**Case study:  
Messaging**

**PREPARED FOR**

Niko Kuijpers

**PREPARED BY**

Maarten Blömer

Nick Krijgsman

Faruk Aydin

Jursley Gonzalez

Vincent Andersen

# **Table of Contents**

[**Table of Contents**](#_pykfpgqrlej) **2**

[**Case Study**](#_3s39dys3cx66) **4**

[Case goal:](#_qwqn8mh5s18) 4

[Case questions:](#_4frtw1206ej) 4

[Questions to be mindful of:](#_rhg0mb87rhxw) 4

[Case purposes:](#_2bhbl6st83h1) 4

[**Brainstorm**](#_whir32t8ue2v) **5**

[**Technology Research**](#_wisaxg2eeii0) **6**

[**RabbitMQ (Vincent)**](#_m0r8exn1o4kt) **7**

[Functionality](#_z7mn4nuuuqu8) 7

[Delivery Acknowledgement](#_6ixo8fk0vj6r) 8

[Protocols](#_ld3py63clj0m) 8

[Routing of messages](#_rgnrqvbzgdaz) 8

[Message priority](#_fxvy4j72wz9e) 8

[Scaling](#_o6y90t2ykvm0) 8

[Monitoring](#_7remm5olx5p4) 8

[Learning curve](#_mqhrgdom5hpq) 9

[Non Functional Requirements](#_kv2ig5u12s1g) 9

[Apache Kafka (Maarten)](#_uzz7tegvq59b) 10

[Functionality](#_bf5hgpxp5z40) 10

[Learning curve](#_9hnhzpn9x2pf) 10

[Type of messaging](#_wf2ifyxt8jyq) 10

[Max messaging rate](#_pvq0qpr17ut5) 10

[Non functionals](#_pvq0qpr17ut5) 10

[Azure Service Bus (Jursley)](#_xgpkt45grrm7) 11

[Functionality](#_qfvs9g16nkhx) 11

[Learning curve](#_dfxsfbsxbk08) 11

[Type of messaging](#_pvq0qpr17ut5) 11

[Max messaging rate](#_bgefv3s3e1kh) 12

[Amazon MQ (Faruk)](#_dj8cqg3ma4zw) 13

[Functionality](#_c1mi0ciykfc2) 13

[ZeroMQ (Nick)](#_4tw0nfup6dsu) 14

[Functionality](#_978121k3pn1x) 14

[**Technology Selection**](#_pseqslrx1x1j) **15**

# Case Study

## **Case goal:**

Research and development agency “High Sky” needs to stitch together 15 different systems into 1 system. The focus of this challenge is to analyse the context of the project and design a system for that. Part of the design is technology selection: select supporting solutions that are almost perfect for the job and accompany them with alternatives and motivation for our choices.

