Message Queues | System Design

Last Updated : 05 Jan, 2024

A message queues is a form of service-to-service communication that facilitates asynchronous communication. It asynchronously receives messages from producers and sends them to consumers.

Important Topics for the Message Queues

- What is a Message Queue?
- Primary Purpose of Message Queue
- Key Components of a Message Queue System
- How Message Queue Work
- Need of Message Queue
- Use Cases of Message Queues

What is a Message Queues?

A Message Queue is a form of communication and data transfer mechanism used in computer science and system design. It functions as a temporary storage and routing system for messages exchanged between different components, applications, or systems within a larger software architecture.

Example:

Think about your favorite pizza place, where they make and deliver pizzas. Behind the scenes, there's a magical system that ensures everything runs smoothly. This magic is called a **Message Queue.** It's like a special to-do list that helps the chefs and delivery drivers know exactly what pizzas to make and where to deliver them, especially when things get super busy.

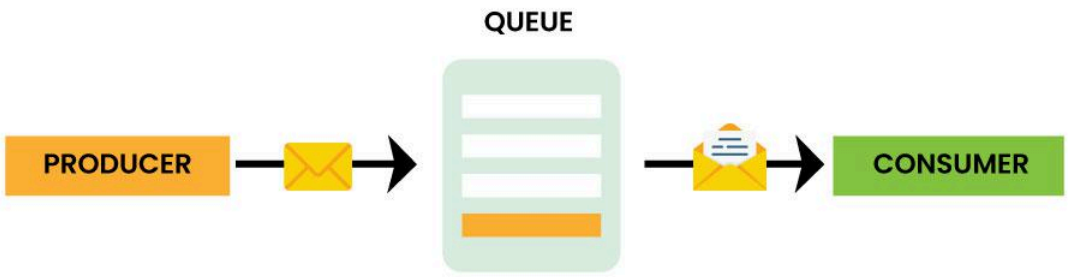
Primary Purpose of Message Queues

The primary purpose of a Message Queue are:

- It enable loosely coupled communication, ensuring that different parts of a system can exchange data without being directly connected or

system components independently, and maintain a buffer for messages in case the sender and receiver are not synchronized in real-time.

Key Components of a Message Queues System



Message Queue

- **Message Producer:** The message producer is responsible for creating and sending messages to the message queue. This can be any application or component withing a system that generates data to be shared.
- **Message Queue:** The message queue is a data structure or service that stores and manages the messages until they are consumed by the message consumers. It acts as a buffer or intermediary between producers and consumers.
- **Message Consumer:** The message consumer is responsible for retrieving and processing messages from the message queue. Multiple consumers can read messages concurrently from the queue.
- **Message Broker (Optional):** In some message queue systems, a message broker acts as an intermediary between producers and consumers, providing additional functionality like message routing,

- **Sending Messages:** The message producer creates a message and sends it to the message queue. The message typically contains data or instructions that need to be processed or communicated.
- **Queuing Messages:** The message queue stores the message temporarily, making available for one or more consumers. Messages are typically stored in a first-in, first out (FIFO) order.
- **Consuming Messages:** Message consumers retrieve messages from the queue when they are ready to process them. They can do this at their own pace, which enables asynchronous communication.
- **Acknowledgment (Optional):** In some message queue systems, consumers can send acknowledgments back to the queue, indicating that they have successfully processed a message. This is essential for ensuring message delivery and preventing message loss.

Need of Message Queues

Message Queue are needed to address a number of challenges in distributed systems, including:

- **Asynchronous Communication:** Message queue allow applications to send and receive messages without having to wait for a response. This is essential for building scalable and reliable systems.
- **Decoupling:** Message queues decouple applications from each other, allowing them to be developed independently. This makes systems more flexible and easier to maintain.
- **Scalability:** Message queues can be scaled to handle large volumes of messages by adding more servers. This makes them ideal for high-traffic applications.
- **Reliability:** Message queues can be designed to be highly reliable, with features such as message persistence, retries, and dead letter queues. This ensures that messages are not lost even in the event of failures.

processing. This can help improve the efficiency and accuracy of these processes.

