**What are Message Queues and why they are widely used in distributed systems?**

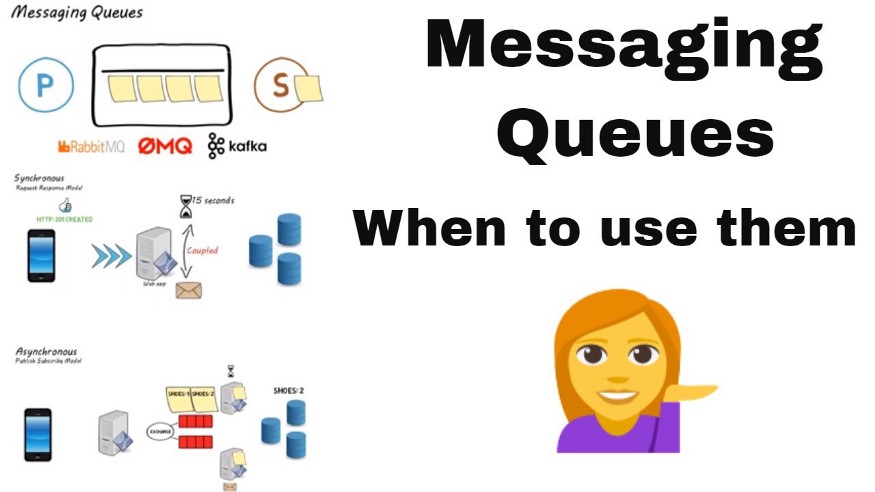


Image Source: <https://i.ytimg.com/vi/sfQwMu0SCT8/maxresdefault.jpg>

Hello everyone. In this article, we are going to see about Message Queues — a common and widely used component in distributed systems. We will start to understand the Message Queue and its internals and then see about the message patterns, use-cases, and the advantage of the Message Queues in detail. To end with, we will also see the list of popular message queues in use now.

**What is a Queue?**



Image Source: [https://miro.medium.com/max/720/1\*ieA8kwcF25JENDEO3rRrAw.jpeg](https://miro.medium.com/max/720/1*ieA8kwcF25JENDEO3rRrAw.jpeg)

A Queue is a data structure in programming that follows the real-world queue. It follows the FIFO pattern (First In First Out). The person who enters the queue exists first. Likewise in a Queue data structure, the element that is added first will be one removed first. The elements enter the queue at the rear i.e the Tail and exit towards the head of the queue.

**What is a Message Queue?**

A message queue is used for asynchronous service-to-service communication and they are widely used in serverless and microservices architectures.

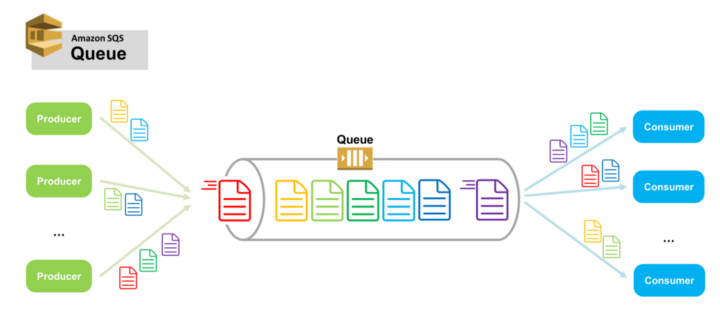


Image Source: [https://miro.medium.com/max/720/1\*DRW4lVeUoIc2qS6Wel4Caw.png](https://miro.medium.com/max/720/1*DRW4lVeUoIc2qS6Wel4Caw.png)

In modern cloud-native architecture, applications are decoupled into smaller, independent building blocks so that they can be developed, deployed, and maintained easily. Message queues provide perfect communication and coordination between the distributed components/applications.

The Messages are stored in the queue until they are processed by the consumer. Each message is processed only once, by a single consumer and then the message is deleted after that to avoid duplicate processing.

A message queue provides a lightweight buffer that temporarily stores messages. It provides endpoints that allow software systems (producers and consumers) to connect to the queue in order to send and receive messages. The messages are usually small and requests, replies, error messages, or just plain information. A message is nothing but a serialized data construct mostly in XML or JSON format, which contains all the required information to process the request.

