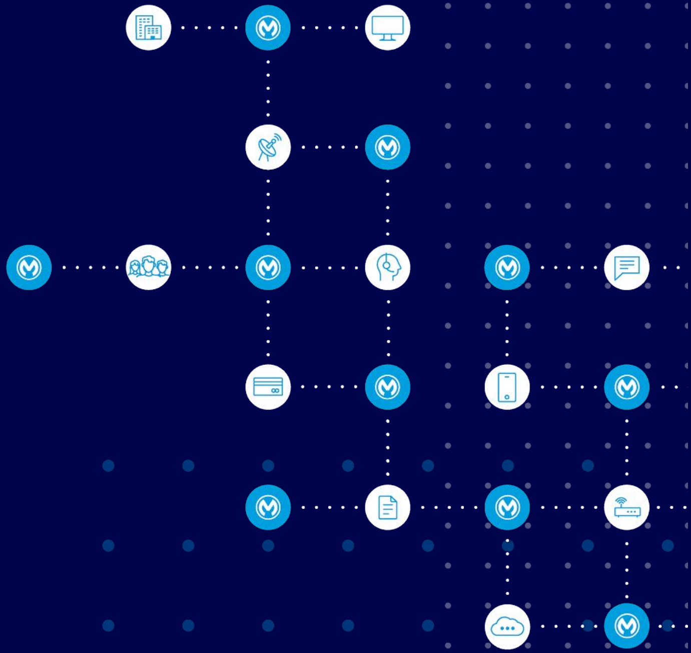
MuleSoft®

Large File Processing

Challenges and Solutions

Chuck Hirstius
Chad Scott



Agenda

1. Background
2. Challenges
3. Solutions and Outcomes

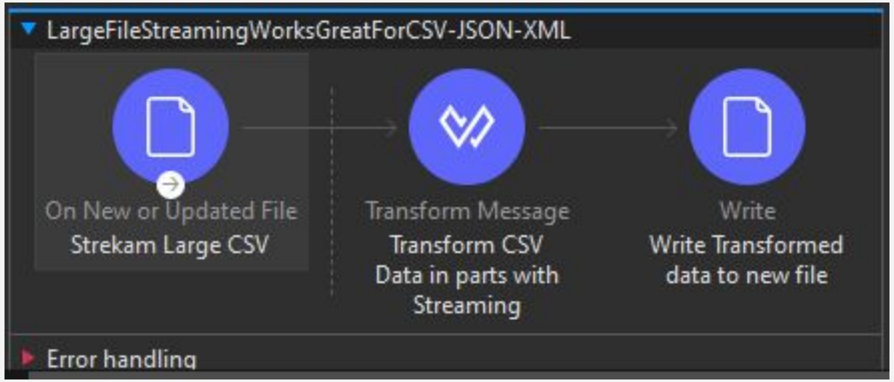
File Streaming in Mule Applications

Mulesoft Flows can be configured to handle data using streaming, with a number of options on how to store the data as it waits to be processed: In memory, or stored on filesystem

But the data must be broken into something mule processors can interpret as something like a record

CSV files, JSON or XML with arrays or nodes all fit this model, as do many java data types

Following this model, very large data sources can be processed with little memory overhead



LargeFileStreamingWorksGreatForCSV-JSON-XML
On New or Updated File
Strekam Large CSV
Transform Message
Transform CSV Data in parts with Streaming
Write
Write Transformed data to new file
Error handling

