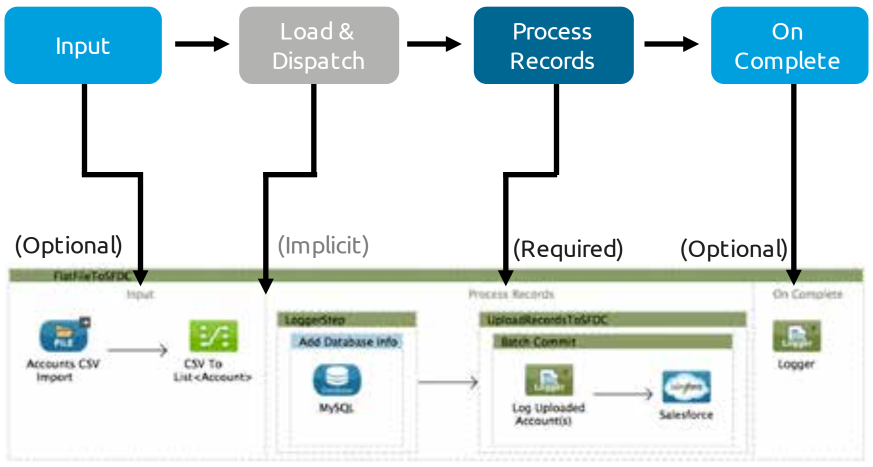
- Input (optional)
  - Triggers the processing via an inbound endpoint
  - Modifies the payload as needed before batch processing
- Load and dispatch (implicit)
  - Performs "behind-the-scene" work
  - Splits payload into a collection of records and creates a queue

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



MuleSoft

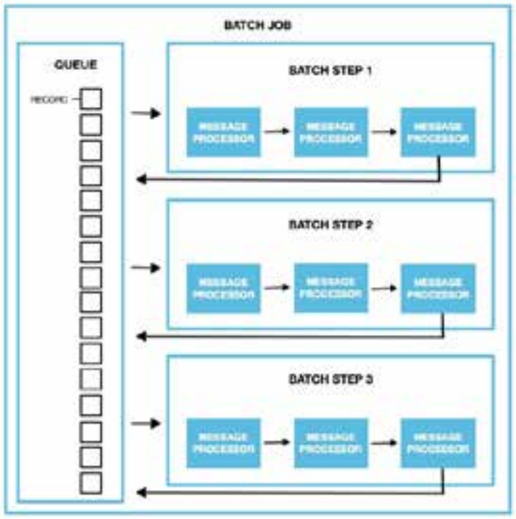
How record processing works

- 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

MuleSoft

How record processing works

- 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



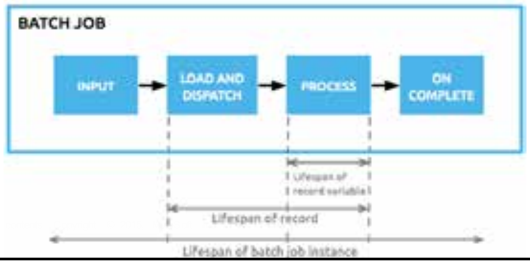
MuleSoft

Batch record variables

Variable

Record Variable

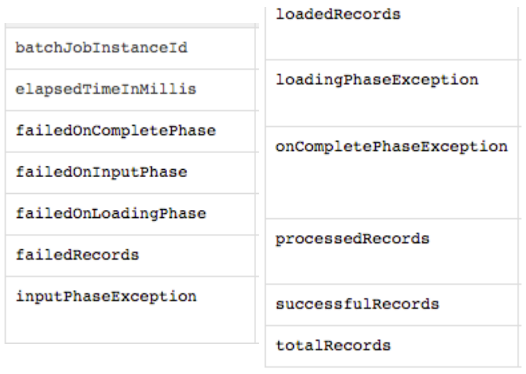
- Store information at the record level, rather than the flow or session level
- Persist across all batch steps in the processing phase
  - A flow variable only persists in a single batch stop
- Commonly used to capture whether or not a record already exists in a database
- Is stored in the recordVars scope



MuleSoft

Reporting in the on complete phase

- Payload is a BatchJobResult
  - Has properties for processing statistics



batchJobInstanceId	loadedRecords
elapsedTimeInMillis	loadingPhaseException
failedOnCompletePhase	onCompletePhaseException
failedOnInputPhase	processedRecords
failedOnLoadingPhase	successfulRecords
failedRecords	totalRecords
inputPhaseException	

