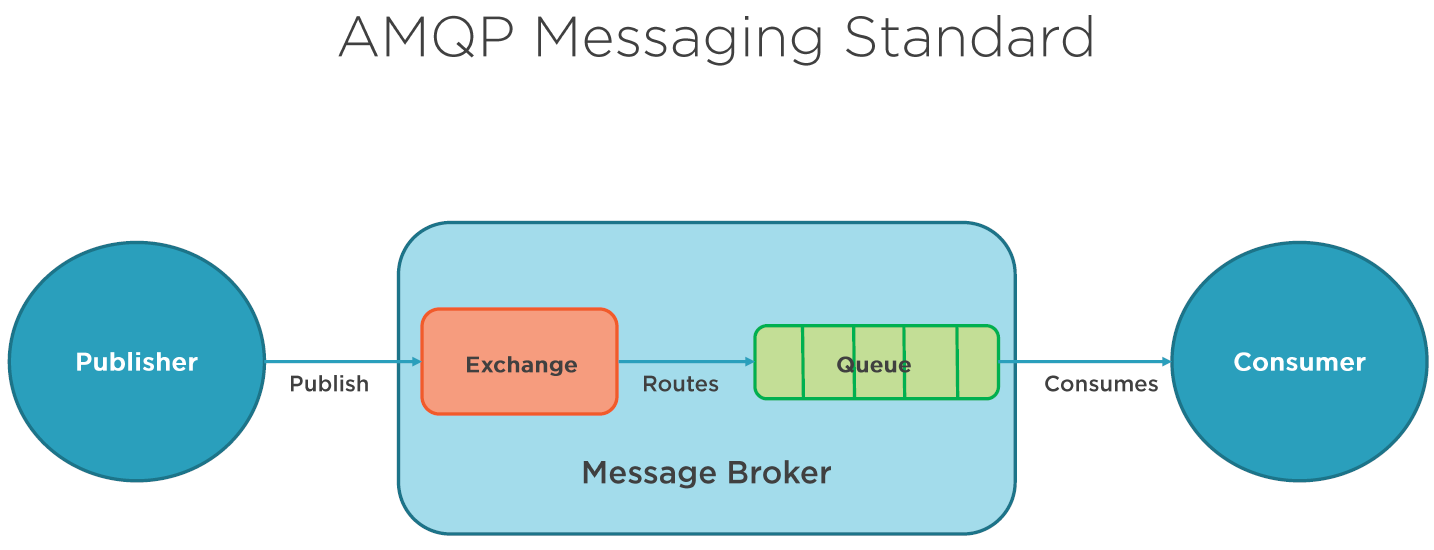
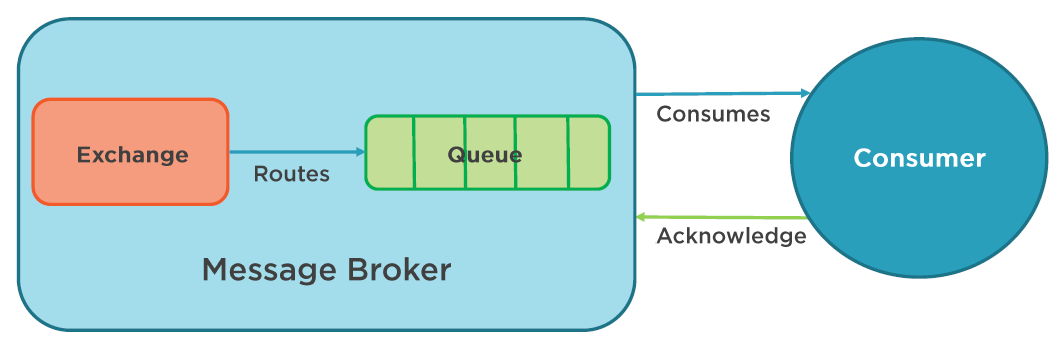
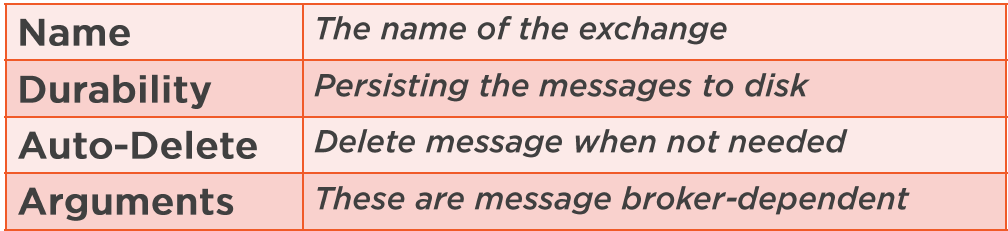
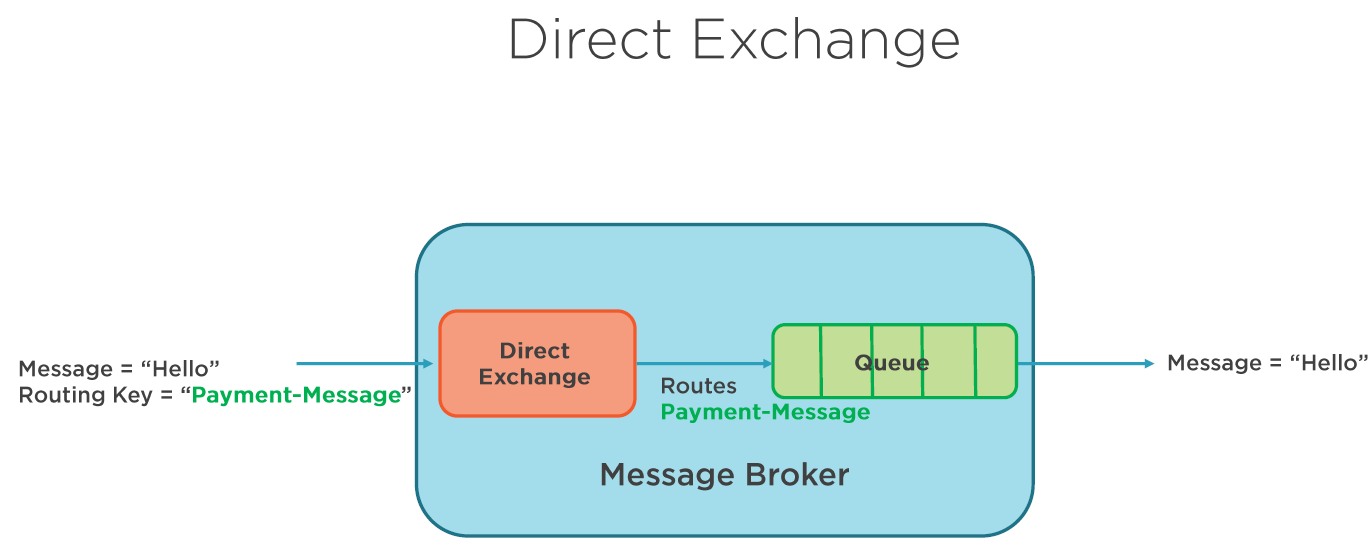
RABBITMQ:

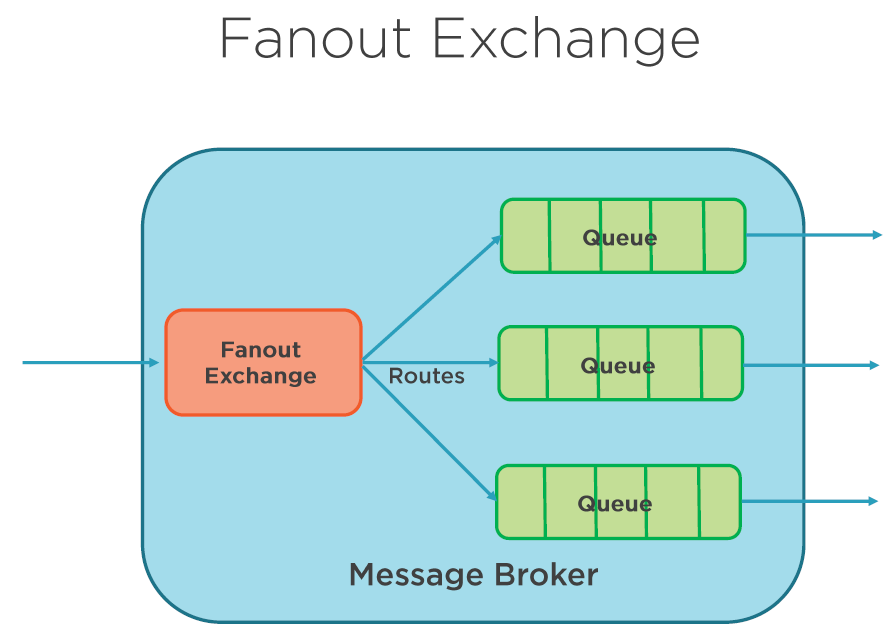
* Opensource messaging system that will allow you to integrate applications together using messages and queues.
* RabbitMQ server is message broker that acts as a message coordinator for applications that you want to integrate together.(common platform for sending and receiving messages)
* Reliability, Salability ,Routing , clustering, cross platform and High Availability. Management Web Interface, Command line interface (rabbitmqctrl,rabbitmq admin,..)
* RabbitMQ is Opensource messaging system that allows to integrates application together by exchanges and queues.
* RabbitMQ implements AMQP (Advanced message queuing protocol)
* RabbitMQ written in Erlang for telecom industry by ercsion. Erlang supports distributed fault tolerance application.so chosen for messaging system.
* Payed RabbitMQ by Pivotal company. (<https://pivotal.io>)
* MSMQ (1997): it is a messeging protocol that allows applications running on separate servers and processes to communicate in a fail safe manner. Queue is a temporary storge location where message can be sent and received.this enables communications across network with windows computers only.
