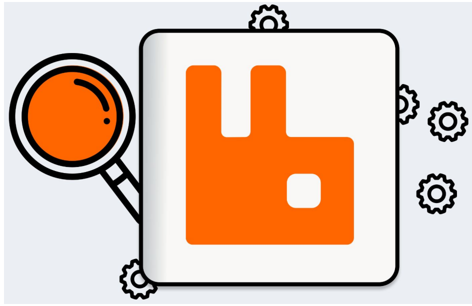
Message Broker: RabbitMQ

[Waheed Iqbal]

DevOps (Fall 2023)
Department of Data Science, FCIT, University of the Punjab
Lahore, Pakistan



Message Broker

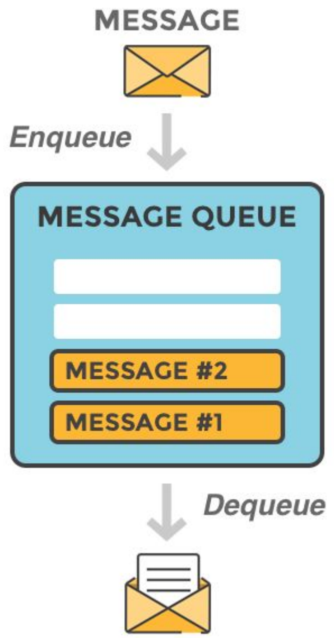
- Message brokers facilitate communication between applications/systems by sending, receiving, and routing messages.
- Help to decouple the sender and receiver of a message, enabling more flexible and resilient architectures.
- Improve the scalability and performance of distributed systems by reducing coupling, also promote async communication among services.
- Apache Kafka, RabbitMQ, Amazon SQS, Google Pub/Sub, Azure Service Bus, and ActiveMQ are commonly used Message Brokers.

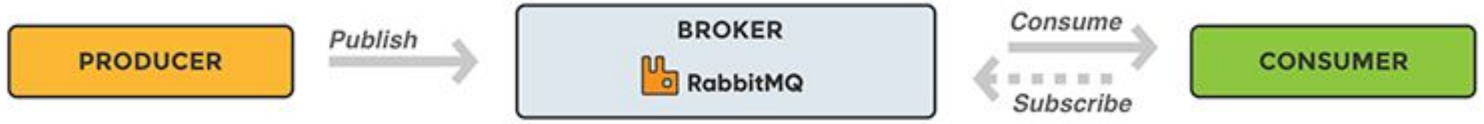
Message Broker Components

- A producer is an app responsible for transmitting messages. It's linked to the message broker, also known as publishers in the publish/subscribe model.
- A consumer is a service that receives messages waiting in the message broker. They’re known as subscribers in the publish-subscribe pattern.
- A queue or topic is a logical channel or container where messages are temporarily stored before they are consumed by the intended receiver(s).
- Exchange is a logical entity in a message broker that routes messages to the appropriate queues or topics based on their routing key or other message attributes.

RabbitMQ

- RabbitMQ is an open-source message broker.
- Supports multiple messaging protocols and patterns, including point-to-point and publish-subscribe messaging.
- Uses AMQP (Advanced Message Queuing Protocol) as its messaging protocol, which is a widely adopted industry standard.
- Offers features such as message persistence, message acknowledgement, and message routing, which ensure reliable and fault-tolerant message delivery.
- Supports a wide range of client libraries and can be integrated with various programming languages and platforms.
- Widely used in various industries and applications, including finance, healthcare, e-commerce, and social media.


MESSAGE
Enqueue
MESSAGE QUEUE
MESSAGE #2
MESSAGE #1
Dequeue


PRODUCER
Publish
BROKER
RabbitMQ
Consume
Subscribe
CONSUMER

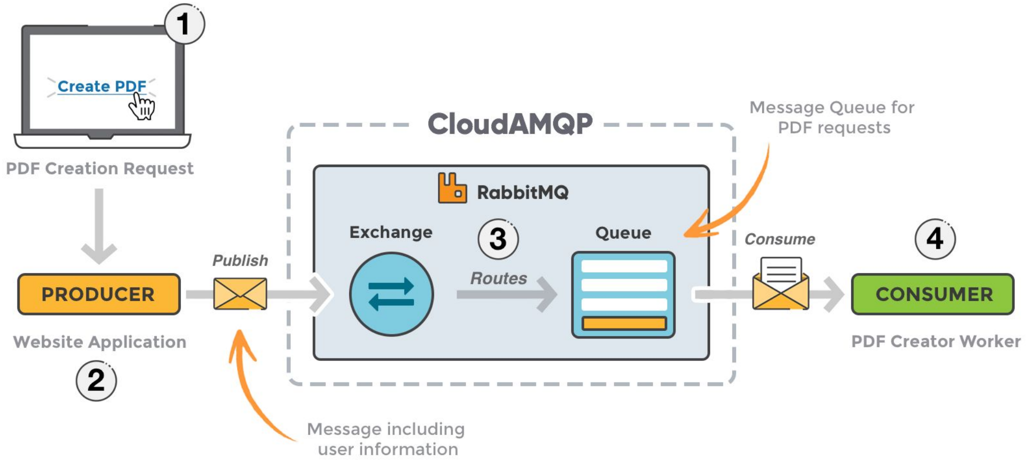
Message Queue Use Cases

When requests take a significant amount of time, it is the perfect scenario to incorporate a message queue. Some real-life examples could include:

- Sending large/many emails
- Search engine indexing
- File scanning
- Video encoding
- Delivering notifications
- PDF processing
- Calculations