Handling record-level errors during processing

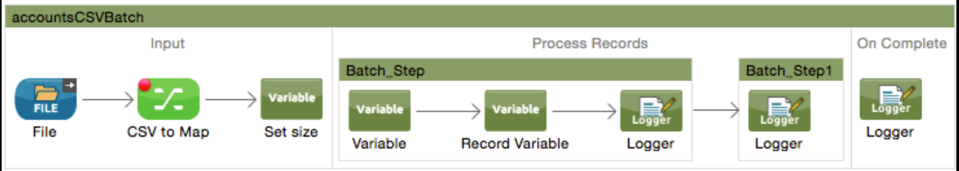
- If a message processor in a batch step cannot process a record (corrupt or incomplete data) there are 3 options

```
<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 9-2: Create a batch job for records in a file

- In this walkthrough, you will:
  - Create a new flow containing a batch job
  - Explore flow and record variable persistence across batch steps and 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, display the number of records processed and failed



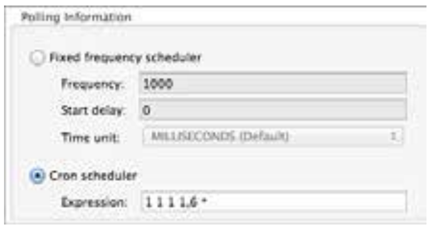
Polling resources

Polling resources

- Most message processors in Mule are triggered when called by a previous element in a flow
- Some connectors use or can use a polling process to actively retrieve messages from an external resource
  - File, FTP, SFTP
- If you want the other message processors to actively call a resource at regular intervals
  - Use a Poll scope element

Scheduling a poll

- By default, a resource is polled every 1000 milliseconds
- There are two methods to change the polling interval
  - Fixed frequency scheduler
  - Cron scheduler



Polling Information

( ) Fixed frequency scheduler

Frequency:	1000
Start delay:	0
Time unit:	MILLISECONDS (Default)

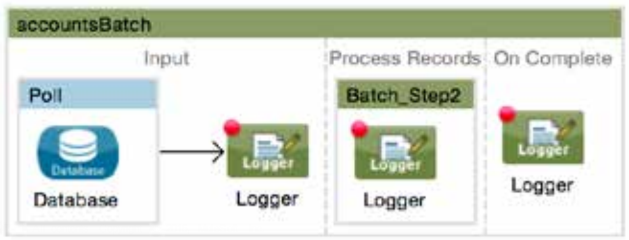
(●) Cron scheduler

Expression:	1 1 1 1,6 *

0 15 10 ? * *	Poll at 10:15am every day
0 15 10 * * ? 2015	Poll at 10:15pm every day in 2015
1 1 1 1,6 *	Poll the first day of January and June every year in the first second of the first minute of the first hour

Walkthrough 9-3: Create a batch job for records in a database

- In this walkthrough, you will:
  - Go a form and add multiple accounts for a specific postal code to the database
  - Create a new flow containing a batch job that polls a MySQL database every 30 seconds for records with a specific postal code
  - Use the Poll scope



MuleSoft

Restricting processing using a poll watermark

Polling for new data using watermarks

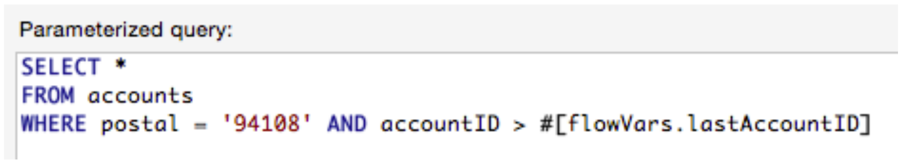
- Instead of polling a resource for all its data every call, you often want to only retrieve the data that has been newly created or updated since the last call
- To do this, you need to keep a persistent record of either
  - The item that was last processed
  - The last time the resource was polled
- In the context of Mule flows, this persistent record is called a watermark

How watermarks work

- The first time the poll runs, the watermark is set to a default value
- It is then used as necessary when running a query or calling a resource
- The value of the watermark may be kept or changed depending upon the logic
- The value must persist across flows
  - Mule uses a built-in object store for persistent storage and exposes the value as a flow variable
    - Saved to file for embedded Mule and standalone Mule runtime
    - Saved to data storage for CloudHub
    - Saved to shared distributed memory for clustered Mule runtimes

Walkthrough 9-4: Restrict processing using a poll watermark

- In this walkthrough, you will:
  - Modify the Poll to use a watermark to keep track of the last record returned from the database
  - Modify the database query to use the watermark
  - Clear application data


Parameterized query:

```
SELECT *
FROM accounts
WHERE postal = '94108' AND accountID > #[flowVars.lastAccountID]
```

MuleSoft

Restricting processing using a message enricher and a batch step accept policy

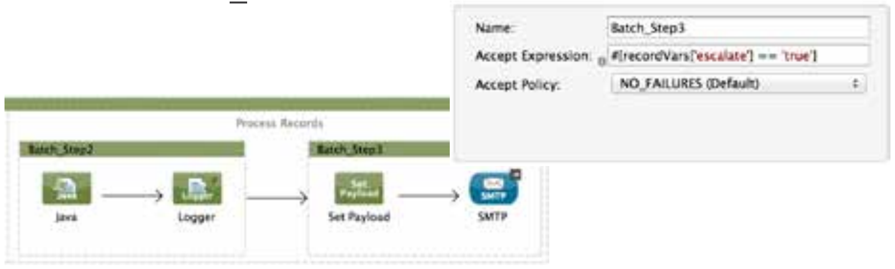
How to avoid processing existing records

- Check to see if a record already exists in the target resource
  - Use the Message Enricher scope to run “nested” message processors that do not modify the original payload
  - Store this result in a record variable
- To subsequent batch steps, add filters to only process qualified records

MuleSoft

Batch step filters

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

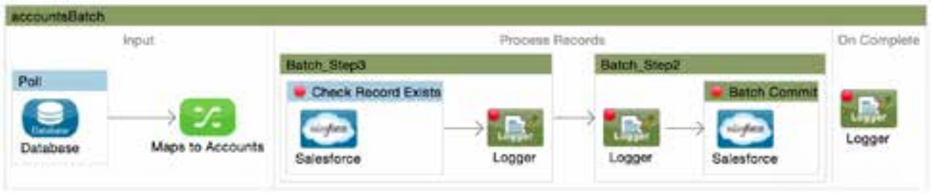


Name: Batch_Step3
Accept Expression: #[recordVars['escalate'] == 'true']
Accept Policy: NO_FAILURES (Default)

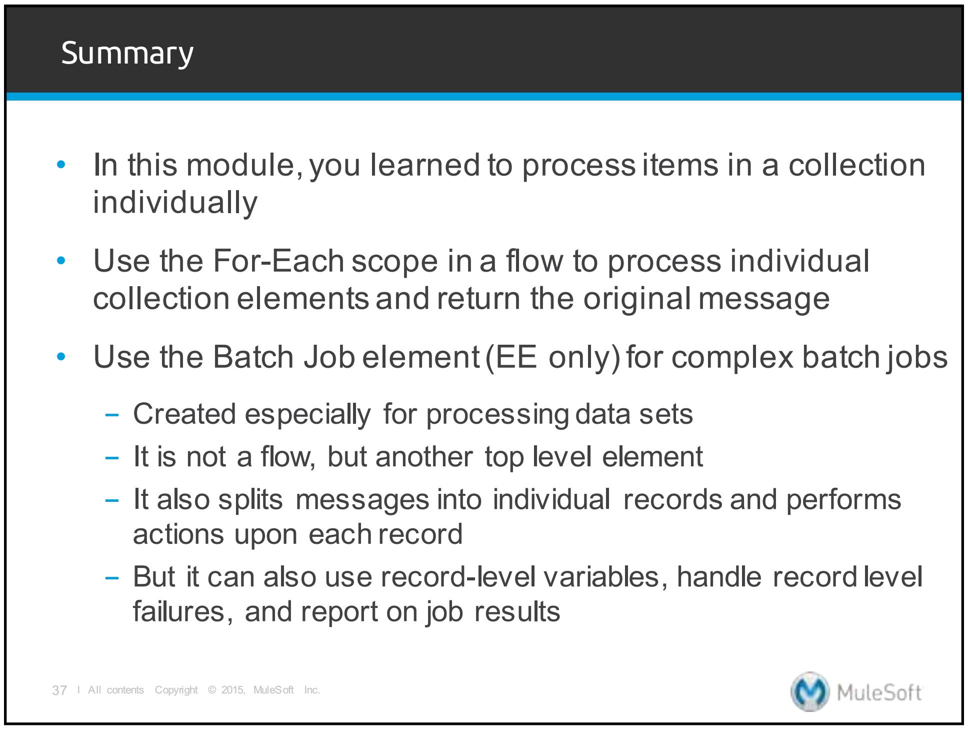
MuleSoft

Walkthrough 9-5: Restrict processing using a message enricher and a batch step filter

- In this walkthrough, you will:
  - Add a first batch step with a Message Enricher scope element that checks 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