  + Reliable delivery of messages
  + Supports security & Priority based messaging
  + Places failed messages into separate queues
  + Supports transactions > it allows multiple operations in multiples queues , all in single transaction. This allows all or none of the operations takes effect.

|  |  |
| --- | --- |
| RABBITMQ | MSMQ |
| Centralized (messages are stored in central server or cluster of servers.client sends message to central server and subscribe can receive it) | Decentralized(each machine as its own queues, client sends message on incoming message queue on another server.each machine as its own sents of queues and services it can access) |
| Multi Platform | Windows only |
| Standards (AMQP) | No Standards |

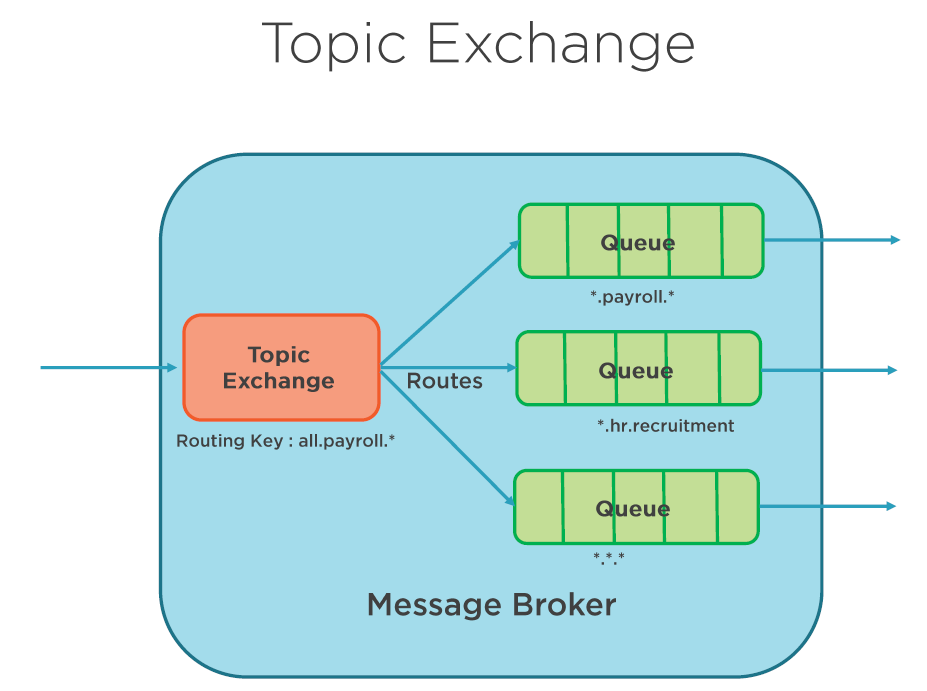
* RABBITMQ MANAGEMENT PORTAL:
  + UI> rabbitmq-plugins enable rabbitmq\_management
  + Declare, list and delete rabbitmq entities
  + Queue and exchange monitoring
  + Sending and receiving messages
  + Monitoring Erlang processes, file descriptors and memory use
  + Force close connections and purge queues
* **AMQP PROTOCOL**
  + **EXCHANGES:** they AMQP entities where messages are sent to message broker
    - Direct Exchange
    - Fanout Exchange
    - Topic Exchange
    - Headers Exchange
  + **QUEUES, BINDINGS, and CONSUMERS**
* 
* Exchnage is like a message box. Exchange send message to a queue by different routes called bindings.consumer pulls the message .
* When a message is published to a queue, a publisher can specify various messaging attributes.some of th attributes used by message broker, rest is opaque(used by consumer).
* 
* Sometime applications applications could fail to correctly processes messages.so AMQP as mechanism for message acknowledgements. Acknowledgement can be automatic or as soon as application developer choose to do so.
* When message acknowledgment is used, the message is only removed from the message broker when it receives notification for that message.
* Messages are routed using routing key (like an address used by an exchange to send message to particular queue)
* EXCHNAGES takes messages and route to one or more queues. Type of routing depends on exchange type used in different exchange rules called bindings.