Message Queues are used in a wide variety of applications, including:

- **Ecommerce:** Message Queues are used to process orders, payments, and shipping notifications.
- **Financial Services:** Message Queues are used to process transactions, fraud detection, and risk management systems.
- **Gaming:** Message queues are used to synchronize game servers and clients.
- **Social Media:** Message queues are used to distribute messages and notifications to users.
- **Internet of Things (IoT):** Message Queues are used to collect and process data from IoT devices.

Example for Message Queues

Problem Statement:

A simple example of a message queue is an email inbox. When you send an email, it is placed in the recipient's inbox. The recipient can then read the email at their convenience. This email inbox acts as a buffer between the sender and the recipient, decoupling them from each other.

Implementation of Message Queues

Message Queues can be implemented in a variety of ways, but they typically follow a simple pattern:

2. **Message Broker:** A server that stores and forwards messages between producers and consumers.
3. **Consumer:** An application that receives messages from a queue.

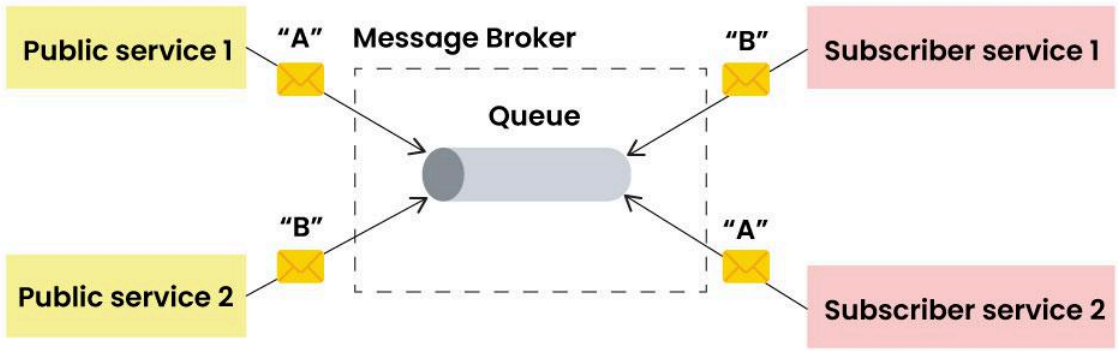
The message broker is responsible for routing messages to consumers and ensuring that they are delivered in the correct order. It also provides features such as message persistence, retries, and dead letter queues.

Types of Message Queues

There are two main types of message queues in system design:

1. **Point-to-point Message Queue**
2. **Publish-Subscribe Message Queue**

Point-to-Point Message Queues



Message Queue Point to Point

Point-to-point message queues are the simplest type of message queue. When a producer sends a message to a point-to-point queue, the message is stored in the queue until a consumer retrieves it. Once the message is retrieved by a consumer, it is removed from the queue and cannot be processed by any other consumer.

- **Request-Response:** A producer sends a request message to a queue, and a consumer retrieves the message and sends back a response messages.
- **Work Queue:** Producers send work items to a queue, and consumers retrieve the work items and process them.
- **Guaranteed Delivery:** Producers send messages to a queue, and consumers can be configured retry retrieving messages until they are successfully processed.

Publish-Subscribe Message Queues

Publish-Subscribe Message Queues are more complex than point-to-point message queues. When a producer publishes a message to publish/subscribe queue, the message is routed to all consumers that are subscribed to the queue. Consumers can subscribe to multiple queues, and they can also unsubscribe from queues at any time.

Publish-Subscribe Message Queues are often used to implement real-time streaming applications, such as social media and stock market tickers. They can also be used to implement event-driven architecture, where components of a system communicate with each other by publishing and subscribing to events.

Message Serialization

Message Serialization is the process of converting complex data structures or objects into a format that can be easily transmitted, stored, or reconstructed. Message Serialization formats include:

- **JSON (JavaScript Object Notation):** A lightweight data interchange format used for structured data, commonly supported by many programming languages.
- **XML (eXtensible Markup Language):** A format that uses tags to

- **Protocol Buffers (protobuf):** A binary serialization format developed by Google that is highly efficient and language-agnostic.
- **Binary Serialization:** Custom binary formats are used for performance-critical applications due to their compactness and speed.

