A picture containing art, amber

Description automatically generated



**RELAY SERVICE**

**FUNCTIONAL SPECIFICATIONS DOCUMENT**

**Version 0.1**

**22.04.2025**

**Document Information**

|  |  |
| --- | --- |
| **Project** | TAZAMA DEVELOPMENT |
| **Document Title** | RELAY SERVICE |
| **Prepared by** | UME SAUDA GHANYANI – SENIOR BA @ PAYSYSLABS |
| **Version Number** | 1.0 |
| **Version Date** | 22.04.2025 |

**Document Revision History**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Author** | **Version** | **Change Reference** |
| 22.04.2025 | UME SAUDA GHANYANI | 0.1 | BASELINE VERSION SHARED WITH TAZAMA FOR REVIEW |
| 13.05.2025 | UME SAUDA GHANYANI | 0.2 | FEEDBACK VERSION SHARED WITH TAZAMA FOR REVIEW |
|  |  |  |  |
|  |  |  |  |

Table of Contents

[Introduction 4](#_Toc55114163)

[1.1 Purpose 4](#_Toc213274172)

[1.2 Scope of the Document 4](#_Toc457707618)

[1.2 Intended Audience 4](#_Toc1287738203)

[System Overview 5](#_Toc934792743)

[2.1 Tazama Transaction Monitoring System Overview 5](#_Toc345157786)

[2.2 Interaction with Relay Service 5](#_Toc447483692)

[Functional Requirements 6](#_Toc1354395201)

[3.1 Objective and Functionality of Relay Service 6](#_Toc690279340)

[Message Subscription 6](#_Toc1127874105)

[Message Relay 7](#_Toc1613090725)

[Error Handling and Logging 9](#_Toc1485363159)

[3.2 Relay Service Setup 10](#_Toc363730418)

[3.3 Relay Service Use Cases 15](#_Toc981157405)

[Use Case 1: Known to Known (Encoded -> Encoded) 16](#_Toc1561658251)

[Use Case 2: Known to Known (Encoded -> Raw) 24](#_Toc807404804)

[Use Case 3: Known to UnKnown (Encoded -> Raw) 32](#_Toc1653168118)

[3.4 Relay Service Integrations 39](#_Toc885336631)

[Architecture Relay Service: 40](#_Toc1097401575)

[3.5 Relay Service Assumptions 41](#_Toc596155690)

[Use Case 1 and Use Case 2: One-to-One 44](#_Toc546467418)

[UC3: One-to-Many (Theoretical) 53](#_Toc1876648843)

# Introduction

## 1.1 Purpose

This document defines the functional requirements for the Tazama Relay Service, which bridges communication between the Tazama Monitoring Service (TMS) and client applications for transaction processing and analysis.

## 1.2 Scope of the Document

This document serves as the foundation for developing, deploying, and maintaining the Tazama Relay Service. It will define the core functionalities, performance expectations, and configuration requirements.

## 1.2 Intended Audience

The intended audience includes developers, system administrators, and stakeholders involved in the setup, maintenance, and use of the Tazama system.

# System Overview

## 2.1 Tazama Transaction Monitoring System Overview

Tazama is a rules-based forward-chaining inference engine that processes transaction data in real-time. It identifies fraudulent and money-laundering behaviors, summarizing results into typologies and triggering appropriate actions (alerts, blocking transactions, etc.).

