Root-Cause-Analysis (RCA) Report

Author:
M. Yauri M. Attamimi
Date:
19/08/2024 07:21

1. Executive Summary

This report documents the investigation and resolution of the `OutOfMemoryError: Java heap space` issue encountered in a Mule application.
The application consists of two flows, each with a scheduler running every 1000 milliseconds.
The first flow retrieves customer records from a database and publishes them to a queue, while the second flow consumes the record from the same queue and logs it.

2. Problem Description

The mule application encountered an `OutOfMemoryError: Java heap space` during operation. This error indicates that the application ran out of available memory, causing it to fail.

3. Steps to (Re)produce

a. Import the application in Anypoint Studio:
   File -> Import -> Anypoint Studio/Packaged mule application (.jar)
b. Set the heap size for the application:
   Menu Run > Run Configurations… -> Locate ‘Mule Application’ on the left -> Select the application -> Click on the ‘Arguments’ tab and enter the following in the ‘VM arguments’ section: -Xms50m -Xmx50m
c. Click ‘Run’.
d. Leave the application running while checking the console log.

4. Root Cause Analysis

a. Insufficient Heap Size
Description: The allocated heap size was not sufficient to handle the volume of data being processed.
Evidence: Memory usage exceeded the maximum heap size, leading to the `OutOfMemoryError`.

b. **Inefficient Data Processing**
Description: The application was processing too much data at once, leading to high memory usage.
Evidence: We might use some profiling tools (e.g., YourKit, JProfiler) to identify the data processing logic as the primary consumer of memory.

c. **Lack of Backpressure**
Description: The schedulers were generating data faster than it could be processed, leading to a buildup of data in memory.
Evidence: The queue size increased steadily over time, indicating that data was being produced faster than it could be consumed.

5. Recommended Solutions

a. Increase Heap Size
This will be the most straightforward solution by allocating more memory to the JVM to handle the increased load.
Implementation: Modify the JVM/VM arguments to increase the heap size. E.g., `-Xms512m -Xmx1024m`.

b. Optimize Data Processing
In some cases increasing the heap size won't always solve the issue. If that's the case, we might consider optimizing the data processing by implementing batch processing or streaming to reduce the memory consumption.
Implementation: Modify the flows to process data in smaller chunks.

c. Implement Backpressure
This particular solution can be considered in a case when the consumer is too slow to proceed the data being consumed from the queue while the producer is too fast. Therefore we can use flow control mechanisms to manage the rate of data processing.
Implementation: Configure the message queue to use flow control features.

d. Tune Garbage Collection
Optimize GC settings to improve memory management.
Implementation: Modify the JVM/VM arguments to tune garbage collection. E.g., `-XX:+UseG1GC -XX:MaxGCPauseMillis=200`.