* Each type of **4 EXCHANGES** has attributes**:**
  + 
  + Default Exchange is Direct



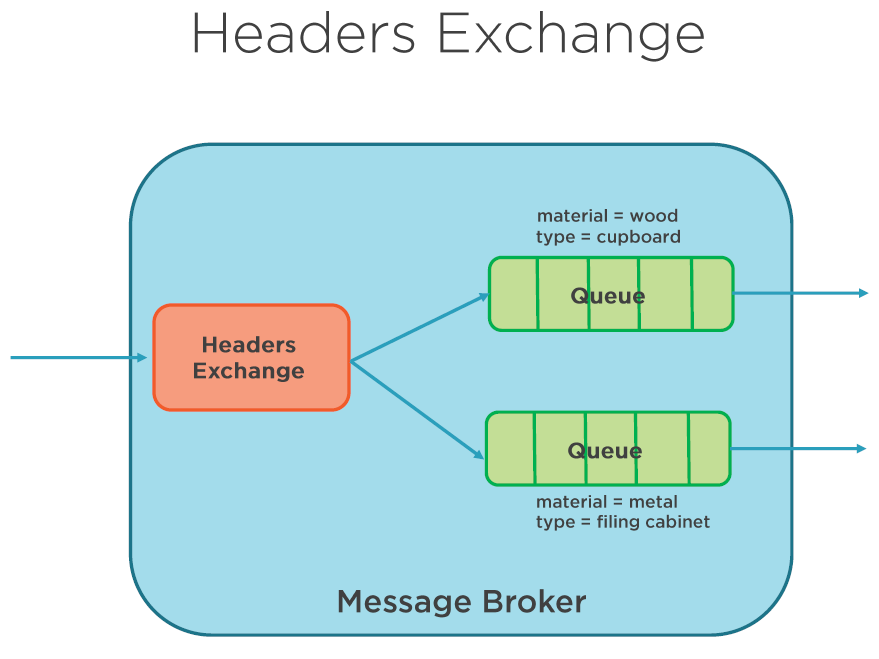
* Here queue binds with an exchange using “routing-key”.so message is arrived with registered key then exchange route to queue of both keys match.
* DIRECT QUEUES: commonly used to distributed messages between multiple worker processes in round-robin manner.



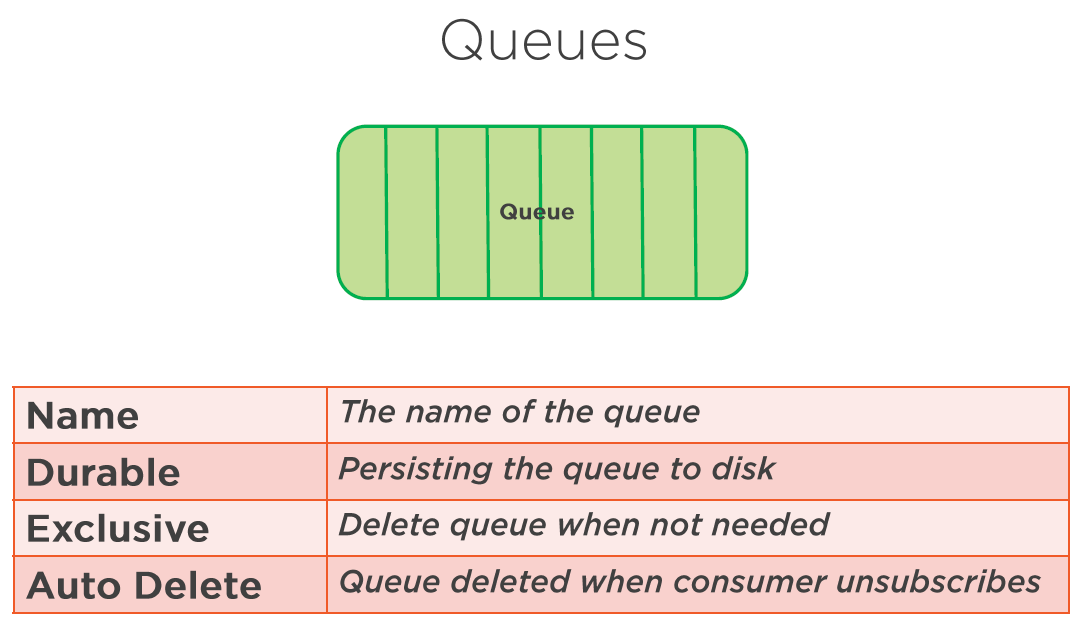
* FANOUT EXHANGE routes messages to all queues that are bound to it.
* Unlike DIRECT exchange , here routing key is ignored.
* Whenever a new message is published to the exchange, copy of that message is delivered to all of the queues.
* FANOUT exchange is ideal for broadcast of messages to all of the queue consumers. Eg: sending online game scores to all players, sending whether updates to all interested systems, chat sessions between groups of people.