## 2.2 Interaction with Relay Service

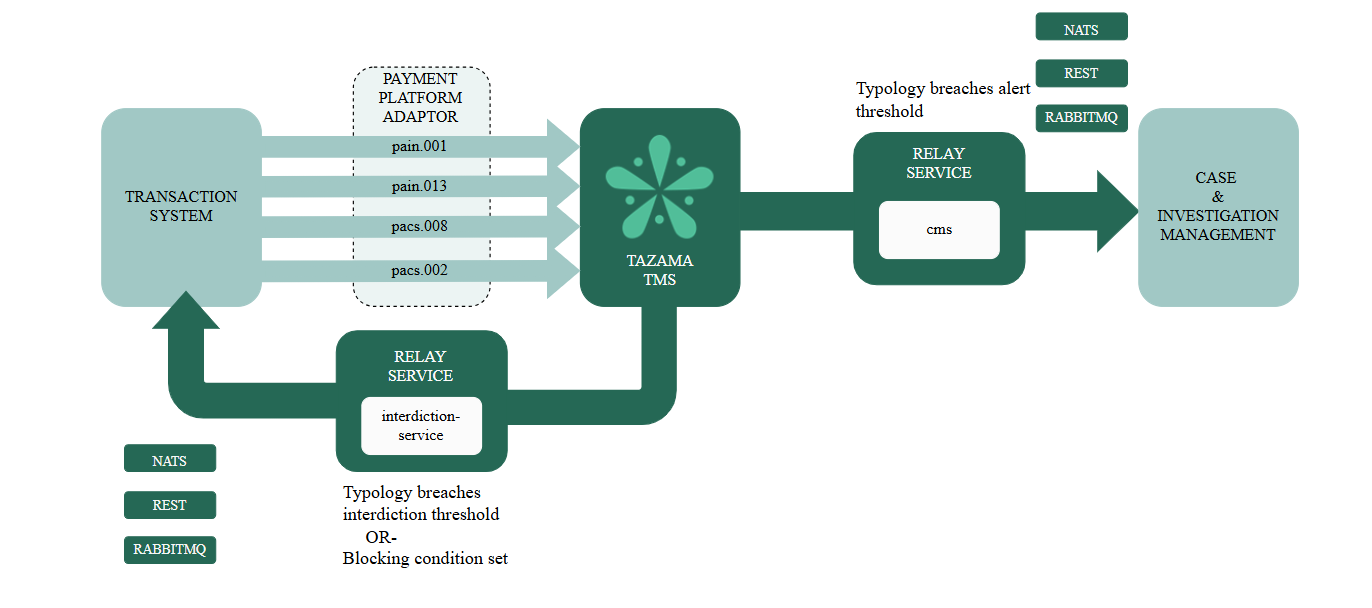
The Tazama Relay Service facilitates the forwarding of messages between the TMS and external systems such as payment systems, case management systems (CMS), and message queues like Kafka and NATS.

# Functional Requirements

## 3.1 Objective and Functionality of Relay Service

The Tazama Relay Service is a special component within the Tazama system that acts as a bridge between the core transaction monitoring system and external systems. It works by listening for events inside Tazama (via NATS messaging) and then sending those events to other systems, like message queues or REST APIs.

This service helps communicate real-time alerts from Tazama’s fraud and money-laundering detection processes to external services. For example, it can send a fraud alert to a case management system or send instructions to block a transaction. By doing this, the Relay Service keeps Tazama’s internal processes separate from the outside systems, making it easier to connect with other services without giving them direct access to Tazama’s internal workings. This way, Tazama can notify other systems about important events, like potential fraud, without those systems needing to be deeply integrated into Tazama itself.



### Message Subscription

The Relay Service is configured to subscribe to a specified NATS server and subject at startup. The subject is explicitly defined based on environment variables set by authenticated authorized users only during deployment.

The subscription enables the Relay Service to listen to a real-time stream of messages published to the NATS subject. This allows for real-time processing of transaction data.

Upon service startup, the startRelayServices () function is invoked to initialize the message subscription. This function ensures that the Relay Service is ready to receive messages from the configured input source.

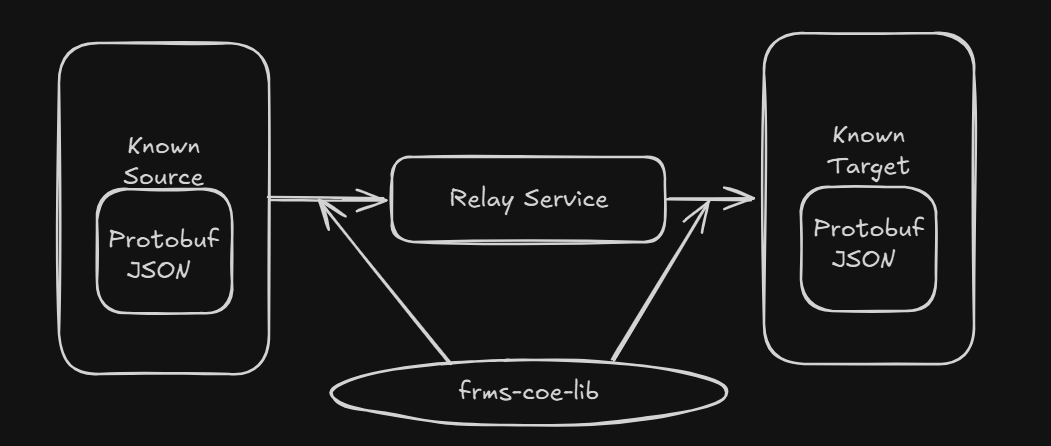
frms-coe-startup-lib is leveraged to handle the initialization of the subscribers and the configuration of the input source.