Message Structure

A typical message structure consists of two main parts:

- **Headers:** These contain metadata about the message, such as unique identifier, timestamp, message type, and routing information.
- **Body:** The body contains the actual message payload or content. It can be in any format, including text, binary data, or structured data like JSON.

Message Routing

Message Routing involves determining how messages are directed to their intended recipients. The following methods can be employed:

- **Topic-Based Routing:** Messages are sent to topics or channels, and subscribers express interest in specific topics. Messages are delivered to all subscribers of a particular topic.
- **Direct Routing:** Messages are sent directly to specific queues or consumers based on their addresses or routing keys.
- **Content-Based Routing:** The routing decision is based on the content of the message. Filters or rules are defined to route messages that meet specific criteria.

Scalability of Message Queues

Scalability is essential to ensure that a message queue system can handle increased loads efficiently. To achieve scalability:

- **Distributed Queues:** Implement the message queue as a distributed

- **Partitioning:** Split queues into partitions to distribute message processing across different nodes or clusters.
- **Load Balancing:** Use load balancers to evenly distribute incoming messages to queue consumers.

Dead Letter Queues

Dead Letter Queues (DLQs) are a mechanism for handling messages that cannot be processed successfully. This includes:

- Messages with errors in their content or format.
- Messages that exceed their time-to-live (TTL) or delivery attempts.
- Messages that cannot be delivered to any consumer.

DLQs provide way to investigate and potentially reprocess failed messages while preventing them from blocking the system.

Securing Message Queues

Securing Message Queues is crucial to protect sensitive data and ensure the integrity of the messaging system:

- **Access Control:** Enforce access controls to restrict who can send, receive, or administer the message queue.
- **Encryption:** Implement data encryption in transit and at rest to protect messages from eavesdropping.
- **Authentication:** Ensure that only authorized users or systems can connect to the message queue.
- **Authorization:** Definer granular permissions to control what actions users or systems can perform withing the messaging system.

Message Prioritization

Message Prioritization is the process of assigning priority levels to messages to control their processing order. Prioritization criteria can

- **Urgency:** Messages with higher priority may need to processed before lower-priority messages.
- **Message Content:** Messages containing critical information or commands may receive higher priority.
- **Business Rules:** Custom business rules or algorithms may be used to determine message priority.

Load Balancing of Messages

Load Balancing ensures even distribution of message processing workloads across consumers. Strategies for load balancing include:

- **Round-Robin:** Messages are distributed to consumers in a cyclic manner.
- **Weighted Load Balancing:** Assign different weights to consumers to control the distribution of messages.
- **Dynamic Load Balancing:** Analyze the load on consumers in real-time and direct messages to less loaded consumers.

These aspects are essential for designing, implementing, and managing message queues, which are fundamental in building scalable, reliable, and efficient distributed systems and microservice architectures. The specific approach may vary based on the message system or technology used and the requirements of the application.

Message Queue Implementation in C++

Problem Statement:

> *In a real-world scenario, you might want to consider using a dedicated message queue service like RabbitMQ or Apace Kafka for distributed systems.*

Step 1: Define the Message Structure:

Start by defining a structure for your messages. This structure should contain the necessary information for communication between different parts of your system.

C++

```
// Message structure
struct Message {
    int messageType;
    std::string payload;
    // Add any other fields as needed
};
```

Step 2: Implement the Message Queue:

Create a class for your message queue. This class should handle the operations like enqueue and dequeue.

C++

```
#include <queue>
#include <mutex>
#include <condition_variable>

class MessageQueue {
public:
    // Enqueue a message
    void enqueue(const Message& message) {
        std::unique_lock<std::mutex> lock(mutex_);
        queue_.push(message);
        lock.unlock();
        condition_.notify_one();
    }
```

```
        std::unique_lock<std::mutex> lock(mutex_);
        // Wait until a message is available
        condition_.wait(lock, [this] { return !queue_.empty(); });

        Message message = queue_.front();
        queue_.pop();
        return message;
    }

private:
    std::queue<Message> queue_;
    std::mutex mutex_;
    std::condition_variable condition_;
};
```

Step 3: Create Producers and Consumers

Implement functions or classes that act as producers and consumers. Producers enqueue messages, and consumers dequeue messages.

