Verteilte Systeme & Cloud-Technologien

02 – RabbitMQ

Introduction

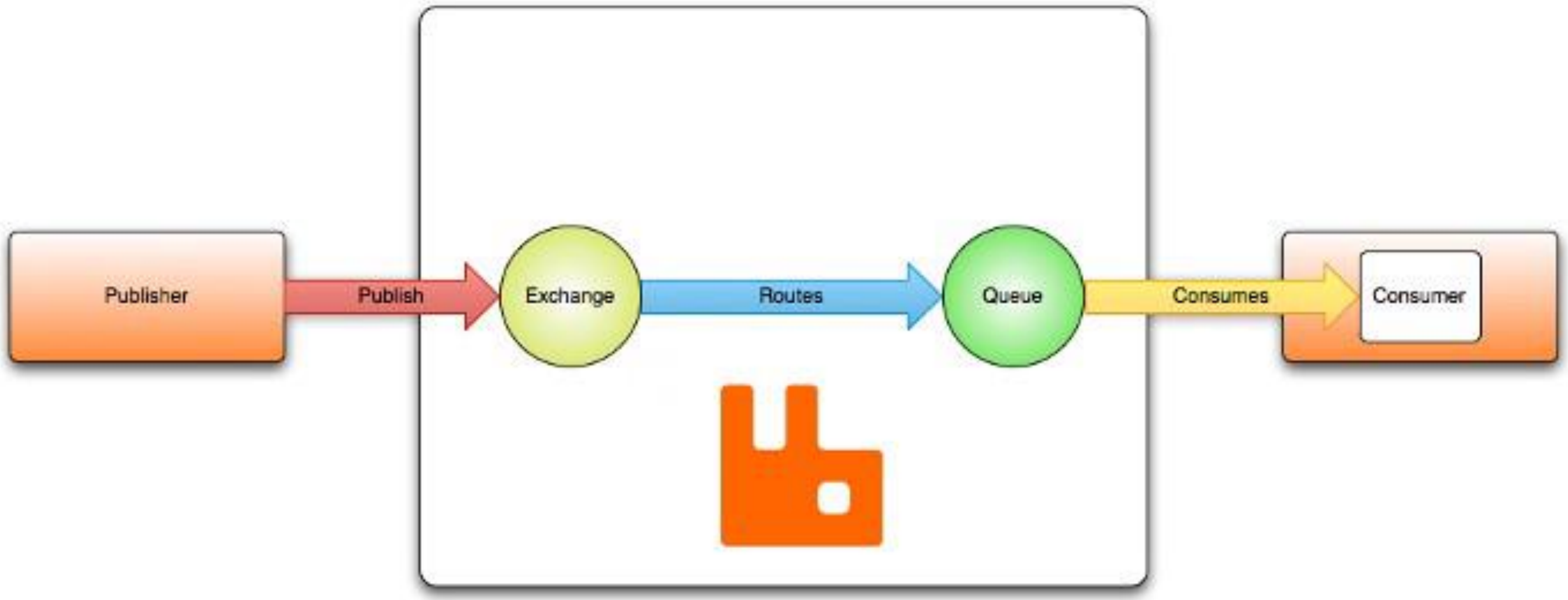
- An Open source message broker written in Erlang
- Implements the AMQP Protocol(Advanced message Queuing protocol)
- Maintained by VMWare (…)
- The source code is released under the Mozilla Public License.

Why are message broker important?

Message brokers provide these core advantages in distributed systems:

