International Journal of Computer Trends and Technology
ISSN: 2231–2803 / https://doi.org/10.14445/22312803/IJCTT-V72I1P106

Volume 72 Issue 1, 35-39, January 2024

Original Article

Guaranteed Delivery in Enterprise Integration Architecture with IPaaS Environment

Swapnil Vaidya

Principal Architect and Solution Engineer, MuleSoft, A Salesforce Company, Virginia, USA.

Received: 20 November 2023 Revised: 29 December 2023 Accepted: 15 January 2024 Published: 24 January 2024

Abstract *- Enterprise Architecture and Integration are becoming very critical fields in the computer science and information technology industry. Especially with the cutting-edge technologies of IPaaS (Integration Platform As A Service), the introduction and adoption of Artificial Intelligence, IoT, and many more trends, data from different dispersed and siloed systems must be integrated to give a 360-degree view of related objects. There are over 65 integration patterns [1], and a combination of these patterns can be used to bring data close to each other. The most important thing that developers and architects need to keep in mind is that the functional and non-functional requirements to solve any problem play an important role in choosing the right design and pattern to solve the problem. This article explains the guaranteed delivery pattern in IP, its importance, and various retry options at the component level with MuleSoft CloudHub IPaaS [2] and Anypoint MQ [3] as a demonstrative toolset.*

Keywords *- Enterprise architecture, Guaranteed delivery pattern, IPaaS, Message queue, MuleSoft.*

1. Introduction

Regarding integration, a crucial aspect of ensuring reliability is the guaranteed delivery of data from source to target systems. Enterprise integration involves connecting diverse software applications and systems within an organization to facilitate seamless collaboration. The goal is to enhance efficiency, streamline business processes, and ensure smooth data flow across various departments or functions. In the current era, organizations, including those in secure sectors like healthcare and banking, increasingly embrace a cloud-first strategy. This trend underscores the widespread adoption of Software-as-a-Service (SaaS) enterprise software in an organization's technical stack alongside existing legacy and on-premises tech stacks.

However, dispersed networks and systems pose the risk of potential data loss due to unplanned outages. According to a study conducted by Business Wire [4], 82% of companies across all sectors have experienced at least one unplanned downtime outage in the past three years, with an average of two outages. Another study [5] reports that the average downtime per hour across all businesses costs $260,000.

Considering the impact of data and transaction loss on businesses, it becomes crucial to have reliable integration for processing data from one system to another. Currently, due to the pressure to deliver projects on time and a lack of sufficient resources, many organizations and teams opt to complete functional requirements while keeping non-functional requirements, such as guaranteed delivery, on the backlog. To raise awareness and emphasize the importance of reliable delivery, this article focuses on guaranteed delivery patterns from the learnings from different industries and discusses essential components defining this pattern. Additionally, based on the chosen tools for implementing these patterns, the article explores multiple levels to set up a retry mechanism. The focus will be on brainstorming and presenting some of these options using MuleSoft, a leading IPaaS in the market.

2. Understanding Guaranteed Delivery

In crafting any solution architecture or design, the focal point consistently revolves around fulfilling both functional and non-functional requirements. Functional requirements succinctly delineate the actual functionality, such as defining a service responsible for receiving orders from the CRM system and updating the ERP systems. On the other hand, non-functional requirements encapsulate the nature and behaviour of the solution, encompassing aspects like response time, reliability, error handling, and exception management.

The concept of guaranteed delivery is among the array of non-functional requirements crucial for rendering any solution reliable. This becomes especially pertinent for solutions that involve alterations (insertions, updates, or deletions) in various systems or the context of event-driven architecture [1], ensuring that events or requests originating from the source are unequivocally delivered to the target system, thereby solidifying the solution's reliability. But what factors contribute to the potential loss of such events? Two primary reasons can be identified:

- Technical Error – Errors occurring during the processing of requests within the solution, stemming from issues such as network connectivity, memory constraints, or CPU-related challenges in the application, can be addressed through retries without necessitating modifications to the overall solution.
- Business Error – Errors arising from business rule validation or data exceptions that mandate corrections to request data present another category of potential event loss.