Message queues can be used to decouple heavy workloads and process them asynchronously and by doing so, they can significantly quicken the development of decoupled applications, while improving performance, reliability, and scalability.

**The Architecture of Rabbit MQ**

Let us look at the architecture of Rabbit MQ, which is one of the most widely used Messaging Queues in modern architecture.

In RabbitMQ, a producer does not send a message directly to a queue. Rather, it uses an exchange as a routing mediator. The exchange decides if the message goes to one queue, to multiple queues, or is simply discarded.

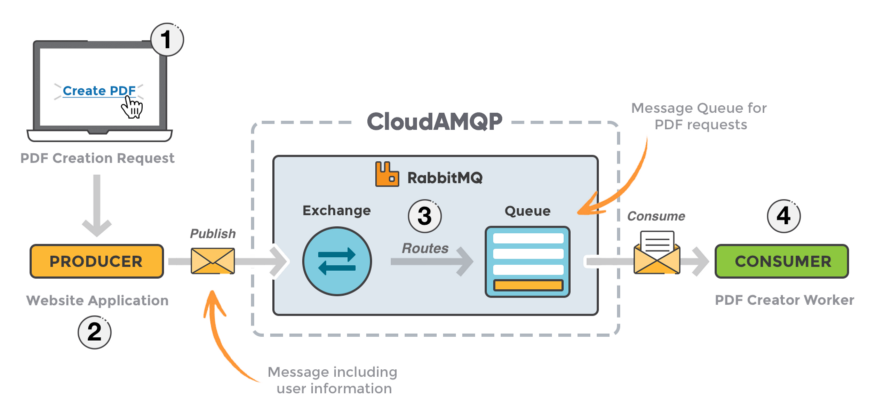


Image Source: <https://www.cloudamqp.com/img/blog/rabbitmq-beginners-updated.png>

The Rabbit MQ consists of 4 components as shown in the diagram above

* **Producer:** The component that adds the message to the queue is known as a message producer.
* **Consumer:** The component that consumes the messages from the queue to do some processing is known as the message consumer.
* **Queue:**This is a data type used by the message broker to store messages using FIFO logic.
* **Exchanger:** Enables route messages and sends them to queues. The messages are filtered and routed depending on the type of exchange.

**What is a Message Broker?**

A message queue is a distributed system consisting of multiple servers called **brokers**. These brokers together form a cluster, thus making the queueing system highly available, scalable, and reliable.

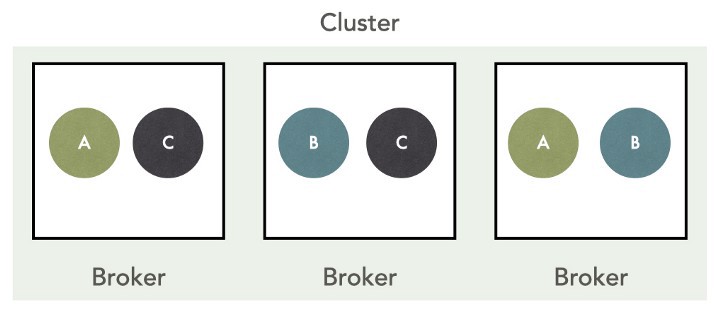


Image Source: [https://miro.medium.com/max/720/1\*viuCjcDxZQzB1rVedq7uCA.jpeg](https://miro.medium.com/max/720/1*viuCjcDxZQzB1rVedq7uCA.jpeg)

Once a message is received, the message queue redundantly stores the message across multiple brokers for availability and durability. As seen in the diagram above, each of the messages A, B and C are stored across 2 different brokers.

This means there can be multiple copies of the same message, stored across multiple brokers. In the event of a failure of a broker, the messages can be recovered from the remaining brokers in the cluster.

In a nutshell, the message queue system follows the exact same principles as distributed systems.

**Messaging Patterns**

Let us look at the different types of messaging patterns used in the Message queues.