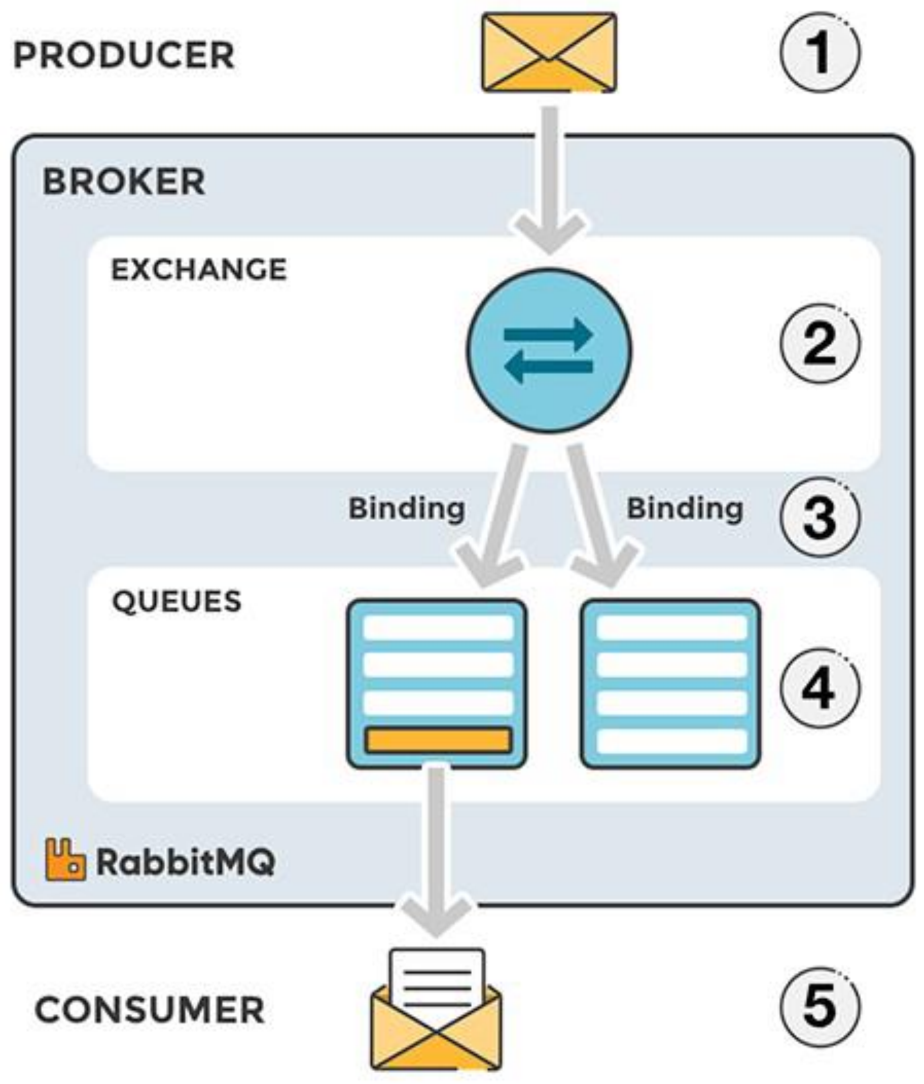
- **Decoupling**: Producers and consumers don't need to know each other, be available simultaneously, or scale at the same rate.
- **Reliability**: Message persistence, retry mechanisms, and dead letter queues prevent data loss during failures.
- **Scalability**: Horizontal scaling through partitioning, load balancing across multiple consumers, and backpressure management.
- **Flexibility**: Support for various messaging patterns (pub/sub, request/reply, streaming) and dynamic service addition/removal without system redesign.
- **Performance**: Asynchronous processing, batching, and efficient message routing algorithms.

RabbitMQ Overview



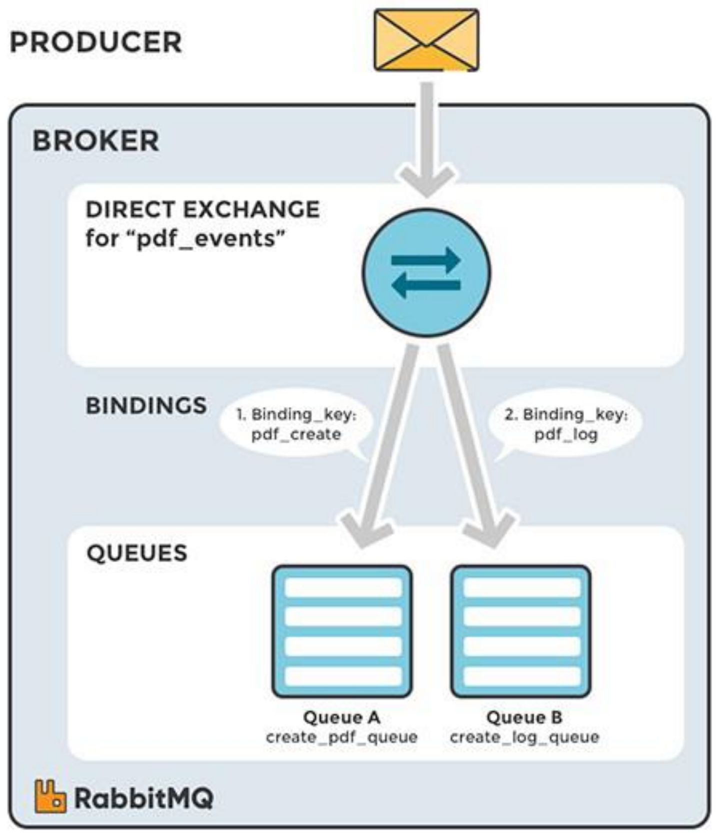
Message Flow

- The producer publishes a message to the exchange
- The exchange receives the message and is now responsible for the routing of the message.
- A binding has to be set up between the queue and the exchange.
- The messages stay in the queue until they are handled by a consumer
- The consumer handles the message.