2.1. Components

Failures are an unavoidable aspect, whether stemming from technical glitches or business-related factors, as previously discussed. Crafting a system with guaranteed delivery involves a meticulous process of catching exceptions during processing and implementing retries for requests or events, thus defining the system's reliability. Broadly speaking, the design of any retry pattern for guaranteed delivery necessitates incorporating or combining the following components at a higher level.

2.1.1. Flexible IPaaS Tool

In the context of this article, the focus centres on exploring guaranteed delivery and retry mechanisms tailored for solutions primarily leveraging SaaS or cloud-first enterprise systems. To maintain simplicity, the discussion is narrowed down to an Integration Platform as a Service (IPaaS) toolset, known for its proficiency in seamlessly integrating both SaaS and on-premises systems. While numerous vendors populate the market, the article uses MuleSoft to showcase diverse capabilities and approaches. MuleSoft stands out as a leader in the Gartner quadrant, solidifying its position as a prominent IPaaS provider [6].

2.1.2. Messaging Queue Tool

A messaging queue, commonly known as a message queue or message broker, serves as a communication mechanism employed in distributed systems to facilitate the exchange of messages among various software components. The fundamental goal of a messaging queue is to streamline asynchronous communication, effectively separating the sender and receiver of messages and enabling them to function independently.

Multiple providers extend message queue services. Within the context of this article, Anypoint MQ [3], a robust API-based MQ solution that seamlessly integrates with IPaaS providers, has been chosen to demonstrate architecture with a messaging queue.

3. Approach

Before discussing the retry approaches, let us define core principles that will be at the centre of the approach to be chosen for the scenarios.

3.1. Core Principles

Below are guiding principles based on lessons learned from various industries:

- Treat business failures and technical system failures differently.
- When determining the Service Level Agreement (SLA) for synchronous calls, account for the time lag caused by the retry mechanism.
- Retries should be integrated into the implementation for all use cases involving guaranteed message delivery.
- Move time-consuming retries (such as exponential or waiting retries) to an asynchronous process to reduce dependencies on resources.
- From a traceability perspective, ensure that correlation and trace ID are available with the actual payload during the retry.
- Ensure that the retry mechanism is not infinite.
- Ensure proper exceptions and necessary alerting are handled during the retry process.

3.2. Retry Approaches

The retry mechanism plays a crucial role in ensuring the successful processing of messages, guaranteeing either delivery or triggering exception notifications within a set number of retries. Various scenarios require diverse retry approaches, such as connectivity failures, backend system downtimes, throttling limit breaches, or certain business errors.

These approaches aim to secure message delivery or initiate notifications within a finite retry framework. The below sub-sections explore specific scenarios, discuss approaches to handle them, and provide examples of relevant toolsets to be used in the solutions.

3.2.1. Handling Transient Errors

Scenario

It is very common to encounter transient errors, such as network glitches while connecting to the end system or API, intermittent failures, and so on, which need a quick retry.

Best fit

Synchronous and Asynchronous processes that need simple connectivity retry.

Approach

- Utilize component or connector-specific options provided by the toolset. For instance, in the context of IPaaS MuleSoft, leverage the Until-Successful scope with the connector, as illustrated in the Anypoint Studio snippet below (Figure 1). In this example, the HTTP request connector is encapsulated within the Until-Successful scope, configured with finite retries. This setup facilitates connecting to an external HTTP API provider using the HTTP requester connector.

- Have finite retries within the defined SLA (service level agreement). (For example, retry 3 times within 1 second so the SLA of 2 seconds can be met)

Limitation

This is not a comprehensive retry. Another approach should be chosen for comprehensive retries.