**1. One-way messaging (Point to Point)**



Image Source: [https://miro.medium.com/max/720/1\*23io3fbKOXzq9L1aihlaew.jpeg](https://miro.medium.com/max/720/1*23io3fbKOXzq9L1aihlaew.jpeg)

In this messaging pattern, the Producer pushes the message to the queue and the Consumer consumes and processes it.

**Example**: A good use case of One-way messaging is to send an email to the user upon purchasing an item online. The producer can put the email content, subject and the sender email inside a message and the consumer uses the content to send emails to the users.

**2. Request Response Pattern**

In this pattern, the Producer sends a message to one queue which is consumed by the consumer. Later upon processing the request, the consumer will send a response in another queue which is consumed by the Producer to complete the process.

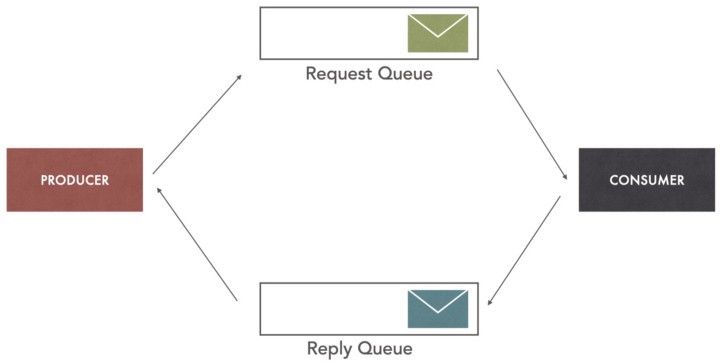


Image Source: [https://miro.medium.com/max/720/1\*WxO5RwcxQX\_P2iGSIiHtHw.jpeg](https://miro.medium.com/max/720/1*WxO5RwcxQX_P2iGSIiHtHw.jpeg)

**Example**: A good use case for this is distributed transaction management using the Saga pattern. The producer could be an Order service and the Consumer could be the Payment service. Upon Order submission by the user, Payment Service is notified. Once the Payment is successful or failure, the Payment service sends a message to notify the Order service. The Order service marks the order status as a success or failure based on the outcome of the payment service.

Please take a look at my article below for more info about Saga Pattern.

**[Distributed Transaction Management in Microservices — Part 2 — Saga Pattern](https://medium.com/javarevisited/distributed-transaction-management-in-microservices-part-2-saga-pattern-53808a55e641" \t "_blank)**

[Hello everyone. This article is a continuation of the previous article](https://medium.com/javarevisited/distributed-transaction-management-in-microservices-part-2-saga-pattern-53808a55e641" \t "_blank)

[medium.com](https://medium.com/javarevisited/distributed-transaction-management-in-microservices-part-2-saga-pattern-53808a55e641" \t "_blank)

**3. Publish-Subscribe / Fan out the pattern**

This is one of the widely used patterns in distributed systems.

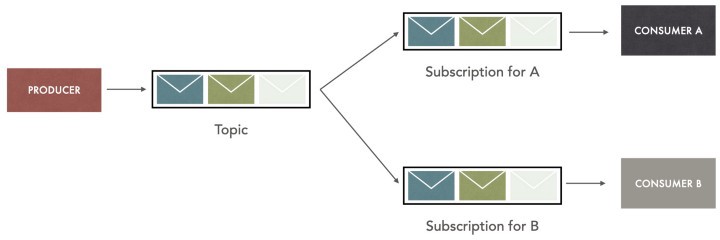


Image Source: [https://miro.medium.com/max/720/1\*0vtmfkPK3RFrw8a-tzq5-g.jpeg](https://miro.medium.com/max/720/1*0vtmfkPK3RFrw8a-tzq5-g.jpeg)

The producer sends a message to a Topic and this topic is listened to by multiple queues. This pattern is called as Pub-Sub or Fan-out. Broadcast messaging is a very good use case for this pattern.

**Example**: Another example of this would be Online ordering. Once an order is received, the message is sent to the topic by Order Service. There could be many consumers like Payment Service, Invoice Service, and Notification service which will get triggered by the message sent by Order Service.

Basically, this pattern can be used to trigger a lot of asynchronous events.