### Message Relay

After a message is published to the configured NATS subject, the Relay Service consumes the message and initiates the forwarding process. Acting as a central routing component, ~~i~~t determines the appropriate destination—whether a service or messaging queue—based on the destination type defined in the configuration. The message is then forwarded to the correct endpoint, in line with the environment settings and supported by the frms-coe-lib.

The relay service supports two main scenarios, and each scenario includes multiple use cases:

* **Known to Known**: Both source and destination systems are explicitly defined in the Tazama ecosystem
  + Use Case 1: Encoded -> Encoded



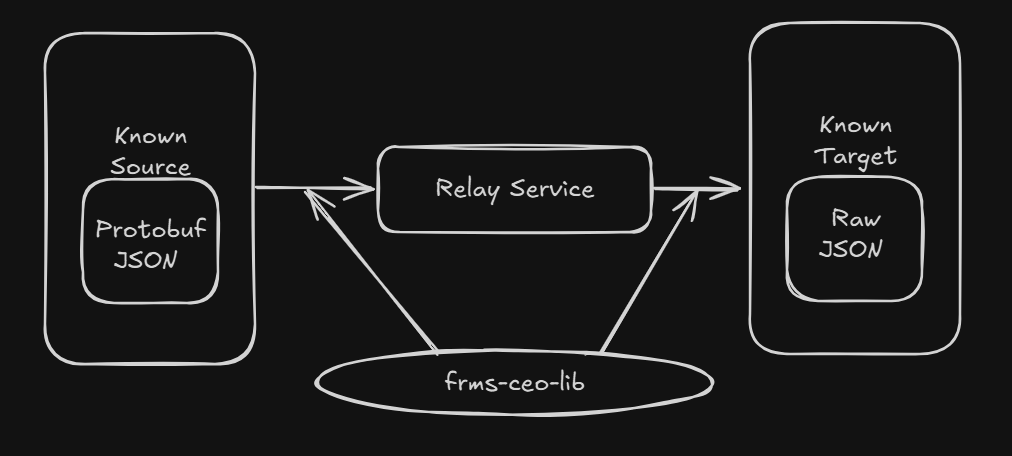
The source and destination both support encoded formats, specifically protocol buffers

Message Relay Throughput:

The throughput benchmarked on total count of 60,000 sent messages for use case 1 is as follows

|  |  |
| --- | --- |
| Message Queues | Throughput (TPS) |
| NATS | 3000 |
| Rabbit MQ | 3333 |
| Rest API | 2649 |

* + Use Case 2: Encoded -> Raw



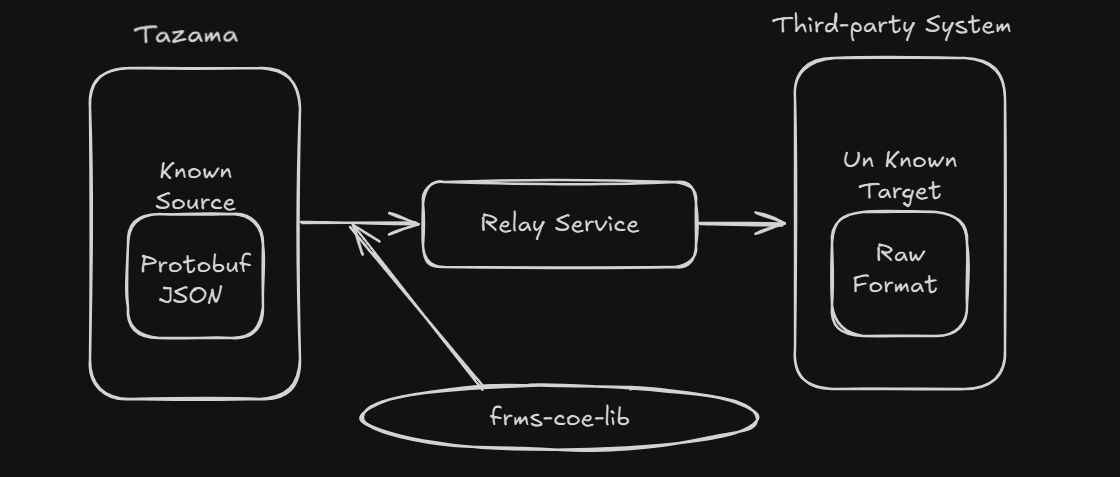
The source system sends an encoded Protobuf message, while the destination system expects a raw JSON payload.