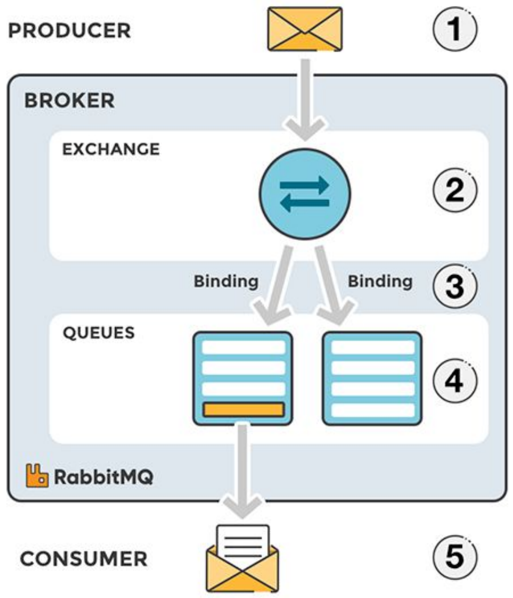
RabbitMQ Example

1. The user sends a PDF creation request to the web application.
2. The web application (the producer) sends a message to RabbitMQ that includes data from the request such as name and email.
3. An exchange accepts the messages from the producer and routes them to correct message queues for PDF creation.
4. The PDF processing worker (the consumer) receives the task message and starts processing the PDF.



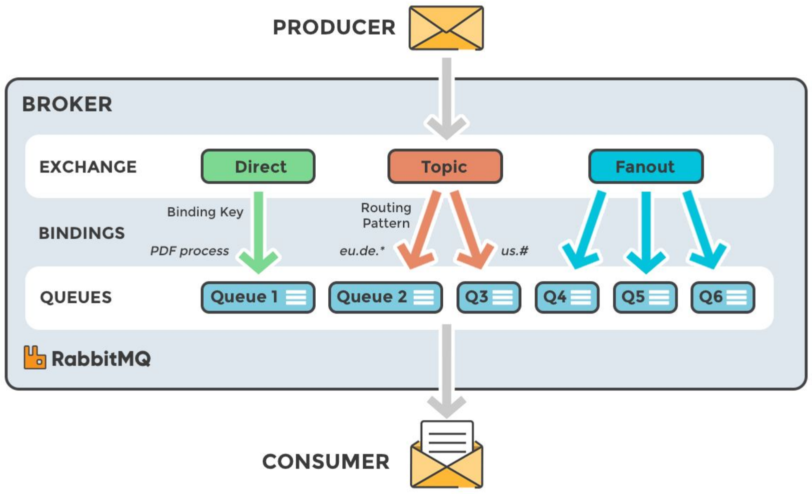
Exchange

- Messages are not published directly to a queue; instead, the producer sends messages to an exchange.
- An exchange is responsible for routing the messages to different queues with the help of bindings and routing keys.
- Routing key is a property of the message that determines its routing, and the binding key is a property of the queue that determines which messages the queue is interested in receiving.



Type of Exchanges

- Direct: The message is routed to the queues whose binding key exactly matches the routing key of the message. For example, if the queue is bound to the exchange with the binding key pdfprocess, a message published to the exchange with a routing key pdfprocess is routed to that queue.
- Fanout: A fanout exchange routes messages to all of the queues bound to it.
- Topic: The topic exchange does a wildcard match between the routing key and the routing pattern specified in the binding.
- Headers: Headers exchanges use the message header attributes for routing.



RabbitMQ Cluster

- A RabbitMQ cluster is a group of two or more nodes (servers) that work together to provide a highly available, scalable, and fault-tolerant messaging system.
- Messages can be replicated across all nodes in the cluster, ensuring that they are always available and can be delivered even if one or more nodes fail.
- Clustering allows you to add or remove nodes from the cluster without disrupting the service, making it possible to scale the service as traffic increases.

RabbitMQ Cluster (Cont.)

- RabbitMQ provides mechanisms for topic and queue partitioning, which allow you to split a topic or queue into multiple partitions and distribute messages across multiple nodes in the cluster.
- Queue partitioning is achieved using a feature called "Sharding", which allows you to split a queue into multiple shards and distribute messages across multiple nodes in the cluster.
- RabbitMQ's Consistent Hash Exchange plugin allows you to distribute messages between multiple queues based on a consistent hash of the message properties.
- These mechanisms require additional configuration and setup beyond the basic RabbitMQ installation.

RabbitMQ High Availability

- RabbitMQ provides high availability features that ensure that messages are delivered even if one or more nodes in the cluster fail.
- RabbitMQ clusters can be configured with mirrored queues, which replicate queues across multiple nodes in the cluster.
- In a mirrored queue (uses master-slave style), messages are automatically replicated to all nodes in the cluster, so if one node fails, the messages can still be delivered from another node.
- RabbitMQ also provides a feature called "Quorum Queues", which provide more robust and performant mirroring compared to traditional mirrored queues.
- Quorum Queues use a consensus-based replication protocol to replicate messages across nodes in the cluster, which ensures that messages are not lost even in the event of multiple node failures.
- Additionally, RabbitMQ provides features like automatic node recovery, automatic failover, and network partition handling, which further improve the availability of the messaging service.

RabbiMQ in Kubernetes

https://www.rabbitmq.com/kubernetes/operator/quickstart-operator.html

Credit

https://www.cloudamqp.com/blog/part1-rabbitmq-for-beginners-what-is-rabbitmq.html