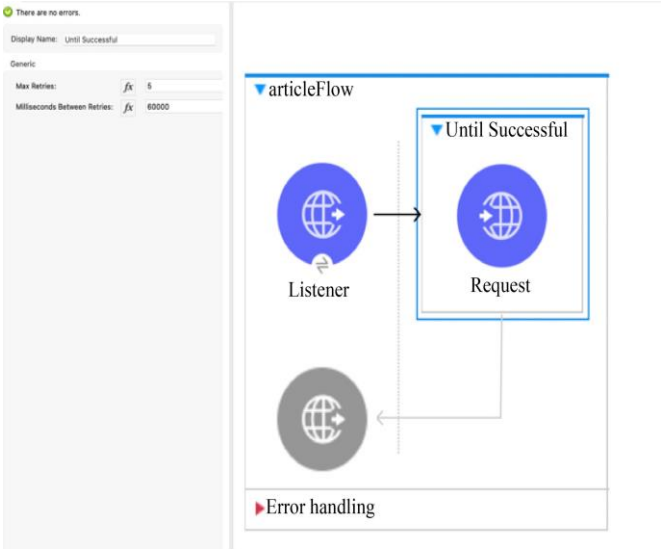

Fig. 1 MuleSoft HTTP connector with Until-Successful

3.2.2. Short Retry Low Volume Failures

Scenario

There might be connectivity issues with the end systems due to maintenance, the end system being down, provider certificate expiry, client certificate expiration, etc. These situations may require some downtime before the issue can be resolved.

Best fit

- Synchronous integration requires guaranteed message delivery or reconciliation, either through retry mechanisms or notifications.
- Asynchronous processes, such as incremental loads involving a few thousand records, require guaranteed message delivery or reconciliation through retry mechanisms or notifications.
- Performance during the retry is a key consideration.
- Number of messages not exceeding a few thousand of messages per use case.

Approach

- Utilize options specific to the component or connector provided by the toolset. For instance, with IPaaS MuleSoft, employ the Until-Successful scope with the connector, as illustrated in the Anypoint Studio snippet below (Figure 1). This helps confirm that it is not a persistent problem.
- If the processing still fails, have a process that
  - Notify the consumer via error response and message that talks about the retry.
  - Send the message to the message queue like Anypoint MQ.
  - Use exponential retry or wait for retry pattern with a finite number of retries and interval times and the ability to push message to dead letter queue or error queue. Figure 2 below shows a sample exponential retry design for MuleSoft and Anypoint MQ.
  - Send notifications or handle messages in the dead letter queue.

Limitation

Based on used tool sets like MQ and their ability to handle inflight messages, these patterns need to be used in low-volume rather than high-volume use cases.

3.2.2. Short Retry High Volume Failures

Scenario

In scenarios involving ETL or high-volume use cases, where processing millions of records is necessary, issues may arise in connecting to the end system. These problems could be attributed to maintenance, the end system being down, provider certificate expiry, client certificate expiration, credentials expiration, and similar factors. Resolving such issues might require some downtime.

Best fit

- Asynchronous processes, such as the initial ETL load use cases requiring the processing of millions of records, demand guaranteed message delivery or reconciliation through retry mechanisms or notifications.
- Guaranteed delivery is a key, and performance during retry is a secondary consideration.
- Use cases process millions of messages.

Approach

- Utilize the out-of-the-box reconnection strategy of the component or connector to reaffirm that it is not a persistent problem. For instance, MuleSoft's HTTP connector, illustrated in Figure 3 below, includes a reconnection strategy that can be employed.

- If the processing still fails, have a process that
  - Send the message to the persistent Message Queue.
  - Implement an exponential retry or wait-for-retry pattern with a finite number of retries, interval times, and the ability to move the message to a dead letter queue or error queue. Figure 2 above illustrates a sample exponential retry design for MuleSoft and Anypoint MQ.
  - Send notifications or handle messages in the dead letter queue.