Message Relay Throughput:

The throughput benchmarked on total count of 60,000 sent messages for use case 2 is as follows (the threshold may vary from 10 – 15%)

|  |  |
| --- | --- |
| Message Queues | Throughput (TPS) |
| NATS | 3000 |
| Rabbit MQ | 3333 |
| Rest API | 2649 |

* **Known to Unknown**: The source is predefined in the Tazama system, but the destination is determined ~~dynamically at runtime~~ ~~by the third-party system~~ destination is configured at deploy-time via environment variables. Although certain runtime behaviors—such as whether to decode the message to raw format—are determined dynamically based on these deploy-time settings, the destination itself is not dynamically selected at runtime by the third-party system. The relay service routes the message to the preconfigured destination, and once transmitted, its further handling is entirely managed by the target system and is outside Tazama’s control.
  + Use Case 3: Encoded -> Raw



A predefined internal system sends an encoded protobuf message, while the unknown third-party destination receives a raw JSON output ~~or some other data format (dynamically defined)~~.

Message Relay Throughput:

The throughput benchmarked for use case 3 is not applicable.

### Error Handling and Logging

The Relay Service logs shall capture error details, the originating service name, and the associated message ID to facilitate efficient issue resolution.

## 3.2 Relay Service Setup

Variables instantiating the Relay Service:Consumer Side Configuration:

* Startup Type: Configure the messaging protocol type of the consumer.Node Environment: Configure the deployment environment (dev, production, etc)CPU Limit: Determine CPU limit  
  Function Name: Configure the service name for context.
* Consumer Stream; Configure the consumer for the relay service.
* Consumer Server URL: Configure the consumer side server URL

Producer/Destination Type Configuration:

The environment variable for specifying the Producer/Destination shall be documented within each plug-in’s deployment guide. Plug-in deployments must define their own configuration parameters as per their functional requirements (e.g., Kafka requiring a topic-name, while REST APIs may not use this parameter).  
The integrator must reference the plug-in documentation to correctly apply the required settings. The integration process should not enforce a uniform configuration structure across all plug-ins, as configuration needs may vary depending on the target system or protocol. This approach ensures flexibility and prevents inconsistency across heterogeneous integration points.

* .

## 3.3 Relay Service Use Cases

### Use Case 1: Known to Known (Encoded -> Encoded)

#### Description:

This use case represents a simple pass-through flow, where the message is received in an encoded format (e.g., Protobuf) and is forwarded without any transformation to another system that also supports the exact same encoded messages. No decoding or reformatting is required by the Relay Service.

#### Purpose:

Used in scenarios where both the source and target systems are schema-aware and support structured binary communication for efficiency, such as within internal microservices or trusted infrastructure.

#### Supported Destination Endpoints:

* NATS – For lightweight internal messaging with high performance.
* REST API – For services that accept Protobuf payloads over HTTP.
* Kafka – For durable, event-stream based forwarding of encoded messages.
* Use Case 2: Known to Known (Encoded -> Raw)

#### Description:

This use case handles the scenario where a message is received in an encoded format (Protobuf) but needs to be decoded into raw JSON before being forwarded to the destination. This transformation is typically required when the destination system is not Protobuf-aware.

#### Purpose:

Ideal for integrations with downstream components that rely on raw JSON for processing, storage, or rule evaluation. Ensures format compatibility across heterogeneous environments.