Direct Exchange

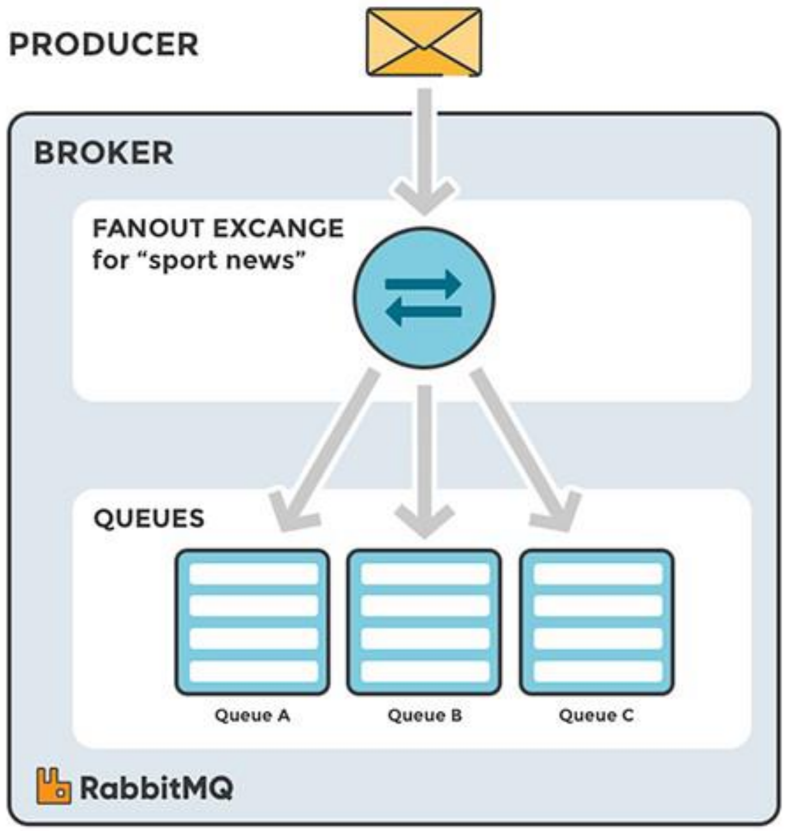
A direct exchange delivers messages to queues based on the message routing key. A direct exchange is ideal for the unicast routing of messages



Default Exchange

- The default exchange is a direct exchange with no name (empty string) pre-declared by the broker. It has one special property that makes it very useful for simple applications: every queue that is created is automatically bound to it with a routing key which is the same as the queue name.
- For example, when you declare a queue with the name of "search-indexing-online", the AMQP 0-9-1 broker will bind it to the default exchange using "search-indexing-online" as the routing key (in this context sometimes referred to as the binding key). Therefore, a message published to the default exchange with the routing key "search-indexing-online" will be routed to the queue "search-indexing-online". In other words, the default exchange makes it seem like it is possible to deliver messages directly to queues, even though that is not technically what is happening.