## Case questions**:**

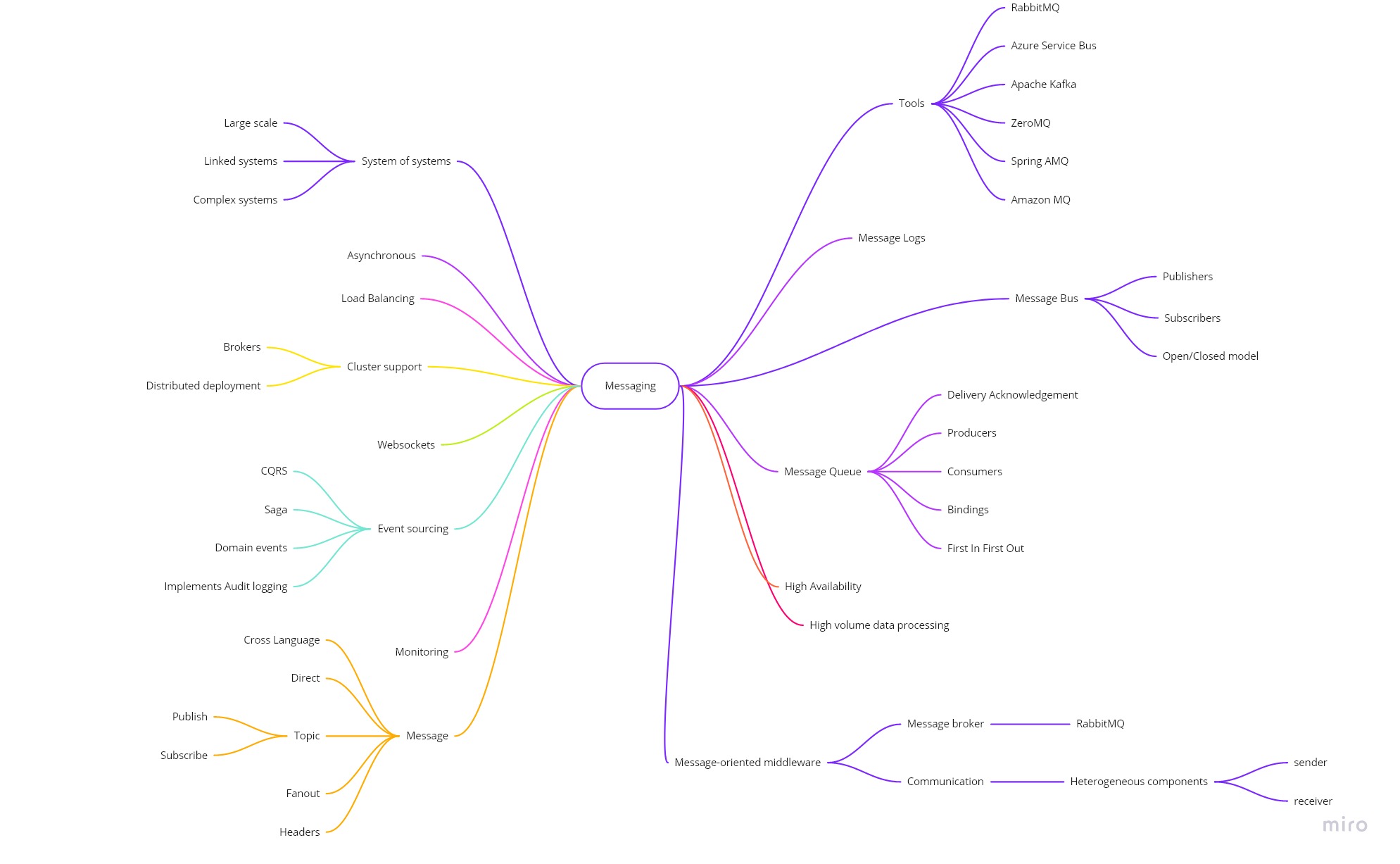
### Questions to be mindful of**:**

* Which tools serve the requested functionalities?
* Which tools introduce too much of a learning curve in order to implement the tech?
* What are the maximum rates of messages that the tools provide?
* How do the tools implement subscription based information sharing?
* What types of messaging does exist?
* Which non-functional requirements does the tool solve?

## Case purposes:

* Try to come up with a list of capabilities that your group project needs to support (you may need to consult your PO and update your project backlog after this)
* Use the system of systems design as an inspiration for your group project: can you reuse elements of it for the design of your group project?
* Decide what supporting technology can also be used for your group project, and motivate why.

# **Brainstorm**



# Technology Research

## 

|  |  |
| --- | --- |
| **Technology** | **Researcher** |
| RabbitMQ | Vincent |
| Apache Kafka | Maarten |
| Azure Service Bus | Jursley |
| Amazon MQ | Faruk |
| ZeroMQ | Nick |

# 

## 

# 

## RabbitMQ (Vincent)

### Functionality

**Asynchronous Messaging**

* Multiple messaging protocols
* Message queuing
* Delivery acknowledgement
* Flexible routing to queues
* Multiple exchange type

**Developer Experience**

* Deploy with BOSH, Chef, Docker and Puppet
* Develop cross-language messaging with favorite programming languages such as: Java, .NET, PHP, Python, JavaScript, Ruby, Go, and many others.

**Distributed Deployment**

* Deploy as clusters for high availability and throughput
* Federate across multiple availability zones and regions.