**Synchronous (API) vs Asynchronous(Message Queue)**

Let us compare the Synchronous and Asynchronous ways of communication between the microservices. We use **API calls for sync** and **Message Queue / Topic**for Async calls.

REST APIs are commonly used for inter-microservice communication and it uses HTTP which is a request/response protocol. It can be used in situations that call for a synchronous request/reply. This means that services making requests via REST APIs must be designed to expect an immediate response. If the client receiving the response is down, the sending service will be blocked while it awaits the reply. This will have a cascading effect on all the following calls from the Client to the Service.

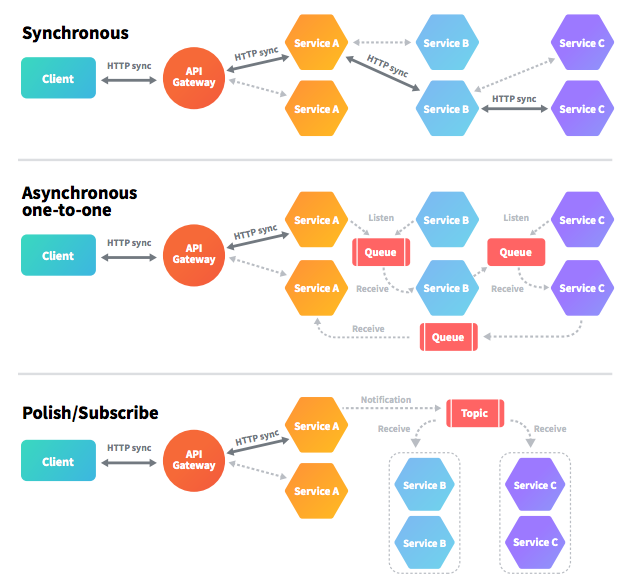


Image Source: <https://dz2cdn1.dzone.com/storage/temp/7542942-screen-shot-2017-12-14-at-42543-pm.png>

In the diagram above under **Synchronous**, Service A calls Service B which in turn calls Service C. Now Service A is blocked for a response from Service C and Service B

The Message brokers enable asynchronous communications between services so that the sending service need not wait for the receiving service’s reply.

In the above diagram under the **Asynchronous one-to-one**, Service A communicates with Service B using Message Queue and Service B follows the same with Service C. Now the system is completely decoupled and there are no blocking requests hence this is efficient.

The section under **Publish / Subscribe** is also decoupled but it uses the Publish-Subscribe mechanism to send messages to multiple consumers.

**Best Use Cases of Message Queues**

Message Queues can be used to address a wide range of business use cases in different system architectures

The common use cases are

* **E-commerce order processing and fulfillment:** With their ability to improve fault tolerance and ensure that messages are consumed only once, Message brokers are an obvious choice for online order processing.
* **Financial transactions and payment processing**: It is mission critical to ensure that payments are only sent once. Using a message broker to handle the data from these transactions ensures that payment information is not lost or accidentally duplicated.
* **Protecting highly sensitive data at rest and in transit:** In a highly regulated environment with significant security risks, using message queues that support end-to-end encryption would be a great choice.
* **Long-running processes and background jobs:** When requests take a significant amount of time, it is the perfect scenario to incorporate a message queue. Some real-life examples could include Images Scaling, Sending large/many emails, File scanning, PDF processing, Calculations, etc

**Applications of Message Queues**

1. Asynchronous Communication
2. Reliability
3. Scalability
4. Decoupling the systems
5. Buffering and load management
6. Delivery Guarantee

**Commonly used Message Queues**

Most of the architecture nowadays uses Message Queues and hence they are a lot of Queues in action now. The most popular ones are as follows

1. Rabbit MQ
2. Apache Kafka
3. Apache ActiveMQ
4. Google pubsub
5. AWS SNS / SQS
6. Azure Queue

**Summary**

In this article, we saw what is a Queue and understood about Message Queues are. Then we saw about the architecture of Message Queue, Message broker, and the different types of Messaging patterns. After that, we saw the difference between Synchronous and Asynchronous communication in microservices. Towards the end of the article, we saw about the use-cases for the Message Queue and its applications, and then we ended the article by listing some of the commonly used Queues.