C++

```
// Producer function
void producer(MessageQueue& messageQueue, int messageType, const std::strin
    Message message;
    message.messageType = messageType;
    message.payload = payload;

    messageQueue.enqueue(message);
}

// Consumer function
void consumer(MessageQueue& messageQueue) {
    while (true) {
        Message message = messageQueue.dequeue();
        // Process the message
        // ...
```

Step 4: Use the Message Queue

Create instances of the message queue, producers, and consumers, and use them in your program.

C++

```
int main() {
    MessageQueue messageQueue;

    // Create producer and consumer threads
    std::thread producerThread(producer, std::ref(messageQueue), 1, "Hello,
    std::thread consumerThread(consumer, std::ref(messageQueue));

    // Wait for threads to finish
    producerThread.join();
    consumerThread.join();

    return 0;
}
```

This is basic example, and in a real-world scenario, you want to handle edge cases, error conditions, and potentially use more advanced features provided by external message queue libraries. Additionally for a distributed systems, you might need to consider issues like message persistence, fault tolerance, and scalability.

Conclusion

In conclusion, message queues are invaluable tools in modern system design. They enable scalable, reliable, and resilient systems by facilitating asynchronous communication, load balancing, and decoupling

maintainability of your software systems. Whether you are building a micro-service based architecture, processing large volumes of data, or managing real-time events, message queues are a vital component of your toolkit.

Want to be a Software Architect or grow as a working professional? Knowing both Low and High-Level System Design is highly necessary. As such, our course fits you perfectly: **Mastering System Design: From Low-Level to High-Level Solutions**. Get in-depth into **System Design** with hands-on projects and **Real-World Examples**. Learn how to come up with systems that are scalable, efficient, and robust—ones that impress. Ready to elevate your tech skills? Enrol now and build the future!

thesu...

21

Previous Article

Caching - System Design Concept

Next Article

Communication Protocols In System Design

Similar Reads

Do Message Queues use Web Sockets?

Yes, some Message Queue systems can use WebSockets as a communication protocol. WebSockets provide a full-duplex communicatio...

2 min read

What is the Maximum Message Size in Message Queue?

The maximum message size in a message queue refers to the largest

Message Broker vs. Message Queue

Understanding the distinctions between Message Brokers and Message Queues is crucial for designing efficient communication architectures. This...

7 min read

What are Message Brokers in System Design?

A message broker is a key architectural component responsible for facilitating communication and data exchange between different parts of ...

12 min read

Design a system that counts the number of clicks on YouTube videos |...

Designing a Click Tracking System for YouTube Videos involves architecting a comprehensive and efficient solution to monitor and analyze user...

15+ min read

Design Restaurant Management System | System Design

In the modern restaurant industry, delivering exceptional dining experiences requires more than just good cuisine. Restaurant Managemen...

15 min read

Design a Picture-Sharing System - System Design

In the present day, the need for good tools to exchange and organize images has never been this much higher. As for social networking, e-...

11 min read

Difference between System Design and System Architecture

When it comes to software development and engineering there can be some confusion, between the two terms "System Design" and "System...

3 min read

8 min read

System Analysis | System Design

In the areas of science, information technology, and knowledge, the difficulty of systems is of much importance. As systems became more...

6 min read

Article Tags : Geeks Premier League System Design Geeks Premier League 2023

GeeksforGeeks
Sanchhaya Education Private Limited
Corporate & Communications Address:- A-143, 9th Floor, Sovereign Corporate Tower, Sector- 136, Noida, Uttar Pradesh (201305) | Registered Address:- K 061, Tower K, Gulshan Vivante Apartment, Sector 137, Noida, Gautam Buddh Nagar, Uttar Pradesh, 201305

English Grammar
Commerce
World GK

Data Science
CS Subjects