  - Modify the second batch step to use a filter that only allows new records (records that don't already exist) to be processed
  - (Optional) Add the record(s) to Salesforce

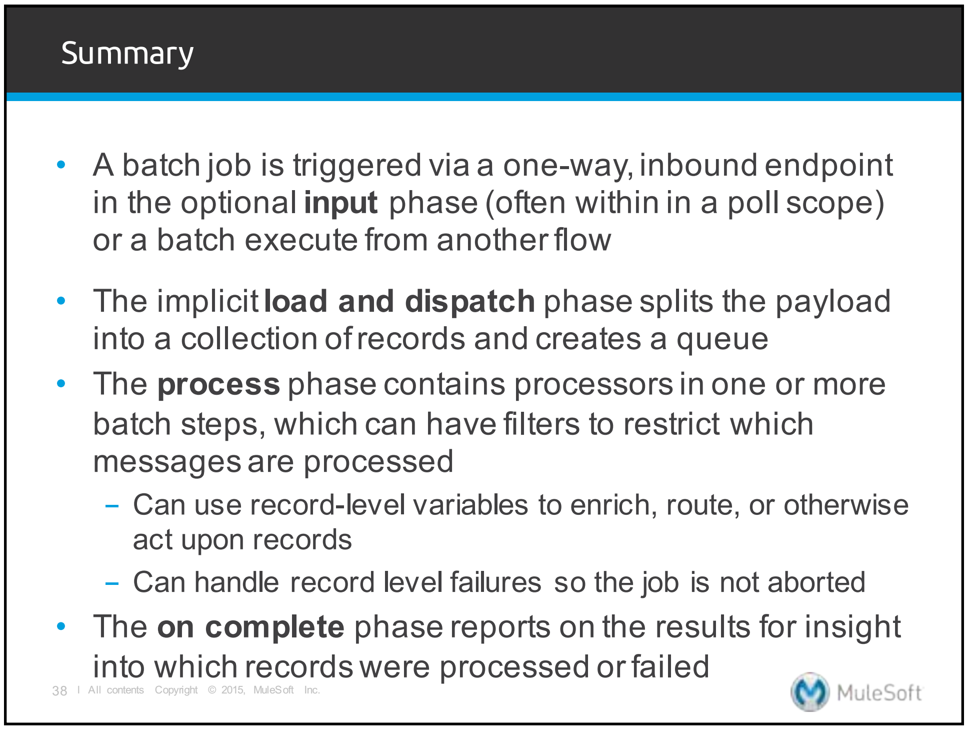


Summary



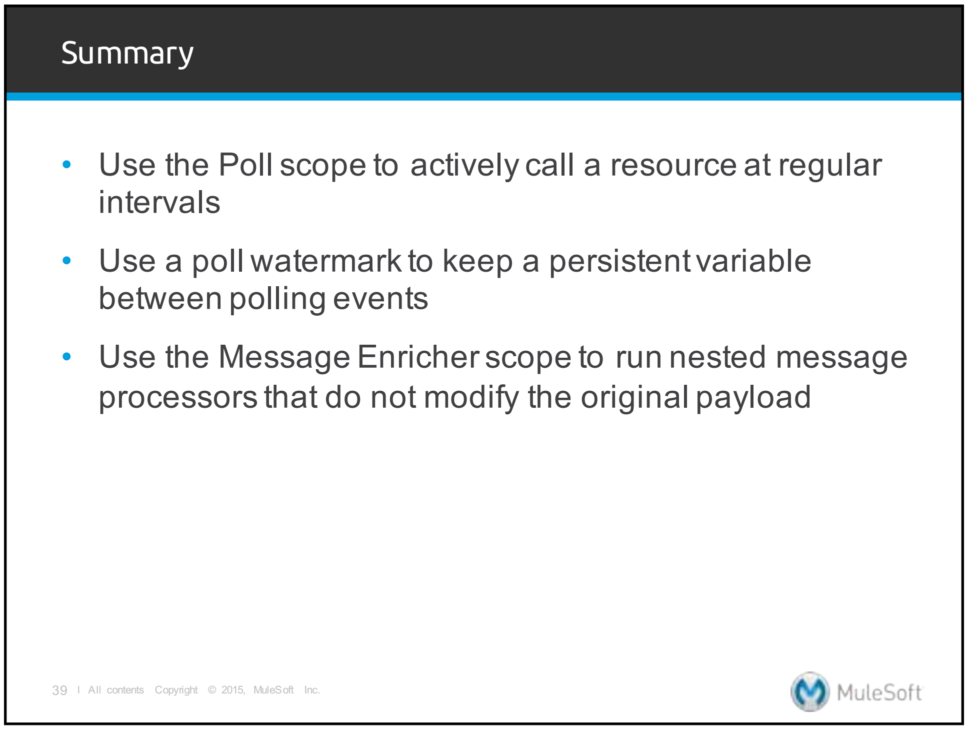
Summary

- In this module, you learned to process items in a collection individually
- Use the For-Each scope in a flow to process individual collection elements 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



Summary

- A batch job is triggered via a one-way, inbound endpoint in the optional **input** phase (often within in a poll scope) or a batch execute from another flow
- 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



Summary

- Use the Poll scope to actively call a resource at regular intervals
- Use a poll watermark to keep a persistent variable between polling events
- Use the Message Enricher scope to run nested message processors that do not modify the original payload

MuleSoft