**Enterprise & Cloud Ready**

* Pluggable authentication, authorisation, supports TLS and LDAP
* Lightweight and easy to deploy in public and private clouds

**Tools & Plugins**

* Supports continuous integration
* Operational metrics
* Integration to other enterprise systems
* Flexible plug-in approach for extending RabbitMQ functionality.

**Management & Monitoring**

* HTTP-API
* Command line tool
* UI for managing and monitoring RabbitMQ.

#### Delivery Acknowledgement

Scenario:   
Service A *puts* a message in the queue and service B *expects* a message in the queue.

Whenever service B picks up a message, the queue expects an acknowledgement message back so that it knows that the service has received the message. When service B should fail and the acknowledgement message doesn’t get delivered, the message automatically gets put back into the queue.

#### Protocols

Since RabbitMQ supports several standardized protocols such as AMQP, MQTT, STOMP, etc. allows us to replace the RabbitMQ broker with any AMQP based broker. This means that it offers more functionality in different scenarios as it’s highly replaceable.

#### Routing of messages

RabbitMQ offers a multitude of exchange types such as Direct, Topic, Fanout and Headers which are all covered. The messages sent through these exchanges are then consumed and acknowledged by the queue and consumer which will remove the message from the queue. The ability to route messages in multiple ways makes using RabbitMQ very flexible.

#### Message priority

RabbitMQ also supports something called priority queues, which allows a queue to support a range of priority levels. When a message gets assigned a priority when it is published, a priority queue may take the message earlier.

#### Monitoring

RabbitMQ offers a built-in user-friendly interface that lets you monitor and handle your server from a web browser.

### Learning curve

RabbitMQ is a mature product and thus delivers a very well made [documentation page](https://www.rabbitmq.com/documentation.html) covering all topics that one might use RabbitMQ for. They cover their own functionality in extensively covered self made tutorials that beginners can use in order to get a handle on different types of message handling.

### Non Functional Requirements

|  |  |
| --- | --- |
| **NFR** | **Handling** |
| **Scalability** | If the messages are published faster than they are consumed in RabbitMQ, we can simply scale the number of consumers that are handling the message in said filled queue. So scaling up and down is as simple as adding and removing consumers from a queue. This does mean that RabbitMQ is more focused on vertical scaling(adding more power to the machine) than vertical scaling (adding more machines). |
| **Security** | RabbitMQ supports two major authentication mechanisms as well as several authentication and authorisation backends such as PLAIN, AMQPLAIN, EXTERNAL and RABBIT-CR-DEMO which is not enabled by default. These authorization features allow things such as topic authorisation and management UI access. |
| **Availability** | RabbitMQ features highly available queues that can be mirrored across several machines in a cluster, ensuring that even in the event of hardware failure your messages are safe. |
| **Performance** | RabbitMQ performs best when queues are kept short which they provide a lot of configuration tools for. On top of that, RabbitMQ offers a variety of features to let you trade off performance with reliability, including persistence, delivery acknowledgements, publisher confirms, and high availability |
| **Transferability** | RabbitMQ supports a multitude of messaging protocols, RabbitMQ clients for almost any programming language. |

## Apache Kafka (Maarten)

Kafka is an event streaming platform.

### Functionality

1. Publish (write) and subscribe (read) streams of events, including continuous import/export of data from other systems.
2. Store streams of events durably and reliably for as long as you want
3. Process streams of events as they occur or retrospectively

### Learning curve

Fairly easy to get used to. Works very similar to other messaging services where a plugin is required in your pom.xml to add functionality and a few lines of code which can be imported from the Kafka website.

### Type of messaging

Event streaming, often compared to a human body’s central nervous system. One system to control all the messages going to all the different places within the system. The servers and clients communicate via TCP network protocol.

The data type of the messages are byte arrays, meaning that they can send any form of digital information. The max size of one of these messages is 1MB.

### Max messaging rate

From multiple benchmark tests Kafka reaches around 605 MB/s at a latency of 2ms.