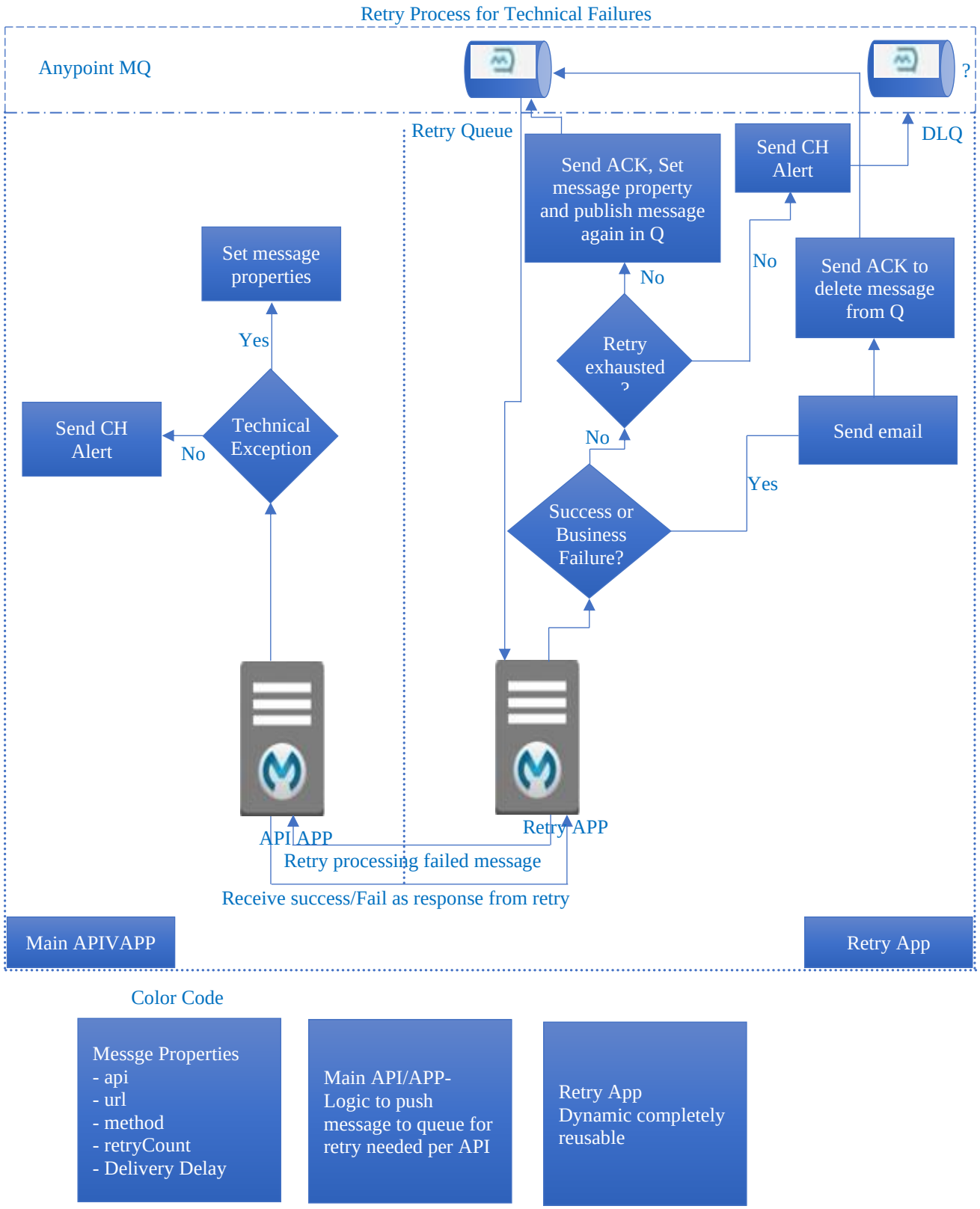

Fig. 2 Sample retry process for technical failures

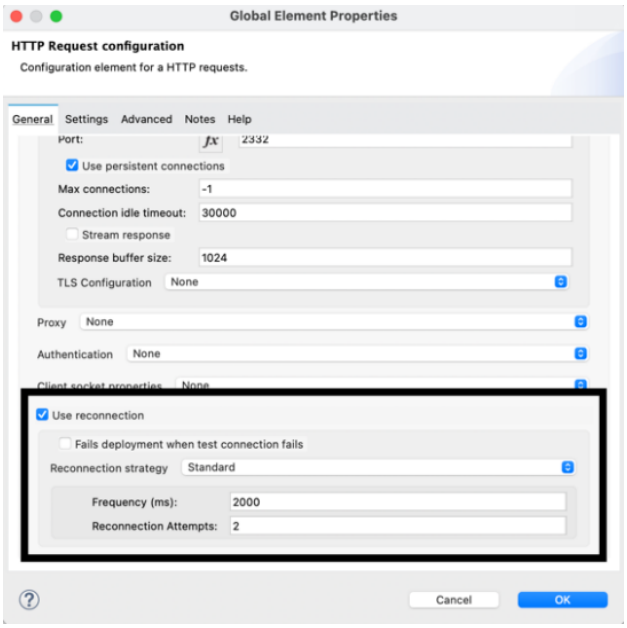

Fig. 3 Sample reconnection strategy of MuleSoft connector

3.2.3. Business Exceptions

Scenario

Failures attributed to business rules or data validation pose challenges for automated retries, as they often vary based on use cases. However, when ensuring guaranteed message processing and delivery becomes a part of Non-Functional Requirements (NFR), a step-by-step retry approach can be employed when applicable. Automated retries for business failures are not always mandatory, and it may suffice to send appropriate business failure notifications or reconciliation reports to the consumer.

Best fit

- Upon failure, the sender cannot resend the corrected messages.
- When business validation type failure is due to failure of dependent lookup events, for example, an account lookup needs to be done to update the address in Salesforce, and if the account is unavailable.
- Applicable for both Synchronous and Asynchronous use cases.

Approach

- Capture the specific business validation exception that needs retry.
- Based on the volume of the use case, send the messages to the appropriate message queue tools for retry.
- As agreed upon with the various stakeholders from the business and project, notify the business validation failures.
- Business users fix the issue and send the message for reprocessing. (This can also be determined by the use case requirements to write an additional retry process that listens to the messages in the message queue).

4. Impact

Based on practical learnings from direct industry experience, the following impacts have been observed when best-fit-for-purpose approaches are used to solve specific retry needs for guaranteed delivery:

- Cost-effective solutions: Not all use cases require dedicated resources to design and implement retry mechanisms. Depending on the nature of the use case, less costly and out-of-the-box components can be used (see Figure 1).
