**What is Messaging?**

# Content

[Content 1](#_Toc98693896)

[1. Research Questions 2](#_Toc98693897)

[2. What is messaging? 2](#_Toc98693898)

[2.1. Distributed architecture & Scalability 2](#_Toc98693899)

[2.2. Design patterns 2](#_Toc98693900)

[3. Available tools & comparison 4](#_Toc98693901)

[4. Conclusion 5](#_Toc98693902)

[Sources 5](#_Toc98693903)

# Research Questions

To answer the question “What is messaging?” we define a couple of sub-question:

1. What is messaging?
2. How does messaging fit in a distributed architecture?
3. How does messaging help with scalability
4. What design patterns can be used for messaging?
5. What tools are available for messaging?
6. Which tool is the optimal for this project?

# What is messaging?

In simple terms, messaging is an asynchronous form of communication. Asynchronous means that when one application sends a message to another application, it won’t wait for the response. This has a couple of advantages. If the second application is offline, it won’t block the first application and the message is stored for later use. Another advantage is that the second application can process the message at any point in time, independently from the first application.

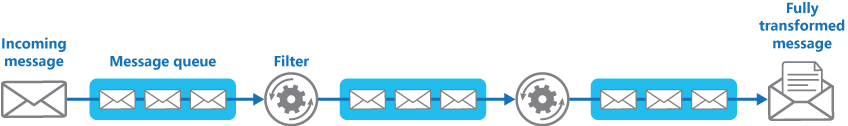
## Distributed architecture & Scalability

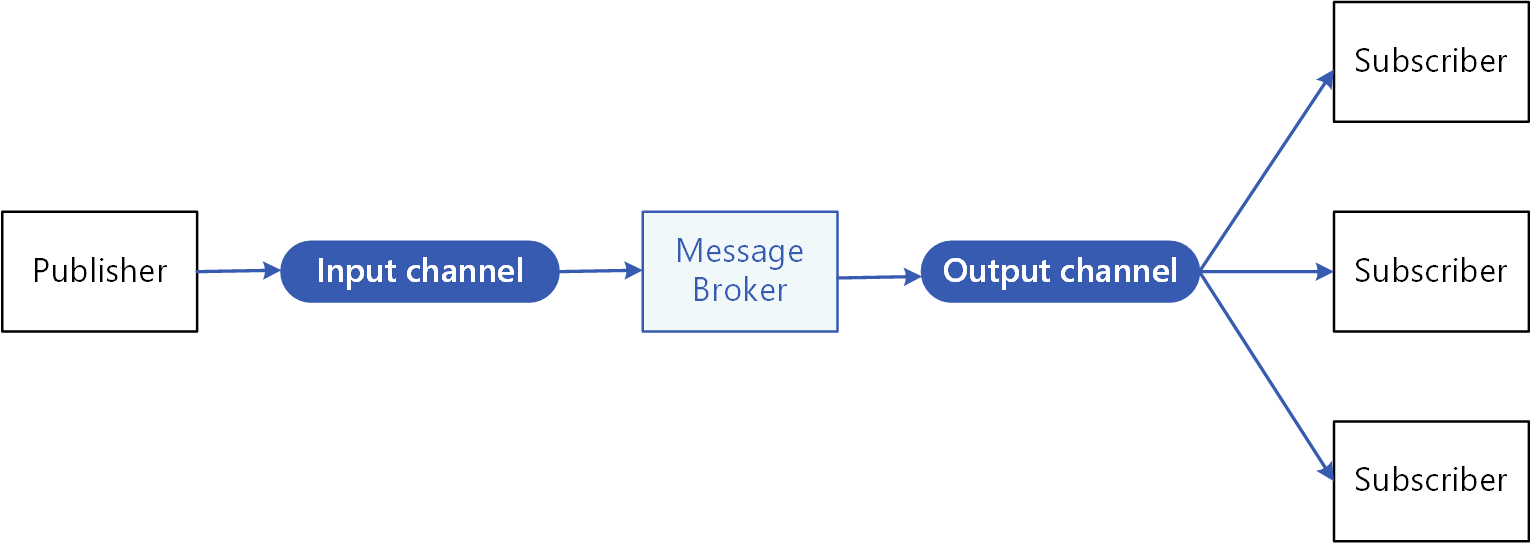
Messaging fits well in a distributed & horizontal scalable architecture. Since it is asynchronous, you can run multiple application producing messages, and multiple application consuming messages to support horizontal scaling. Since messages are stored until consumed, you prevent producing applications from waiting and can handle bursts of messages without blocking producers.