<https://www.confluent.io/blog/kafka-fastest-messaging-system/>

<https://engineering.linkedin.com/kafka/benchmarking-apache-kafka-2-million-writes-second-three-cheap-machines>

### Non functionals

|  |  |
| --- | --- |
| **NFR** | **Handling** |
| **Scalability** | Kafka clusters can scale up to a cluster with a thousand brokers, which means trillions of messages per day, petabytes of data, and hundreds of thousands of partitions. This can also be downscaled when the clusters are no longer needed. |
| **Security** | Kafka security is natively integrated into any Kafka project. It encrypts data in transit between applications and kafka brokers. It also provides client authentication and authorization. An SSL (Secure Sockets Layer) certificate is required. |
| **Availability** | Because all the messages are stored, both past and present messages can be viewed at any point. |
| **Performance** | Currently from several different sources that have been benchmarking the different messaging frameworks, Kafka seems to handle more messages, bigger message size and faster transfer speeds. |
| **Transferability** | Kafka supports C/C++, Go, Java, .NET and Python. However not all features are implemented for all the programming languages. The most supported language is Java. |

## 

## Azure Service Bus (Jursley)

Azure Service Bus is a cloud based message broker with message queues and publish-subscribe topics used to connect any applications, devices, and services running in the cloud to any other applications or services.

### Functionality

* Scheduling
* Dead-lettering
* Transactional processing
* Message ordering
* Expiration (TTL)
* Duplicate detection
* Deferring
* Pub/Sub
* Filtering

One important difference with other message brokers is that you don’t have to worry about the following actions. Azure takes care of those chores for you.

* Post logs and manage disk space
* Handling backups
* Keep the operating systems or products patched
* Worrying about hardware failure
* Failure to a backup machine

### Learning curve

Microsoft offers a multiplicity of documentation on the Azure Service Bus starting from beginner topics to advanced topics explained in a detailed way with example codes. Because of the maturity of the service if you are acquainted with other service bus technologies it is easy to pick up because of similarities.

### Type of messaging

JSON, XML, Apache Avro, plain text.

Some common message scenarios are:

* Messages. Transfer business data, such as sales or purchase orders, journals, or inventory transactions.
* Unlink applications. Improve the reliability and scalability of applications and services. Producer and consumer do not have to be online or directly available at the same time. The load is leveled in such a way that traffic peaks do not overload a service.
* Load Balancing. Allow multiple competing consumers to read from a queue simultaneously, with each securely gaining exclusive ownership of specific messages.
* Topics and subscriptions. Enable 1: n relationships between publishers and subscribers so that subscribers can select specific messages from a published message stream.
* Transactions. It allows you to perform a variety of operations, all in the context of an atomic transaction. For example, the following operations can be performed within the scope of a transaction.

1. Get a message from one queue.
2. Post the results of the processing to one or more different queues.
3. Move the input message from the original queue.

* The results only become visible to downstream consumers after success, including the successful completion of the input message, allowing for a one-time semantics to be processed. This transaction model is a robust basis for the compensating transaction pattern in the larger solution context.
* Message sessions. Implement large-scale coordination of workflows and multiplex transfers that require strict message sequence or message deferral.

### Max messaging rate

Azure service bus has 2 tiers when it comes to Message sizes. For the standard this is up to 256KB per message, for the premium it is up to 1 MB per message.