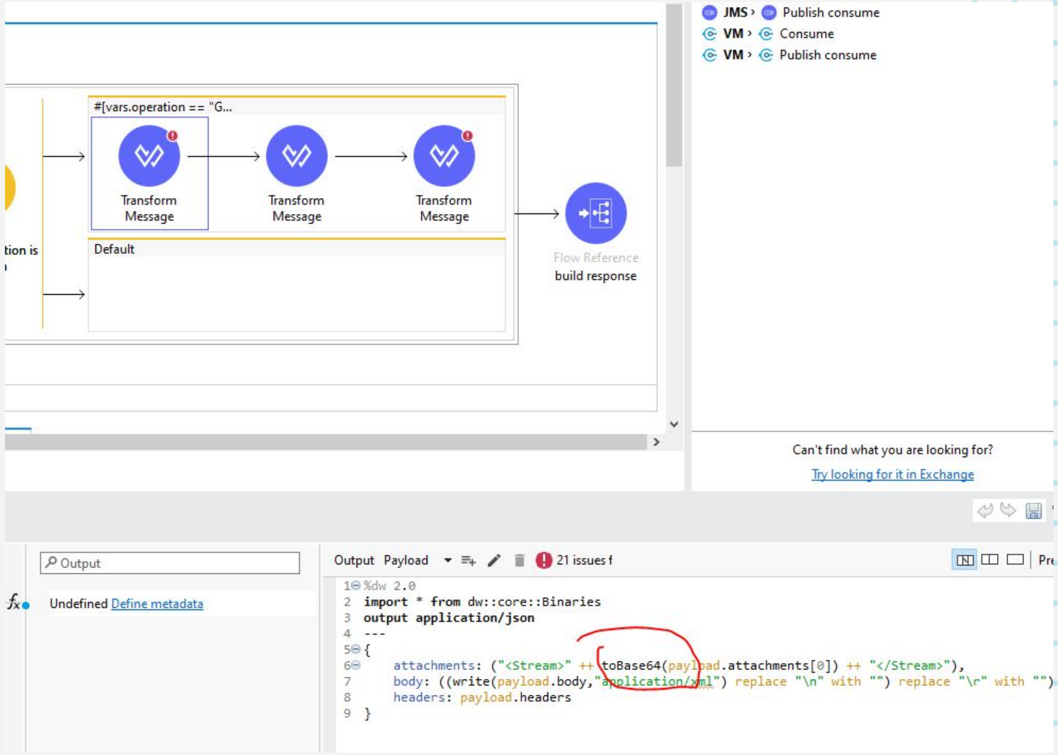
File Streaming Non-“Record” Type Data

If the payload Mule is streaming is not in a structure that can be split into blocks of records then when many processors retrieve the next block (up to the buffer size) they will get the entire payload.

For large binary file data, that means the entire file will be loaded into memory

Potentially numerous times for a single processor

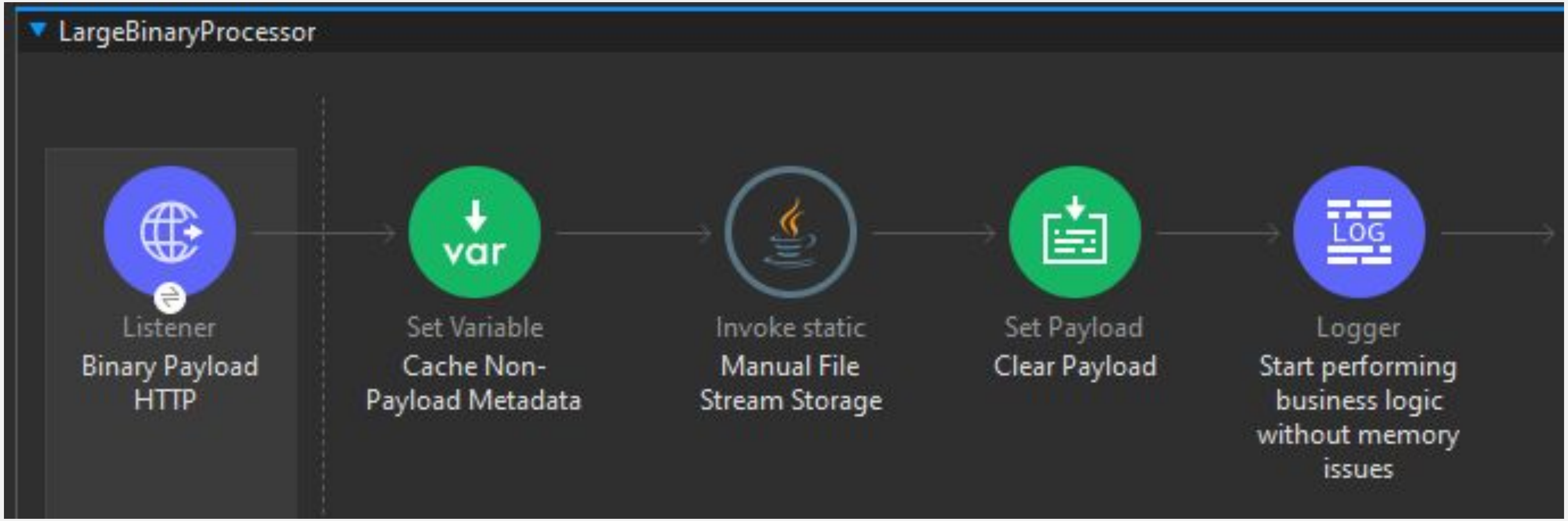
Fortunately, some processors CAN handle binary data streaming. Such as HTTP Listener and java invocation



```
%dw 2.0
import * from dw::core::Binaries
output application/json
---
{
    attachments: ("<Stream>" ++ toBase64(payload.attachments[0]) ++ "</Stream>"),
    body: ((write(payload.body,"application/xml") replace "\n" with "") replace "\r" with "")
    headers: payload.headers
}
```