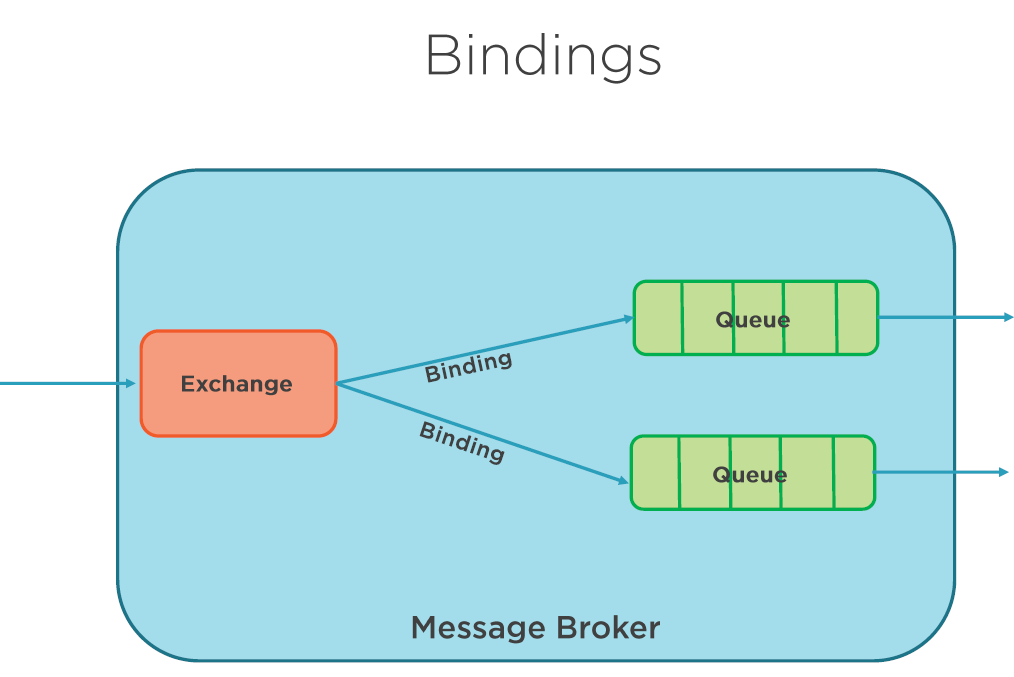
* TOPIC EXCHNAGE route messages to one or many queues by matches between message routing key and pattern that binds queue to an exchange.
* It is often used to implement various publisher square pattern variations.
* Commonly used for multicast routing of messages to different queues.
* This types has broad set of usecases.
* Whenever a problem involved multiple consumers/applications to selective choose which type of message they want to receive, consider TOPIC exchange.
* It works with wild carded routing key. Eg: In the context of companies departmental hierarchy, if you sent a message to
  + all.\*.\* > message will be sent to all departments
  + all.payroll.\* > message will be sent to all consumers interested in payroll department
* it is so powerful, it can behave like other exchanges.
* When a queue is bound to hash binding key, it will receive all messages regardless of the routing key. Exactly same manner as FANOUT exchange.
* Whenever special character #,\* is not used in bindings, TOPIC exchange behave just like DIRECT exchange.