#### Supported Destination Endpoints:

* NATS – For simple JSON-based consumers or subscribers.
* REST API – For HTTP-based services that require JSON in request bodies.
* Kafka – For publishing JSON events into data pipelines or logging streams.
* Use Case 3: Known to Unknown (Encoded -> Raw)

#### Description:

This use case involves determining the output format based on runtime conditions, message metadata, or recipient capabilities. The Relay Service would send the message in raw form at the time of forwarding, based on an environment variable set at the time of deployment.

#### Purpose:

The purpose of use case 3 is to send raw JSON file to the unknown destination.

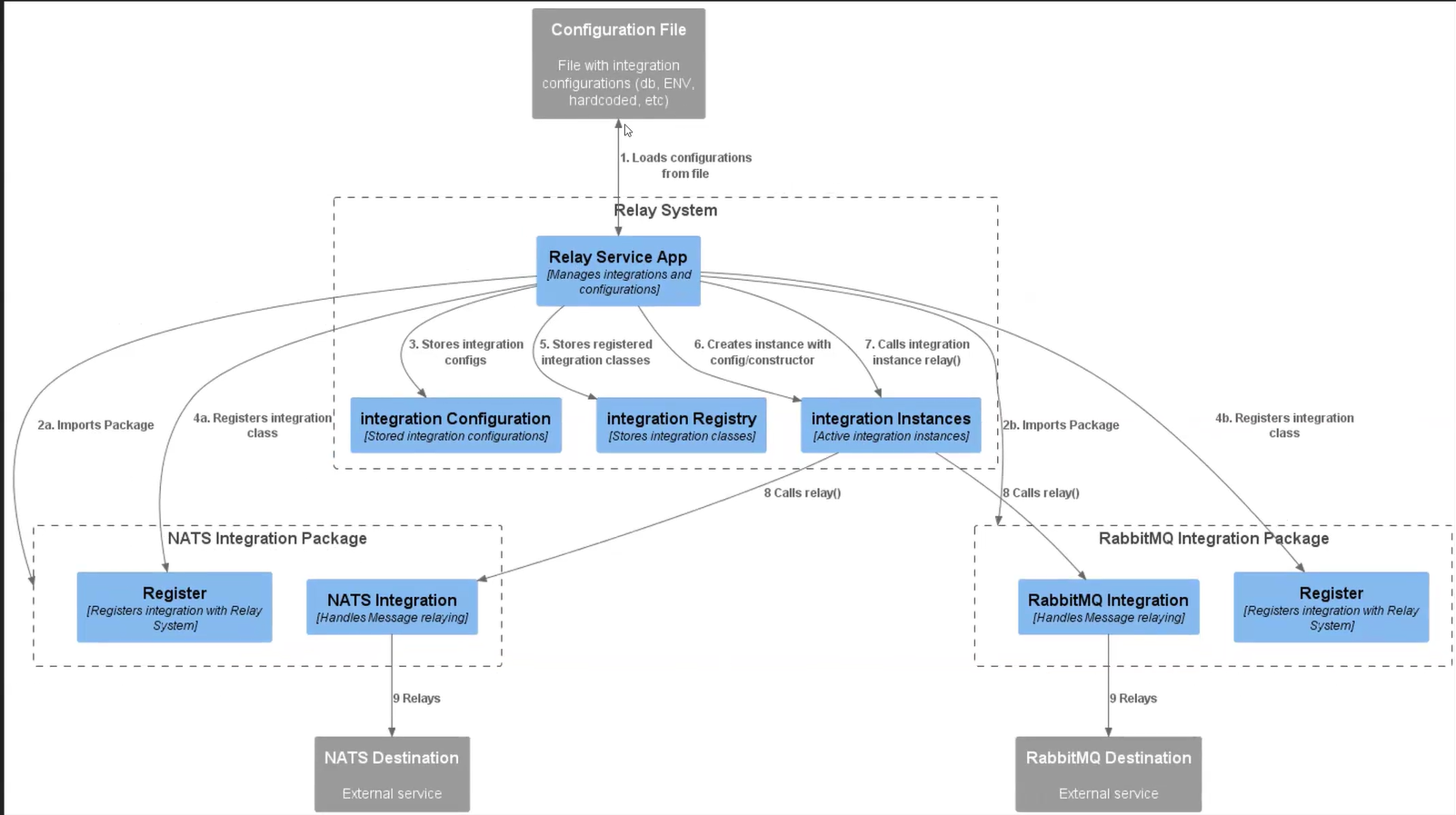
**Note:** *The internal execution flow for both Known-to-Known (Raw) and Known-to-Unknown scenarios is functionally identical within the Tazama system. The primary distinction lies in the intended actor and extensibility objective.*

