**RabbitMQ Integration for Order Processing in Magento using Guzzle HTTP Client**

## **Document Revision History**

| **SNo** | **Date** | **Version** | **Author** | **Changes** |
| --- | --- | --- | --- | --- |
| 1 | 05/12/2024 | 1.0 | Prakash | Created the document for Rabbit MQ Integration |
|  |  |  |  |  |
|  |  |  |  |  |

# 

This document outlines the architecture and workflow for integrating **RabbitMQ** with the **Guzzle HTTP client** in a **Magento**-based order processing system. The solution enables asynchronous order processing, including fetching order data from an external API and inserting it into the Magento database using RabbitMQ consumers: **OrderFetchConsumer** and **OrderInsertConsumer**

### **Source Code:**

<https://drive.google.com/file/d/1iMuDAW8EKRipvw8ebAHwrwaH-ZvmeoMD/view?usp=sharing>

### 

### **1. System Overview**.

The system provides an efficient way of managing the entire order processing lifecycle using RabbitMQ for message queuing and Guzzle for HTTP communication. The architecture is designed for asynchronous processing to improve scalability and reliability.

* **Order Numbers Fetching**: Order numbers are fetched from an external API based on a timestamp.
* **Queueing**: Order numbers or details are published to RabbitMQ queues for further processing.
* **Consumers**:
  + **OrderFetchConsumer**: Listens to the **order\_fetch\_queue** and Fetches the complete order data from the external API and forwards it to the **order\_insert\_queue**.
  + **OrderInsertConsumer**: Listens to the **order\_insert\_queue** and inserts order data into the Magento database.

### **2. Architecture Flow**

#### **Architecture Reference link:**

<https://docs.google.com/document/d/1MjoXKgcDTxf47MPTFrbZ2vQo_XLeGsmVt2Io8zIbRl4/edit?tab=t.0>

#### **Order Data Workflow**

1. **Triggering the Process**: The process is initiated via a URL, which provides a timestamp. This URL triggers the fetching of order data.
   * **Example URL**:  
     <http://localhost/magento246/pub/klizer/srs/ordercontroller?timestamp=2024-12-04%2019:00:00>
   * The timestamp is used to filter order numbers for a specific date/time range.
2. **Fetching Order Numbers**: The **OrderController** listens for the timestamp parameter and call an external API, which returns order numbers within the specified time range. These order numbers are then published to the **order\_fetch\_queue** using **Guzzle** in RabbitMQ for further processing.
3. **Fetching Order Data via OrderFetchConsumer**:
   * **OrderFetchConsumer** listens to the **order\_fetch\_queue**.
   * It fetches order data from the external API based on the order number.
   * After fetching the complete order details, it publishes this data to the next queue (**order\_insert\_queue**) for insertion into the database.
4. **Inserting Order Data via OrderInsertConsumer**:
   * **OrderInsertConsumer** listens to the **order\_insert\_queue**.
   * It consumes the order data from the queue, processes it, and inserts the order details into the **srs\_erp\_orders** table in Magento.

### **3. Key Components and Technologies**

* **RabbitMQ**:
  + **Message Queuing**: RabbitMQ decouples the components of the system, allowing asynchronous communication. It ensures that orders are fetched, processed, and inserted independently of each other.
  + **Queues**:
    - **order\_fetch\_queue**: Holds order numbers for fetching order data.
    - **order\_insert\_queue**: Holds order data to be inserted into the Magento database.
* **Consumers**:
  + **OrderFetchConsumer**: Fetches order details based on the order number from the external API and pushes them into the **order\_insert\_queue**.
  + **OrderInsertConsumer**: Listens to the **order\_insert\_queue**, consumes the order details, and inserts them into the Magento database.

### **4. Detailed Workflow Process**

1. **Triggering the Process**: A user triggers the order fetching process by visiting a URL with a timestamp parameter. Example:  
   <http://localhost/magento246/pub/klizer/srs/ordercontroller?timestamp=2024-12-04%2019:00:00>

This timestamp is used to filter order numbers for a specific time range.

