What is message queueing?

Lovisa Johansson

Message queuing allows applications to communicate by sending messages to each other. The message queue provides a temporary message storage when the destination program is busy or not connected.

A **queue** is a line of things waiting to be handled - in sequential order starting at the beginning of the line. A message queue is a queue of messages sent between applications. It includes a sequence of work objects that are waiting to be processed.

A **message** is the data transported between the sender and the receiver application; it's essentially a byte array with some headers on top. An example of a message could be something that tells one system to start processing a task, it could contains information about a finished task or just be a plain message.

The basic architecture of a **message queue** is simple, there are client applications called producers that create messages and deliver them to the message queue. An other application, called consumer, connect to the queue and get the messages to be processed. Messages placed onto the queue are stored until the consumer retrieves them.

Message queues

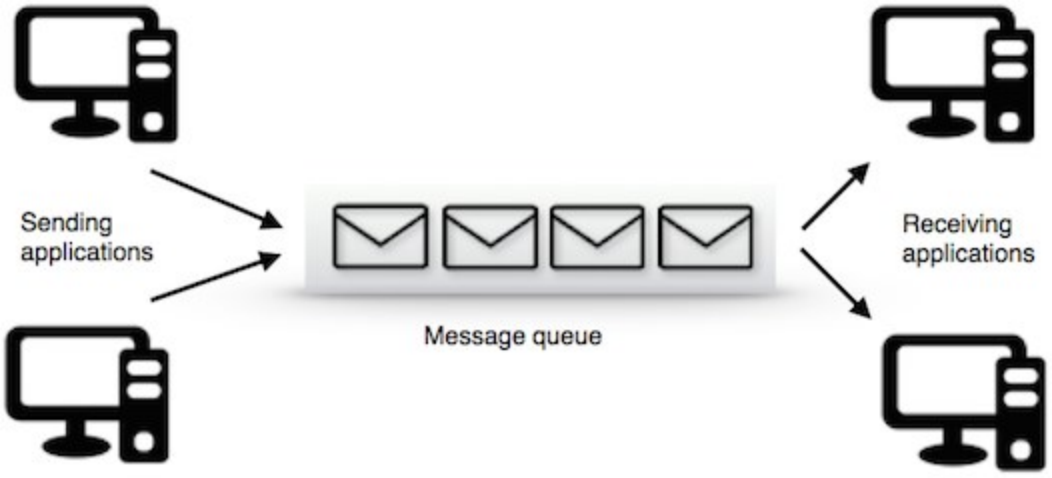
A message queue provides an **asynchronous communications protocol,** a system that puts a message onto a message queue does not require an immediate response to continuing processing. Email is probably the best example of asynchronous messaging. When an email is sent can the sender continue processing other things without an immediate response from the receiver. This way of handling messages **decouple** the producer from the consumer. The producer and the consumer of the message do not need to interact with the message queue at the same time.

Decoupling and Scalability

Decoupling is used to describe how much one piece of a system relies on another piece of the system. Decoupling is the process of separating them so that their functionality will be more self-contained.

A decoupled system is achieved when two or more systems are able to communicate without being connected. The systems can remain completely autonomous and unaware of each other. Decoupling is often a sign of a computer system that is well structured. It is usually easier to maintain, extend and debug.

If one process in a decoupled system fails to process messages from the queue, other messages can still be added to the queue and be processed when the system has recovered. You can also use a message queue to delay processing; A producer post messages to a queue. At the appointed time, the receivers are started up and process the messages in the queue. A queued message can be stored-and-forwarded, and the message be redelivered until the message is processed.



Instead of building one large application, is it beneficial to decouple different parts of your application and only communicate between them asynchronously with messages. That way different parts of your application can evolve independently, be written in different languages and/or maintained by separated developer teams.

A message queue will keep the processes in your application separated and independent of each other. The first process will never need to invoke another process, or post notifications to another process, or follow the process flow of the other processes. It can just put the message on the queue and then continue processing. The other processes can also handle their work independently. They can take the messages from the queue when they are able to process them. This way of handling messages creates a system that is easy to maintain and easy to scale.

Message queuing - a simple use case

Imagine that you have a web service that receives many requests every second, where no request is afford to get lost and all requests needs to be processed by a process that is time consuming.

Imagine that your web service always has to be highly available and ready to receive new request instead of being locked by the processing of previous received requests. In this case it is ideal to put a queue between the web service and the processing service. The web service can put the "start processing"-message on a queue and the other process can take and handle messages in order. The two processes will be decoupled from each other and does not need to wait for each other. If you have a lot of requests coming in a short amount of time, the processing system will be able to process them all anyway. The queue will persist requests if their number becomes really huge.

You can then imagine that your business and your workload is growing and you need to scale up your system. All that is needed to be done now is to add more workers, receivers, to work off the queues faster.