* *Use Case 2 is designed for scenarios where both the source and destination systems are known to Tazama at deploy-time, and the routing and decoding logic is governed by the core system and pre-configured environment variables.*
* *Use Case 3, by contrast, targets scenarios where a third-party developer is expected to implement their own destination integration via a custom plug-in, aligned with their specific system requirements. In this context, the "user" of the use case is the third-party developer, not the Tazama Core or the destination system itself.*

## 3.4 Relay Service Integrations

The current Tazama message relay service requires the following improvements

### Plug-in Architecture Relay Service:



*Note: this diagram is just to support the below explanation for the relay service architecture*

The Relay Service supports a modular, plug-in-based integration framework that enables seamless message relaying to various external systems such as NATS, RabbitMQ, and others. This architecture ensures scalability, maintainability, and ease of adding new integration targets through a well-defined plugin interface.

### 1. Key Components

#### 1.1 Configuration File

* Stores integration configurations including database connections, environment variables, and hardcoded settings.
* Loaded at system startup by the **Relay Service App**.

#### 1.2 Relay Service App

* Acts as the central coordinator that manages integration configurations and plug-in lifecycle.
* Responsibilities:
  + Loads configuration from file.
  + Stores integration configs.
  + Stores registered integration classes.
  + Creates instances using the configuration and registered classes.
  + Invokes a relay() method on integration instances.

### 2. Plugin Lifecycle

#### 2.1 Integration Package (e.g., NATS, RabbitMQ)

Each plugin must consist of:

* An **Initiation** function that initiates the connection with the Relay Service.
* An **Integration** function that contains the logic for message relaying to the target system.

#### 2.2 Workflow

1. **Package Import**:
   1. Integration packages are installed and imported by the Relay Service based on the configurationn at the time of deployment (e.g., NATS, RabbitMQ).
2. **Instance Creation**:
   1. The Relay Service uses the stored configuration and registered class to create an instance of integration.
3. **Message Relay**:
   1. The instance's relay () method is invoked with message data.
   2. The plugin handles destination-specific logic to forward the message to the external service (e.g., topic for NATS, queue for RabbitMQ).

#### 3. Extensibility Requirements

* The system must support additional plugin packages without modifying the Relay Service core logic.
* Each plugin must follow the standard structure:
* New destination types must define only their specific configuration parameters.

### Authenticated REST API Integration:

When relaying messages to known REST API destinations, authentication must be enforced using tokenization mechanisms provided by the **Authentication Library** (Auth Lib). This ensures secure and standardized access control for pre-registered endpoints.

For unknown destinations, the responsibility for implementing appropriate authentication measures rests with the client. In such cases, the Relay Service shall forward the message without applying internal authentication, assuming the client has pre-configured the necessary authentication headers or tokens externally.

## 3.5 Relay Service Assumptions

The current design and implementation of the Relay Service are built on a few foundational assumptions regarding message routing and delivery patterns across use cases. These assumptions guide architectural boundaries and clarify what is in or out of scope for this release.

### Use Case 1,Use Case 2 and Use Case 3: One-to-One

* For Use Case 1 (Encoded to Encoded) and Use Case 2 (Encoded to Raw), the message follows a strict one-to-one pattern.
* This means:
  + Messages from the source system (e.g., NATS) will be routed to a single configured destination, based on the producer configuration.
  + The message will not be duplicated or forwarded to multiple endpoints in parallel.
* This simplifies routing logic and ensures deterministic behavior, making it easier to manage delivery guarantees, format expectations, and failure handling.
* Examples:
  + A Protobuf message from NATS is sent to a specific Kafka topic.
  + A decoded JSON message is sent to one REST API endpoint.