1. **Order Numbers Fetching**:  
   The **OrderController** listens for the timestamp parameter and call the external API. This call returns the order numbers within the specified time range. These order numbers are then published to the **order\_fetch\_queue** using **Guzzle** for further processing.
2. **Fetching Order Data via OrderFetchConsumer**:  
   The **OrderFetchConsumer** listens to the **order\_fetch\_queue**. It fetches the full order details from the external API and publishes the fetched data to the **order\_insert\_queue**.
3. **Inserting Order Data via OrderInsertConsumer**:  
   The **OrderInsertConsumer** listens to the **order\_insert\_queue**. It consumes the order data from the queue, processes it, and inserts the data into the **srs\_erp\_orders** table in Magento.

### **5. Guzzle Integration**

#### **Guzzle HTTP Client to push order numbers to the Queue**

**Guzzle** is used for making asynchronous calls to push order numbers into the **order\_fetch\_queue**. Key points:

* **Client Configuration**: Guzzle is configured with a base URL and necessary headers, including the API Token for authentication.
* **Request Execution**: Guzzle sends the order numbers to the order\_fetch\_queue.
* **Error Handling**: If the request fails, **Guzzle** throws exceptions (e.g., RequestException, TimeoutException), which are logged and retried.

### **6. Multiple Consumers**

#### **Enhancing Performance with Multiple Consumers**

The **Multiple Consumers** implementation allows for parallel processing of messages, improving the system's ability to handle large volumes of orders more quickly.

**Note**: We can limit the number of consumers processed in parallel by using the **maxConsumers** variable.

* **Parallel Processing**: By running multiple instances of **OrderFetchConsumer**, the system can fetch and push order data concurrently to the **order\_insert\_queue**, significantly reducing the time required for processing batches of orders.
* **Dynamic Scaling**: The number of consumers can be scaled dynamically based on the order volume. For example, you can increase the number of **OrderFetchConsumer** instances to handle more order numbers concurrently.

### **7. Batch Consumer**

#### **How Batch Consumers Work**

The **Batch Consumer** concept refers to the ability of consumers to process multiple messages in parallel. Specifically:

**Note**: We can limit the number of messages processed in parallel by using the **batchSize** variable.

* **OrderInsertConsumer** listens to the **order\_insert\_queue** and processes multiple order datas concurrently. reducing the time needed to process large numbers of orders.

By processing messages in parallel, the system becomes more efficient and scalable, capable of handling significantly more orders in the same amount of time.

### **8. Requeue Mechanism**

#### **Requeueing Failed Messages**

The **Requeue Mechanism** helps handle transient errors that may occur during message processing. If a message fails to be processed successfully by a consumer, it can be requeued for retry. This is useful for errors like network timeouts or server unavailability.

* **Requeueing** happens automatically when an exception is thrown or when a message fails to be processed.
* **Requeue Delay**: A configurable delay can be set between retries to avoid overloading the system.
* **Max Retry Attempts**: We can configure how many times a message can be retried using this **maxRetryLimit** variable before it is considered permanently failed and moved to the **Dead Letter Queue (DLQ)**.

### **9. Conclusion**

By integrating **RabbitMQ**, **Guzzle**, **Batch Consumers**, and **Multiple Consumers**, the order processing system in Magento becomes highly scalable, efficient, and resilient. The **Requeue Mechanism** and **Dead Letter Queue (DLQ)** ensure that errors are handled gracefully, minimizing the risk of system failures. This architecture guarantees that the order processing workflow remains smooth, even under heavy loads, ensuring a better overall user experience.

### **Performance Metrics**

| **Number of Records** | **Fetching order numbers** | **Fetching Order Details** | **Insert or Update Order Details** | **Total Process Time** |
| --- | --- | --- | --- | --- |
| 10 | 9 Seconds | 17 Seconds | 1 Second | 27 Seconds |
| 50 | 11 Seconds | 22 Seconds | 1.5 Seconds | 34.5 Seconds |
| 100 | 24 Seconds | 52 Seconds | 3 Seconds | 1 Minute and 19 Seconds |
| 250 | 26 Seconds | 86 Seconds | 7 Seconds | 1 Minute and 59 Seconds |
| 500 | 40 Seconds | 178 Seconds | 15 Seconds | 3 Minutes and 53 Seconds |