MATLAB Logging Overhaul and RabbitMQ Integration

Tomasz Olejarczuk

Fontys

20-06-2024

Contents

MATLAB Logging Overhaul and RabbitMQ Integration

1. Summary

This document details the complete rewrite of the MATLAB logging mechanism and its integration with RabbitMQ. Previously, logging was handled through file writing and console output, which led to data inconsistencies and application crashes due to file-handling issues. The new implementation leverages RabbitMQ as a message broker, allowing for asynchronous and robust logging. This enhancement improves logging reliability, maintainability, and opens avenues for future log analysis and monitoring capabilities. The new logger uses the `mlog` logging utility from MATLAB.

2. Introduction

2.1. Background

The initial logging approach in MATLAB was flawed and problematic:

1. **File-Based Logging:** Writing logs to files within the MATLAB container was prone to errors, especially if the file system was unavailable or permissions were incorrect.
2. **Console Output:** Printing logs to the console was not sustainable for production environments and made it difficult to collect and analyze log data.
3. **Data Modification Risks:** The logging process could modify the data being processed by the MATLAB algorithm, leading to unexpected results and errors.
4. **Application Crashes:** If the logging process failed (e.g., due to file creation errors), it could crash the entire application.

2.2. Objectives

The primary objectives of this rewrite were to:

- **Eliminate File System Dependency:** Remove the reliance on writing logs to files within the MATLAB container.
- **Decouple Logging:** Separate the logging mechanism from the core MATLAB algorithm logic using object-oriented principles.
- **Introduce RabbitMQ:** Utilize RabbitMQ as a message broker to handle log transmission asynchronously and reliably.

- **Prevent Data Modification:** Ensure that the logging process does not alter the data being processed.
- **Improve Reliability and Error Handling:** Guarantee that logging failures do not crash the application and implement robust error handling.

3. Methodology

3.1. Architecture

The revised architecture introduces RabbitMQ as a central component for log transmission:

1. **MATLAB Logging:** The MATLAB algorithm sends log messages to a designated RabbitMQ exchange.
2. **RabbitMQ Routing:** The exchange routes log messages to a specific queue based on predefined rules.
3. **API Wrapper Consumption:** The C# API wrapper consumes messages from the queue, processes them (stores them in the database).

3.2. Implementation

- **MATLAB Code:**
  - The `LoggingMaster` class was created to encapsulate logging functionality, decoupling it from the core algorithm.
  - Logging methods (`Log`, `LogSuccess`, `LogStart`, `LogError`) were implemented to handle different log levels and formats.
  - The RabbitMQ .NET client library was used to publish log messages (encoded as JSON) to the exchange.
  - The `mlog` logger was used to write logs to the console, providing immediate feedback during development and testing.
- **API Wrapper:**
  - The wrapper was updated to consume log messages from the RabbitMQ queue.
  - Error handling and retry logic were implemented to ensure robust message delivery and prevent application crashes.
- **RabbitMQ Configuration:**
  - A RabbitMQ exchange and queue were created and configured for log message routing.
  - The Docker Compose file was updated to include the RabbitMQ container.

4. Results & Insights

4.1. Key Outcomes

- **Improved Modularity and Maintainability:** The MATLAB algorithm's logging is now completely decoupled from the core logic, leading to a cleaner and more maintainable codebase.
- **Enhanced Reliability:** RabbitMQ ensures reliable log delivery, even if the API wrapper or other components are temporarily unavailable.
- **Elimination of File System Dependency:** The removal of file-based logging eliminates the risk of application crashes due to file-handling issues.
- **Data Integrity:** The logging process no longer interferes with the data being processed by the algorithm, ensuring data consistency and accuracy.
- **Scalability:** The system can now seamlessly handle logs from multiple MATLAB instances without complex connection management.
- **Standardization and Readability:** The use of `mlog` logger provides a standardized format for log messages, improving readability and facilitating analysis.

4.2. Performance Impact

The introduction of RabbitMQ did introduce a minor overhead in terms of message serialization and network transmission. However, this overhead is minimal compared to the significant benefits gained in reliability, scalability and maintainability.

5. Insights and Conclusion

The successful rewrite of the MATLAB logging mechanism and its integration with RabbitMQ represents a major step forward in addressing the critical flaws of the previous implementation. By eliminating file system dependencies, decoupling logging, and ensuring reliable message delivery, the project has achieved a more robust, scalable, and maintainable logging solution. The use of RabbitMQ also opens up possibilities for future enhancements, such as implementing log aggregation and analysis tools for deeper insights into system behavior.