Streaming Error Handling - Draft

German Sales Technology Platform

Exported on 12/20/2023

Table of Contents

1 About

This page describes an approach for error handling in Mule applications, which use streaming.

The target audience is developers, architects and product owners

2 Summary

While streaming large files, errors cannot easily be handled using the Mule ON-error components because this would close the stream. Steaming however is critical to avoid forcing entire datasets into memory, which would require excessive memory and might cause out-of-memory errors.

A potential alternative is using large Cloudhub workers, which can allocate large data structures in memory. However, this is expensive in terms of vCores consumption and often unnecessary.

The "total function" approach consists of implementing DataWeave transformations, which do not throw errors in case unsuccessful processing but return the error as a part of the result of their invocation. The term "total" indicates that the DataWeave transformation is defined over all input, including input, which might lead to error.

For large input typically consists of CSV files. DataWeave transformations typically use the "map" and "filter" functions to process each row individually. The "map" function uses a function as an argument, which does the actual processing. This function takes a row as an argument and returns the new, processed row. The approach described here makes this processing function "total".

Since there is no unique "choice" type in CSV structures, we use a specific column for errors. An empty errors field indicates succeful processing.

3 Implementation

3.1 Functionality

The solution consumes an input file in CSV format, applies a total function to each row and then splits resulting CSV file into one for which the
application caused errors and one for which it was successful.

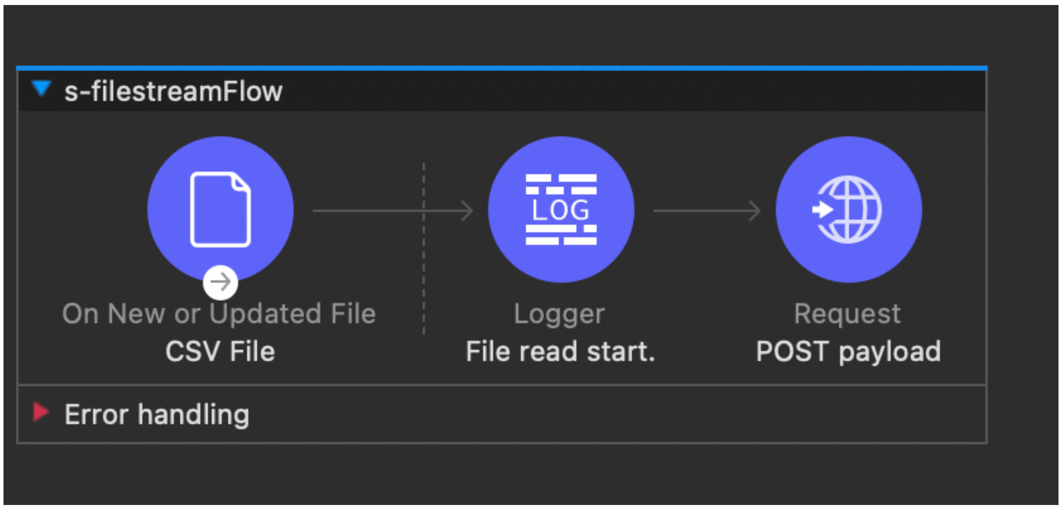
3.2 Mule applications

The solution consists of two Mule applications:

3.2.1 The application "s-streamer"

This application
* reads a CSV from the file system and
* POSTs it to the other application's POST endpoint.

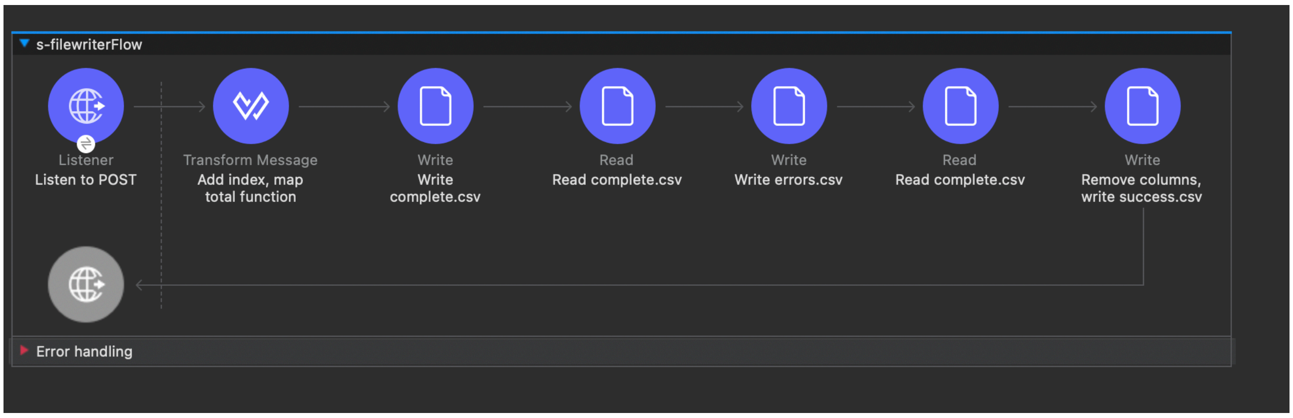
To use this approach on Cloudhub the file processor would typically be replaced by an SFTP connector.



3.2.2 The applciation "s-writer"

This application
* listens to the CSV file content on an HTTP listener,
* applies the total function
* writes the resulting CSV file to the file system

* filters out the error results and writes these to the file system
* filters out the success results and writes these to the file system



3.3 Streming enablement

* The "s-streamer" application uses request streaming mode "ALWAYS" and
* has set the "mule.http.requestStreaming.enable=true" property
* The "s-writer" application uses the streaming strategy "Repeatable file store stream"
* The DataWeave tranformation uses the @StreamCapable() annotation and the "deferred=true" output property

3.4 DataWeave tranformation

The DataWeave transfomation below maps a function "totFun" on the rows of the CSV payload. This function simulates partial failure using a random number.

The transformation additionally adds a new index column "ind".

```
%dw 2.0
@StreamCapable()
input payload application/csv streaming=true
output application/csv deferred=true

fun totFun( arg: Object): Object =
    if( floor(random() * 4) == 0 )
        {
            "errorMsg": "failed",
        } ++ (arg mapObject( v,k,i) -> { (k) : "" })
    else
        {
            "errorMsg": "",
        } ++ arg
---
payload as Array<Object> map (row) ->  {"ind": "$$"} ++ totFun(row)
```

4 Results

To test drive this application I used a test dataset, which I generated using the Groovy script below.

The Studio runtim VM arguments wer set to " -Xms512m -Xmx512m" to simulate an XS worker.

The solution transformed a dataset with 20 M records (550 MB) without memory problems within a few minutes.

5 Appendix

5.1 Test data creation script

```
def BigInteger numLines = args[0] as BigInteger
println "iLine,first,last"
for (BigInteger i = 1; i < numLines+1; i ++) {
  println String.format ("%d,f%d,l%d",i,i,i)}
```