**Non functionals**

|  |  |
| --- | --- |
| **NFR** | **Handling** |
| **Scalability** | Partitioned Queues (and topics)lets you send more messages over multiple queues but look like a single queue to your API. So the more traffic the more queues you can use.  With Azure auto scaling capabilities you can automatically monitor the messages in a queue (or CPU load) and add/remove instances to maintain that target. You can use Azure Service Bus Automation and Monitoring by CloudMonix to help you scale and monitor queues. CloudMonix lets you Scale compute resources in real-time based on demand indicated by Service Bus queues or topics |
| **Security** | You can protect Azure resources with virtual networks thanks to the integration with the Azure Private Link service. This enables secure private access to messaging capabilities of workloads such as virtual machines bound to virtual networks. Create a private endpoint connection to your Service Bus namespace. The private endpoint uses a private IP address of your virtual network, effectively bringing the service into your virtual network. All traffic to the service can be routed through that private endpoint, so no gateways, NAT devices, ExpressRoute or VPN connections, or public IP addresses are required. |
| **Availability** | Azure Service Bus spreads the risk of catastrophic failures of individual machines or even entire racks across clusters spanning multiple failure domains within a data center and implements transparent failure detection and failover mechanisms to keep the service operating within guaranteed service levels and typically without noticeable interruptions when such malfunctions occur.  The Geo-Disaster Recovery feature ensures that the entire configuration of a namespace (queues, topics, subscriptions, filters) is continuously replicated from a primary namespace to a secondary namespace when attached, and it allows you to create a one-time failover move to initiate from primary to secondary at any time. |
| **Performance** | Best practices to boost performance are :   * Choosing the right protocol for a specific job (Advanced Message Queuing Protocol, The proprietary Service Bus Messaging Protocol or HTTP/S) * Perform operations - such as send, receive, and delete - asynchronously. This concurrent processing, through the SendAsync messaging method in Service Bus, allows applications to perform more tasks in a given time period than when operations are performed serially. |
| **Transferability** |  |

## Amazon MQ (Faruk)

Amazon MQ is a managed message broker service for Apache ActiveMQ and RabbitMQ that makes it easy to set up and operate Message Brokers on AWS.

### Functionality

* Managed Service
* Security
* Monitoring
* Broker Instance Types
* Pay-as-you-go Pricing
* Get Started for Free

RabbitMQ Features

* High Availability, Throughput, and Message Durability
* Message Routing
* Broad Client Language Support
* Other features <https://www.rabbitmq.com/#features>

ActiveMQ Features

* High Availability, Throughput, and Message Durability
* Industry-standard APIs and Protocols
* JMS Messaging Features
* Other features <http://activemq.apache.org/components/classic/>

### Learning curve

There is a [resource](https://aws.amazon.com/amazon-mq/resources/?mq-blogs.sort-by=item.additionalFields.createdDate&mq-blogs.sort-order=desc) page. Starters can visit this page to get educated about the usage of Amazon MQ. It is easy to start with Amazon MQ because of the amount of information you can get as a starter.

### Non functionals

|  |  |
| --- | --- |
| **NFR** | **Handling** |
| **Scalability** | With Amazon MQ you can scale your messaging middleware horizontally, vertically, or in a hybrid model. Horizontal scaling enables you to increase your throughput and connection count without interruptions, because your resources remain active and online. Vertically, you can increase the compute capacity of your broker instances from 1 vCPU and 1 GiB up to 16 vCPU and 64 GiB. |
| **Security** | There are some recommended design patterns. These are:   * Prefer brokers without public accessibility * Always configure an authorization map * Block unnecessary protocols |
| **Availability** | The choice to have two brokers in two different Availability Zones, configured in a redundant pair. These brokers communicate synchronously with the application. Only one of the brokers is active. The other is in standby mode. The standby one can jump into action when the active one is out for a reason. |
| **Performance** | There are some performance issues with large messages. That’s why it’s recommended to use shorter messages. |
| **Transferability** | Amazon MQ can be used to transfer your existing applications with messaging to the cloud quickly and easily. It supports industry-standard APIs and protocols. |

## ZeroMQ (Nick)

ZeroMQ is an open source, high-performance asynchronous messaging library, aimed at use in distributed or concurrent applications.

The zero stands for multiple things in the ZeroMQ philosophy. No broker, no latency, no costs and no administration.

### Functionality

* Message queue, without broker (unlike other message-oriented middleware solutions)
* Supports common messaging protocols
  + Publish/Subscribe
  + Request/Response
    - Synchronous
    - Asynchronous
  + Push/Pull
  + Exclusive pair
* Variety of transport options
  + Inter-process
  + TCP
  + WebSocket
  + In-process
  + Multicast

**Learning curve**

On the ZeroMQ website there is a getting started guide. With support libraries for 28 languages there will be a library for your needs. For each language there are a lot of examples of how to implement the ZeroMQ library.

You can have something up and running in just a few lines of code. The way ZeroMQ works is quite simple. It mostly works via sockets. You just bind the server/publisher to a port and then connect the worker (pull) to the same port. And you are ready to send and receive messages.