- Reduced operations effort: Tackling exceptions and retrying them helps with better tracking and providing timely actions, either retry or notification, which aids in maintaining continuous synchronization of data between source and destination systems.
- Business continuity: The impact due to downtime would be absorbed by the guaranteed delivery, resulting in little to no impact on business.

5. Conclusion

Guaranteed delivery is a critical yet often overlooked non-functional requirement that significantly influences the reliability and stability of any integration. In a cloud-first environment, implementing retries requires special attention and different strategies. The factors and some variants discussed in this article will assist various organizations in selecting the right approach and toolset needed to make their integration in the IPaaS environment reliable.

References

[1] Gregory Hohpe, and Bobby Woolf, *Enterprise Integration Patterns*, Pearson India, pp. 1-737, 2003. [Google Scholar] [Publisher Link]

[2] CloudHub Overview, MuleSoft. [Online]. Available: https://docs.mulesoft.com/cloudhub/

[3] Anypoint MQ Overview, MuleSoft. [Online]. Available: https://docs.mulesoft.com/mq/

[4] Human Error is More Common Cause of Unplanned Downtime in Manufacturing than any other Sector, According to New Research, Businesswire, 2017. [Online]. Available: https://www.businesswire.com/news/home/20171106006370/en/Human-Error-Common-Unplanned-Downtime-Manufacturing-Sector

[5] Stat of the Week: The (Rising!) Cost of Downtime, Aberdeen Strategy & Research, 2016. [Online]. Available: https://www.aberdeen.com/blogposts/stat-of-the-week-the-rising-cost-of-downtime/

[6] Gartner Names MuleSoft a Leader, MuleSoft. [Online]. Available: https://www.mulesoft.com/lp/reports/gartner-magic-quadrant-ipaas