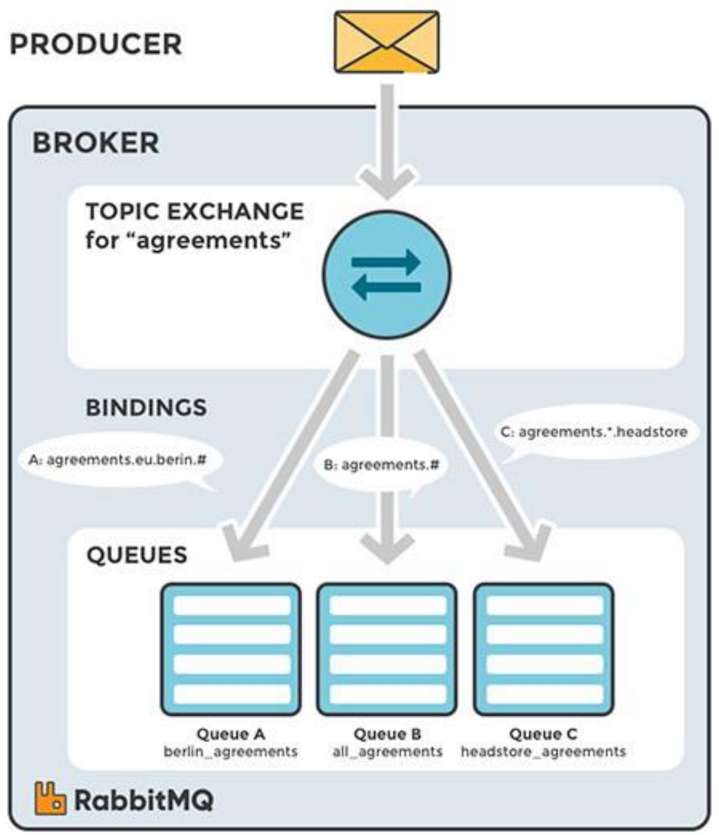
Fanout Exchange



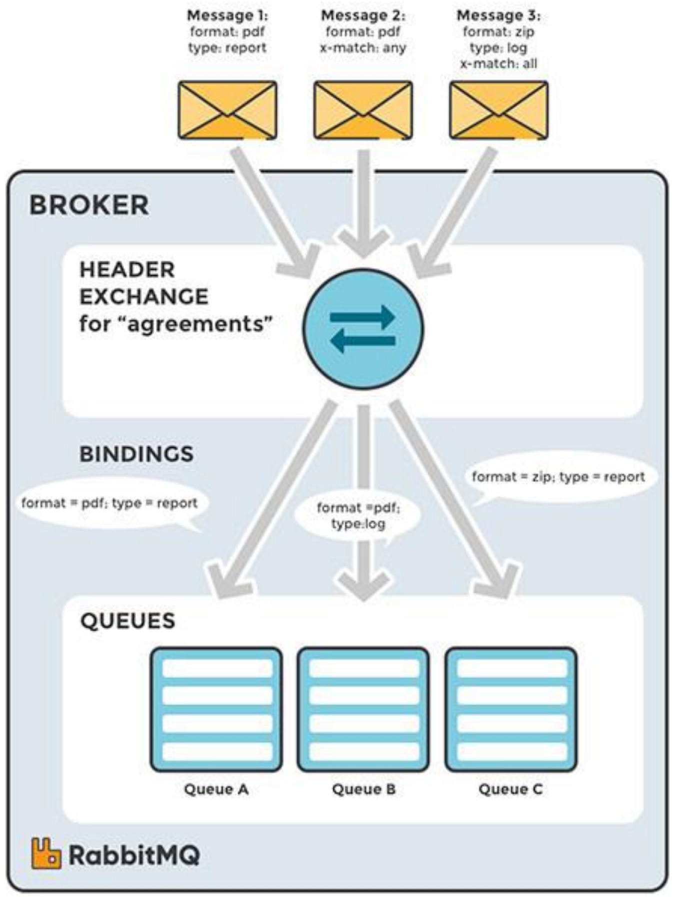
A fanout exchange routes messages to all of the queues that are bound to it and the routing key is ignored. If N queues are bound to a fanout exchange, when a new message is published to that exchange a copy of the message is delivered to all N queues. Fanout exchanges are ideal for the broadcast routing of messages.

Topic Exchange

Topic exchanges route messages to one or many queues based on matching between a message routing key and the pattern that was used to bind a queue to an exchange Topic exchanges are commonly used for the multicast routing of messages.


PRODUCER
BROKER
TOPIC EXCHANGE for "agreements"
BINDINGS
A: agreements.eu.berin.#
B: agreements.#
C: agreements.*.headstore
QUEUES
Queue A berlin_agreements
Queue B all_agreements
Queue C headstore_agreements
RabbitMQ

Header Exchange



Headers exchanges ignore the routing key attribute. A headers exchange is an exchange which routes messages to queues based on message header values.

Getting started in C#

Client Library supporting C#

- https://www.rabbitmq.com/client-libraries/dotnet

MassTransit is an open-source framework for .NET that simplifies asynchronous communication between distributed systems and services through message-based interactions, also with RabbitMQ.

- https://masstransit.io/quick-starts/rabbitmq

Links

- https://www.rabbitmq.com/
- https://www.cloudamqp.com/blog/part4-rabbitmq-for-beginners-exchanges-routing-keys-bindings.html
- https://masstransit.io/quick-starts/rabbitmq

Practice makes perfect.