Since messaging is universal it can extend over multiple separate applications. It is often used in a microservice architecture. Because of this it allows you to build a loosely coupled distributed architecture.

## Design patterns

There exist a couple of messaging design patters. Each has their own use cases and pros and cons.

* **Pipes and Filters**  
  This is a pattern that allows you to send messages from one service to another. The pipers represent these connections, and the filters is the part that decides which message is send to the next service. Using this pattern, you could change the message in between like adding a timestamp or some more complex task.  
    
  
* Afbeelding met tekst

  Automatisch gegenereerde beschrijving**Point to Point**This is typically a queue. You have an application which puts a message on the queue, and another application (referred to as a consumer) who consumes the message, after which the message is no longer on the queue. You have different types of queues: first in first out (FIFO), or a priority queue for example.
* **Publisher-Subscriber**This pattern is somewhat the same as a point-to-point FIFO queue, however it allows you to define multiple subscribers who each need to receive the message produces by a publisher. One way to do this is by grouping subscribers and delivering each message once to each group. This pattern requires a mechanism for copying each message from the input channel to the output channels for each subscriber group. This is usually done by a message broker or even bus.  
    
  

# Available tools & comparison

There exists a lot of different tools that each tackle the eventing and messaging problems from a different perspective, with each having a different set of features and capabilities.

1. **Apache Kafka**  
   is an open-source distributed event streaming for high-performance data pipelines, streaming analytics, data integration, and mission-critical applications. An Azure hosted equivalent is Azure Event Hubs. It provides the same interface for libraries as Apache Kafka, such that it is interchangeable and does not require code changes.
2. **RabbitMQ**  
   is lightweight and easy to deploy on premises and in the cloud message broker. It supports multiple messaging protocols. RabbitMQ can be deployed in distributed and federated configurations to meet high-scale, high-availability requirements. An Azure hosted equivalent is Azure Service Bus.
3. **Apache ActiveMQ**  
   is the most popular open source, multi-protocol, Java-based message broker. It comes in two different versions, the “Classic” and “Artemis” which is an extension of the first one.
4. **MSMQ**  
   is an Azure hosted message queue. The AWS equivalent is Amazon SQS.
5. **Redis**  
   Redis is an open source, in-memory data structure store, used as a database, cache, and message broker.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Pattern | Messaging | Storing | Processing | At least once delivery\* | Topics / multiple queues | Multi- Consumer support\*\* | Pull / Push | Horizontal Scalability | Ordered |
| Kafka | Publish/Subscribe | ✓ | ✓ | ✓ | ✓ ✓ ✓ | ✓ | ✓ *groups* | ✓ ✓ | ✓ | ✓ |
| RabbitMQ | Publish/Subscribe | ✓ | ✓ | 🗶 | ✓ ✓ 🗶 | ✓ | 🗶 | ✓ ✓ | 🗶 | 🗶 |
| ActiveMQ |  | ✓ | ✓ | 🗶 | ✓ ✓ 🗶 | ✓ | 🗶 | ✓ ✓ | ✓ | ✓ |
| MSMQ |  | ✓ | ✓ | 🗶 | ✓ 🗶 🗶 | 🗶 | 🗶 | ✓ 🗶 | ✓ | 🗶 |
| Redis | Point to Point | ✓ | ✓ | 🗶 | ✓ 🗶 🗶 | 🗶 | 🗶 | ✓ 🗶 | ✓ | 🗶 |

\* Three checks mean it supports at least once delivery, at most once delivery and exactly once delivery.

\*\* You can always still push to two different queues and handle multi consumers that way.

# Conclusion

Messaging is an asynchronous form of communication. It fits well in a distributed & horizontal scalable architecture. Apache Kafka is the most feature rich and battle tested message broker available and is there for a great choice for any type of enterprise application.

Another advantage of Apache Kafka is that is has an Azure hosted equivalent. Such that developing with Kafka allows us to host our own Kafka cluster or use the hosted Azure variant without changing our code. That makes Kafka the better choice for this project.

# Sources

* <https://pratikkale0789.medium.com/kafka-or-rabbitmq-cb31e42d8f2>
* <https://kafka.apache.org/documentation/>
* <https://www.rabbitmq.com/reliability.html>
* <https://activemq.apache.org/components/artemis/documentation/>
* <https://redis.io/documentation>
* <https://docs.microsoft.com/en-us/previous-versions/windows/desktop/legacy/ms711472(v=vs.85)>
* <https://docs.microsoft.com/en-us/azure/architecture/patterns/publisher-subscriber>
* <https://docs.microsoft.com/en-us/azure/event-grid/handler-service-bus>