How to protect a Mule app from a large binary file

By catching the large file early enough, we can protect it from EVER having to be loaded into memory in its entirety.


LargeBinaryProcessor
Listener
Binary Payload HTTP
Set Variable
Cache Non-Payload Metadata
Invoke static
Manual File Stream Storage
Set Payload
Clear Payload
Logger
Start performing business logic without memory issues

How to protect a Mule app from a large binary file

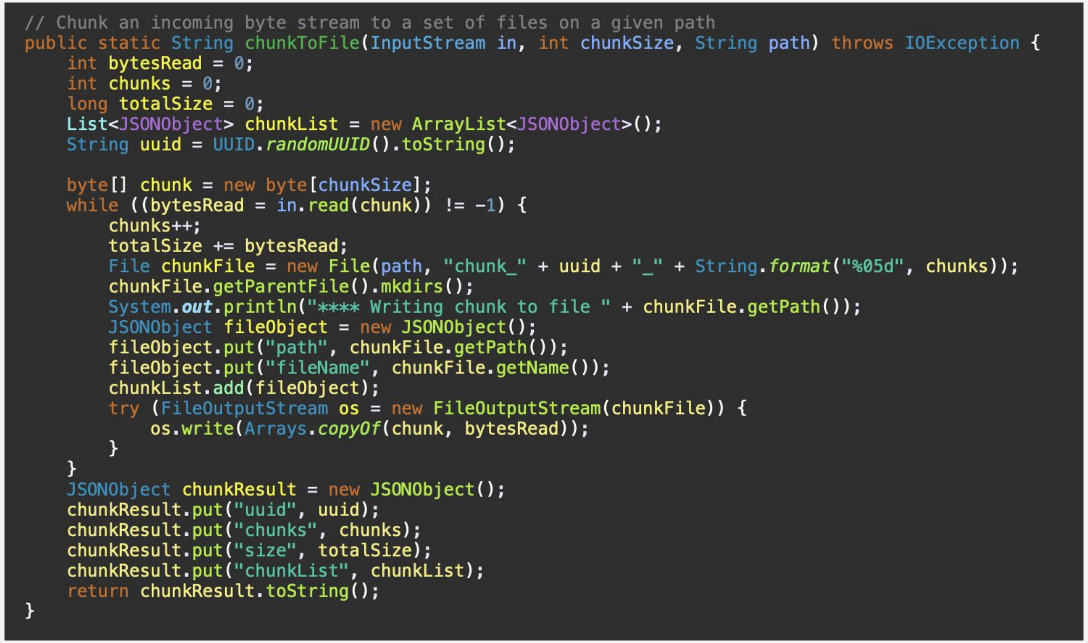
MuleSoft

Java code iterates through the incoming streamed binary file

Stores the file in chunks sized to be easily processed in memory

Returns a list of chunks (files) stored in the mule replica (application) file system