* Usage Eg: sending messages to relevant department in an organization, categorize new updates like business,regular,sports,..



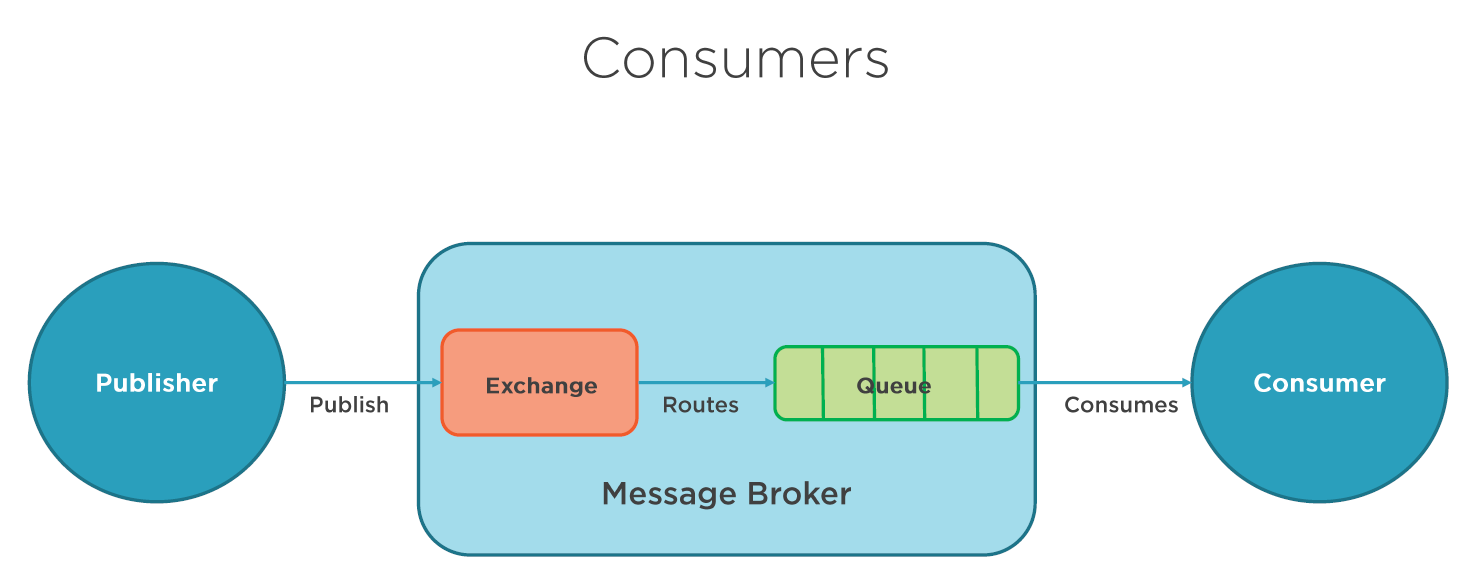
* HEADERS exchange is for routing multiple attributes that are expressed in headers
* Routing-key is ignored for this exchange due to it can only express one piece of information
* You can assign multiple headers values are passed into the exchange and routed to the queues
* It is looked like a supercharged DIRECT exchanges as a route based on header values
* This can also be used as DIRECT exchanges but the routing-key does not have to be string



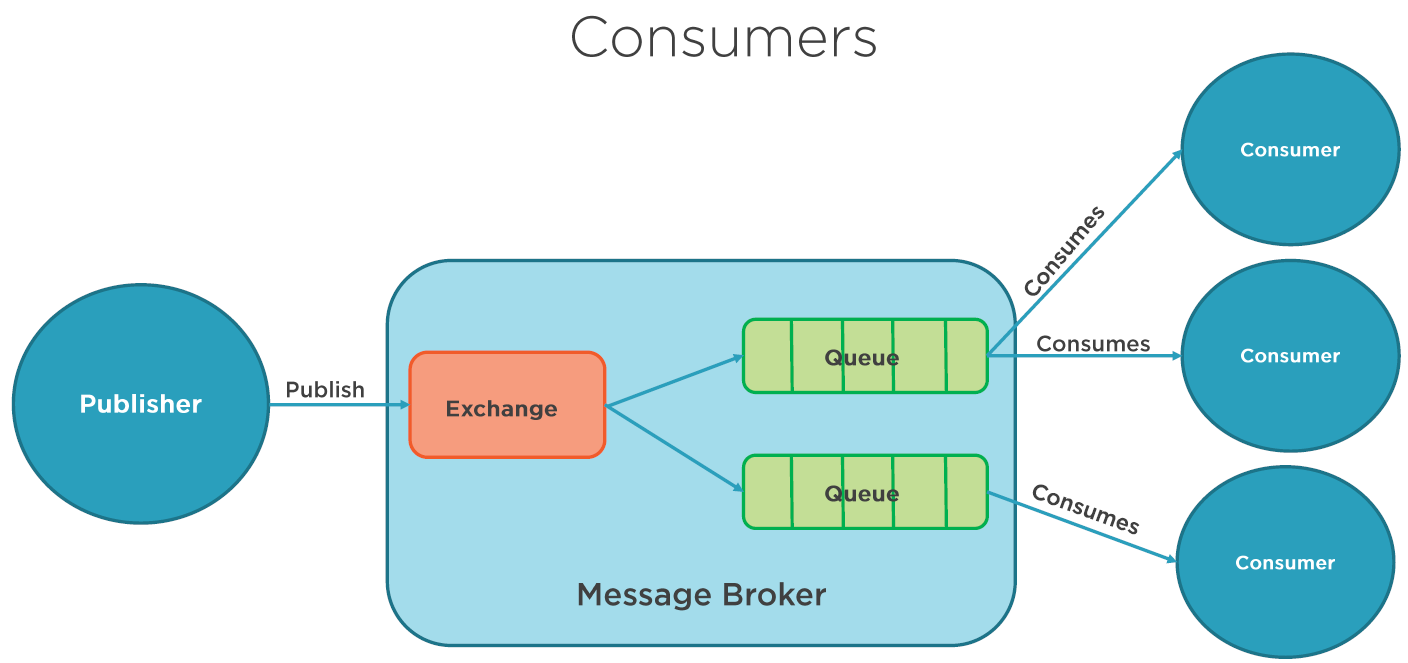
* Before using queue , it must be declared. If the queue not exists , it will be created. If the queue already exists then no addition effect on the queue happens
* QUEUES can be name by the application or they can automatically named by the broker that generates them.
* QUEUE name length upto 255 chrs.
* QUEUE is durable means QUEUE persisted to disk. This only makes QUEUE persistant and not messages.
* QUEUE durability means QUEUE will be redeclared once the broker is restarted.