### 

### Non functionals

|  |  |
| --- | --- |
| **NFR** | **Handling** |
| **Scalability** | ZeroMQ is a lightweight library which when scaled will not use too many resources. Internally it uses |
| **Security** | ZeroMQ has its own wire protocol (ZMTP 3.0), its own security protocol (CurveZMQ). The developer can configure the connection as secure as he wants, from no security to a basic username and password to ZeroMQ’s Curve protocol. The CurveZMQ protocol is fast and considered strong. |
| **Availability** | Whenever a part of the system fails another part will take over the load without any problems. |
| **Performance** | ZeroMq is one of the fastest messaging protocols out there. This only is the case if the library is used properly. |
| **Transferability** | It is very easy for a developer to start using ZeroMQ. The API is written in a way that it is easy-to-use cross-language. The library handles all the complex operations which requires the developer not to learn the whole API. |

# Matrix

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Evaluation Matrix** | **Weight** | **RabbitMQ** | **Apache Kafka** | **Azure Service Bus** | **Amazon MQ** | **ZeroMQ** |
| **Ease of Use** | 0.1 | **0.9** (10) | **0.8** (8) | **0.7** (7) | **0.8** (8) | **1.0** (10) |
| **Functionality** | 0.1 | **0.9** (9) | **1** (10) | **0.9** (9) | **1** (10) | **0.7** (7) |
| **Learning Curve** | 0.1 | **0.9** (9) | **0.8**(8) | **0.9** (9) | **0.8** (8) | **0.9** (9) |
| **Scalability** | 0.2 | **1.6** (8) | **1.8** (9) | **1.8** (9) | **1.6** (8) | **1.4** (7) |
| **Security** | 0.1 | **0.8** (8) | **0.8** (8) | **0.8** (8) | **0.7** (7) | **0.9** (9) |
| **Availability** | 0.2 | **1.6** (8) | **1.8** (9) | **1.8** (9) | **1.8** (9) | **1.4** (7) |
| **Performance** | 0.15 | **1.2** (8) | **1.5** (10) | **1.05** (7) | **0.9** (6) | **1.05** (7) |
| **Transferability** | 0.05 | **0.45** (9) | **0.4** (8) | **0.4**(8) | **0.4** (8) | **0.45** (9) |
| Total score: | 1 | **8.35** (69) | **8.9** (70) | **8.35** (66) | **8** (64) | **7.8** (64) |

# 

# Conclusion

When comparing the different tools in the matrix, we discovered that a lot of similar traits were being shared between the tools themselves. They broadly offer the same functionality overall and differ very little functionality wise as some components from one could simply be added in another, but it may not be their primary feature.

|  |  |
| --- | --- |
| **Tool Choice: Case** | **Apache Kafka** |
| **Tool Choice: Group Project** | **RabbitMQ** |

One of the primary non functional requirements that the case wanted was robustness in its system. Apache Kafka is known for their replayability in their messaging system in which the system can keep all of the messages in a single or multitude of files. This main feature was the breaking point for our choice as Kafka offers almost all of the features that the case wants.

We did however, assume herein that the developers in the case would be seasoned developers as stated in the beginning which would allow for a quicker setup of the whole system as it can be a tad bit more difficult to start out with and require more knowledge from the team.

This is also one of the reasons why we chose RabbitMQ for our group project as we did not want to invest too much time into learning and setting up the system within our limited time frame. Although the replayability of Kafka seems great, RabbitMQ is not lacking in features for our application purposes which may be more simple. The replayability can then later be added if the feature is very desired which may help for transactional messages within the Energy Grid system.

Another was RabbitMQ integration and support with our existing technology which uses Java Spring Boot for which they have good support and tutorials for as they use a standard AMQP protocol. In addition to that, RabbitMQ offers a nice and simple UI where one can see the messages and the queue interactions, which is very helpful for debugging.

All in all, we may have valued the learning curve in our group project a bit higher than expected which heavily influenced our decision. RabbitMQ delivers enough performance for our application without hindering us with a knowledge wall which allows us to develop the application faster.