For RTF, storage comes from underlying operating system



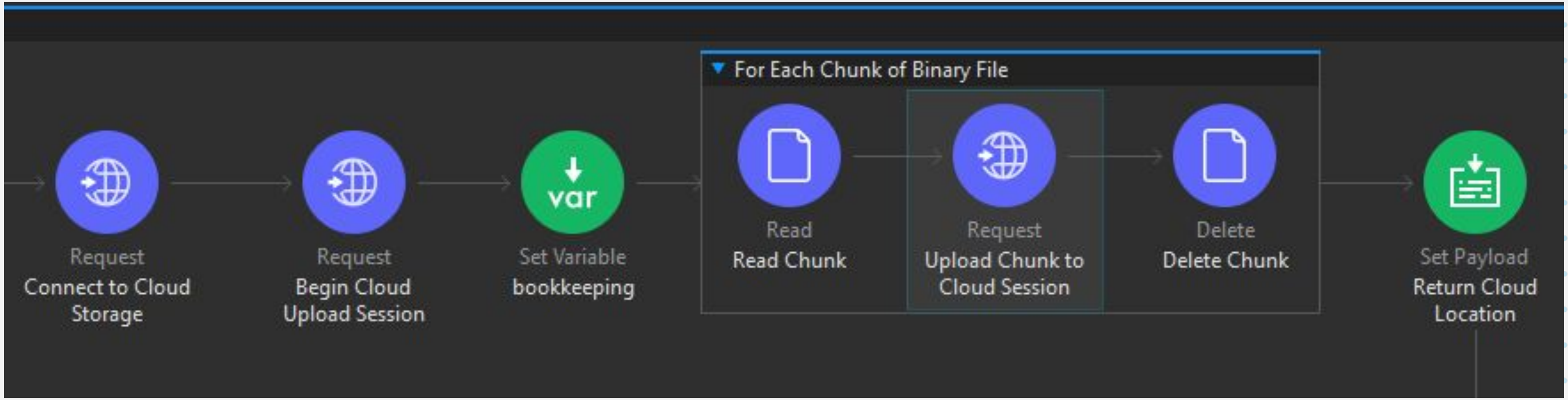
```java
// Chunk an incoming byte stream to a set of files on a given path
public static String chunkToFile(InputStream in, int chunkSize, String path) throws IOException {
    int bytesRead = 0;
    int chunks = 0;
    long totalSize = 0;
    List<JSONObject> chunkList = new ArrayList<JSONObject>();
    String uuid = UUID.randomUUID().toString();

    byte[] chunk = new byte[chunkSize];
    while ((bytesRead = in.read(chunk)) != -1) {
        chunks++;
        totalSize += bytesRead;
        File chunkFile = new File(path, "chunk_" + uuid + "_" + String.format("%05d", chunks));
        chunkFile.getParentFile().mkdirs();
        System.out.println("**** Writing chunk to file " + chunkFile.getPath());
        JSONObject fileObject = new JSONObject();
        fileObject.put("path", chunkFile.getPath());
        fileObject.put("fileName", chunkFile.getName());
        chunkList.add(fileObject);
        try (FileOutputStream os = new FileOutputStream(chunkFile)) {
            os.write(Arrays.copyOf(chunk, bytesRead));
        }
    }
    JSONObject chunkResult = new JSONObject();
    chunkResult.put("uuid", uuid);
    chunkResult.put("chunks", chunks);
    chunkResult.put("size", totalSize);
    chunkResult.put("chunkList", chunkList);
    return chunkResult.toString();
}
```

What do you do with “Chunks”?