* If you messages also to be persisted then you have to post persisted messages. This is useful to remain messages intact if the broker/server is restarted
* Making QUEUE durable comes with additional overhead.so decide before enabling. But if your application cant be in a position where it cant loose messages they you need QUEUE durability



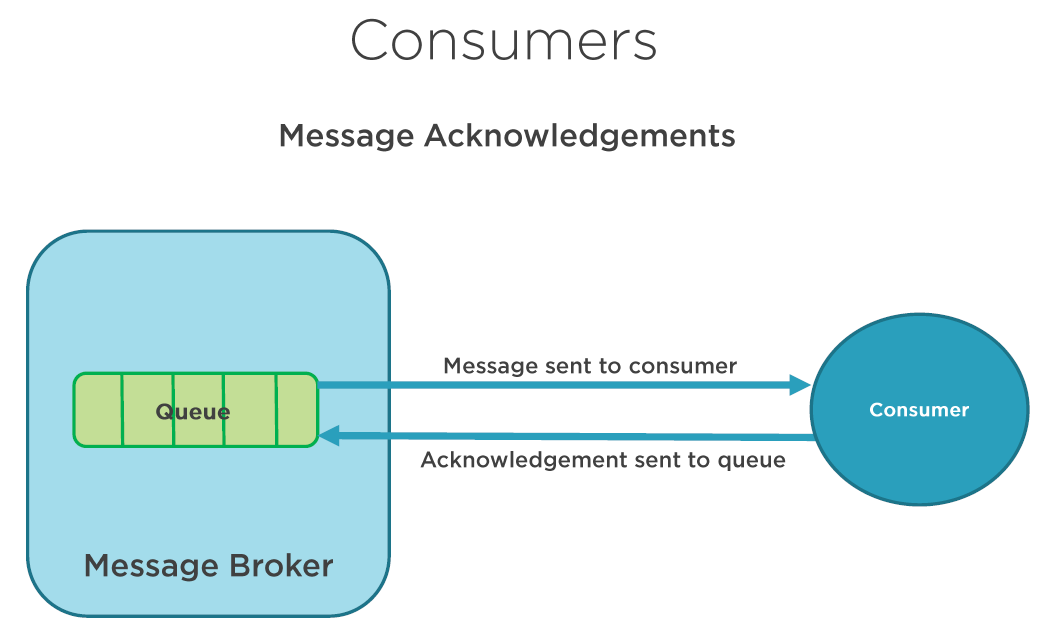
* When you want to specify how messaged are routed from EXCHANGES to QUEUES, the you need to define BINDINGS.
* BINDINGS may have optional routing key attribute that is used by some exchange types to route messages from exchange to queues.
* The purpose of the routing key is to select certain messages published to an exchange be routed to that bound queue. Ie. Routing-key acts like a filter
* If AMQP message cannot be routed to any queue because it does not have a valid binding from echange to that queue, then it will be dropped or returned to the publisher , depends on the message attributes publisher as set.



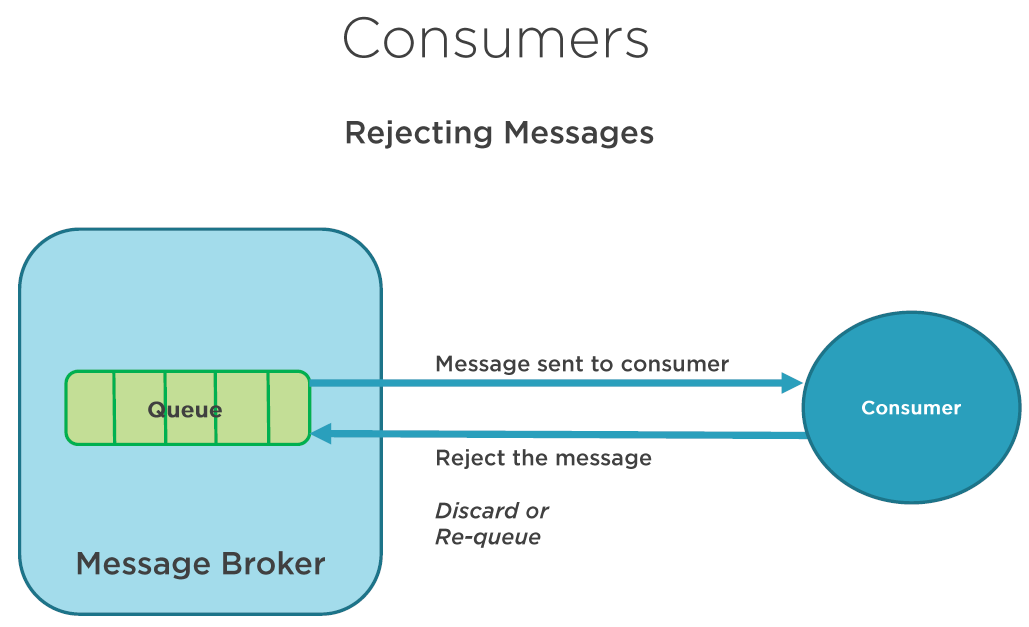
* Applications are registered as CONSUMERS/SUBSCRIBERS to the queue.