Most cloud storage solutions allow files to be sent in parts

Sharepoint (OneDrive) supports this option



Considerations

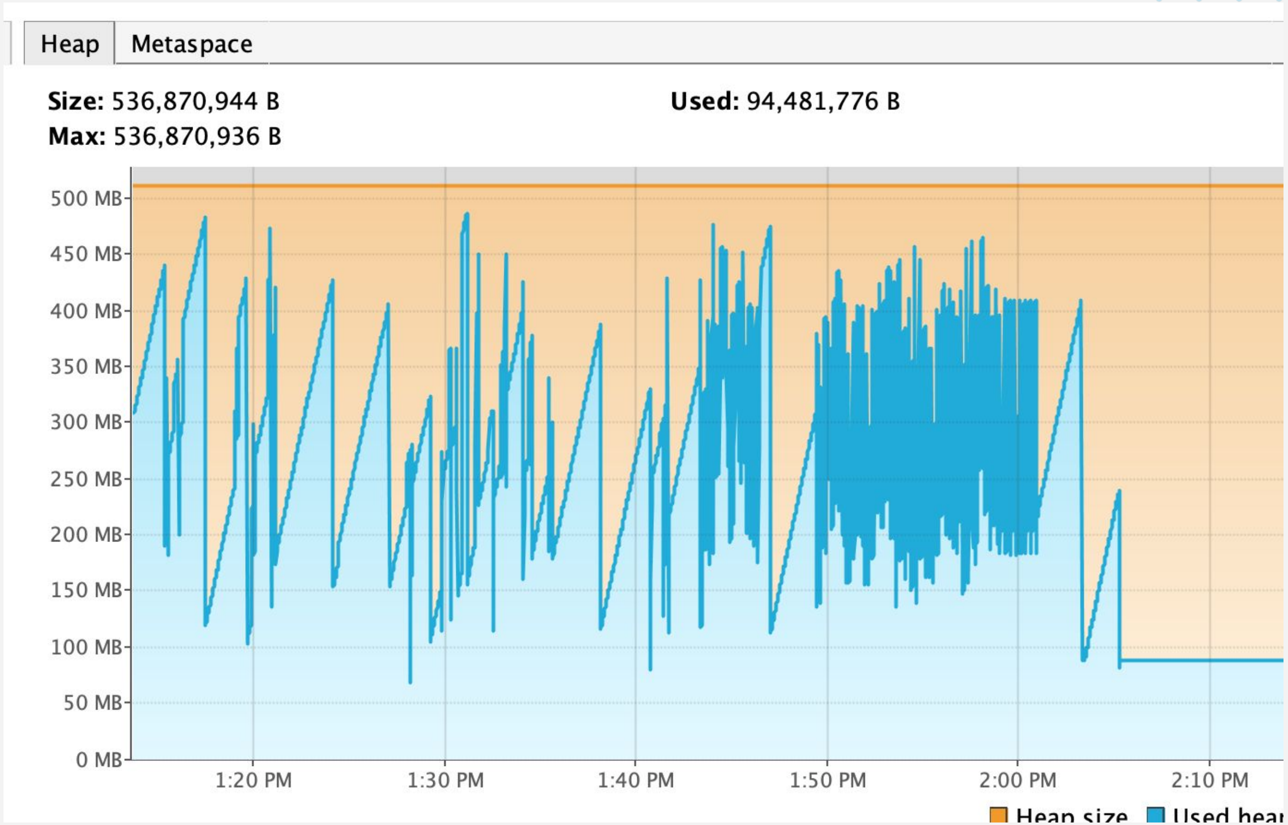
If the binary file is part of a complex structure (payload.attachments[0])

- Dataweave cannot be used to easily isolate the binary data
- Java must be used to scan the InputStream

MuleSoft

Demo

Heap Consumption While Processing 2GB Files



Size: 536,870,944 B
Max: 536,870,936 B
Used: 94,481,776 B