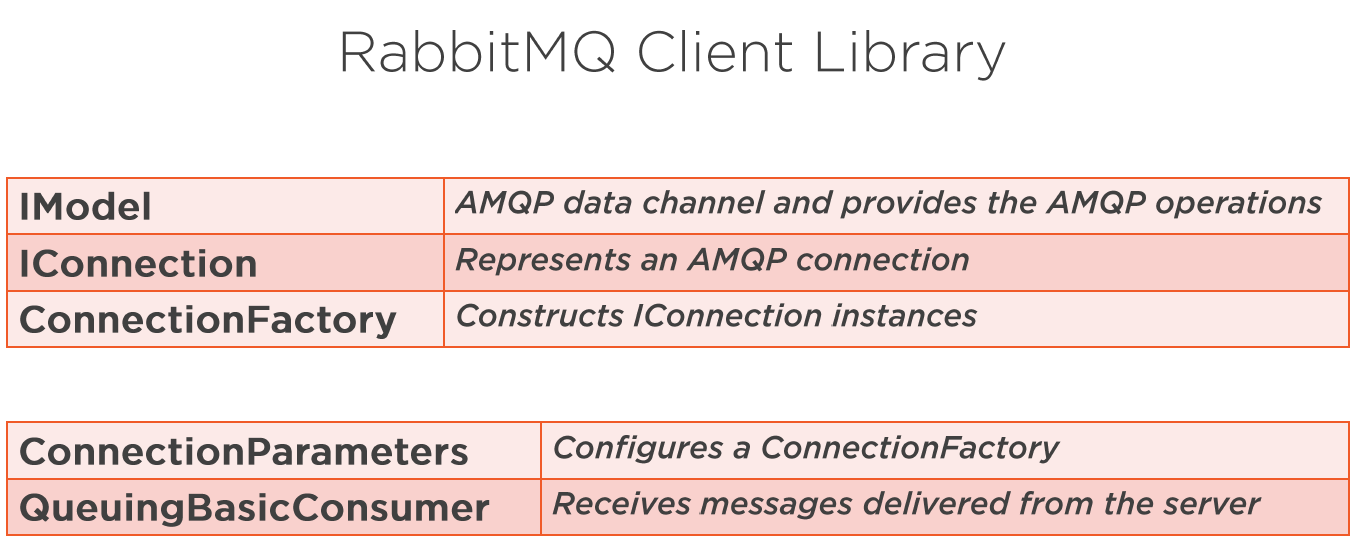
* There can be >1 applications registered as CONSUMERS/SUBSCRIBERS to a queue or sets of queues.
* This is a common usage scenario when you want to balance load of applications feeding from the queues in high volume scenario

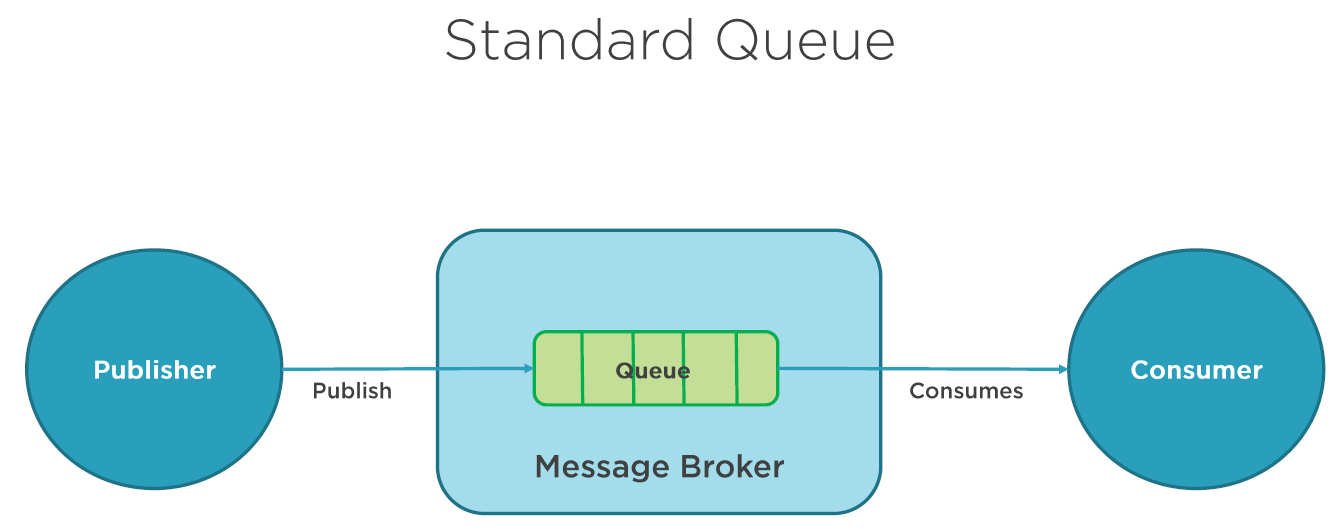


* When a consuming application acts on a message from a queue, it is possible that a problem could occur in that application, which means message is lost.
* Generally when an application acts on a message, that message is removed from the queue. But you may not want this to happen until you successfully process the message.
* AMQP protocol gives couple of ways of defining when a message is removed from the queue
* The message is removed once broker as sent message to the application or the message is removed once the application sends an acknowledgement back to the broker
* In the explicit acknowledgement, it is upto the application to decide when to remove message from that queue. This could be when you just received a message or it could be after you finished processing
* If the consuming application crashes before acknowldgement is sent , then message broker will try to redeliver the message to another consumer

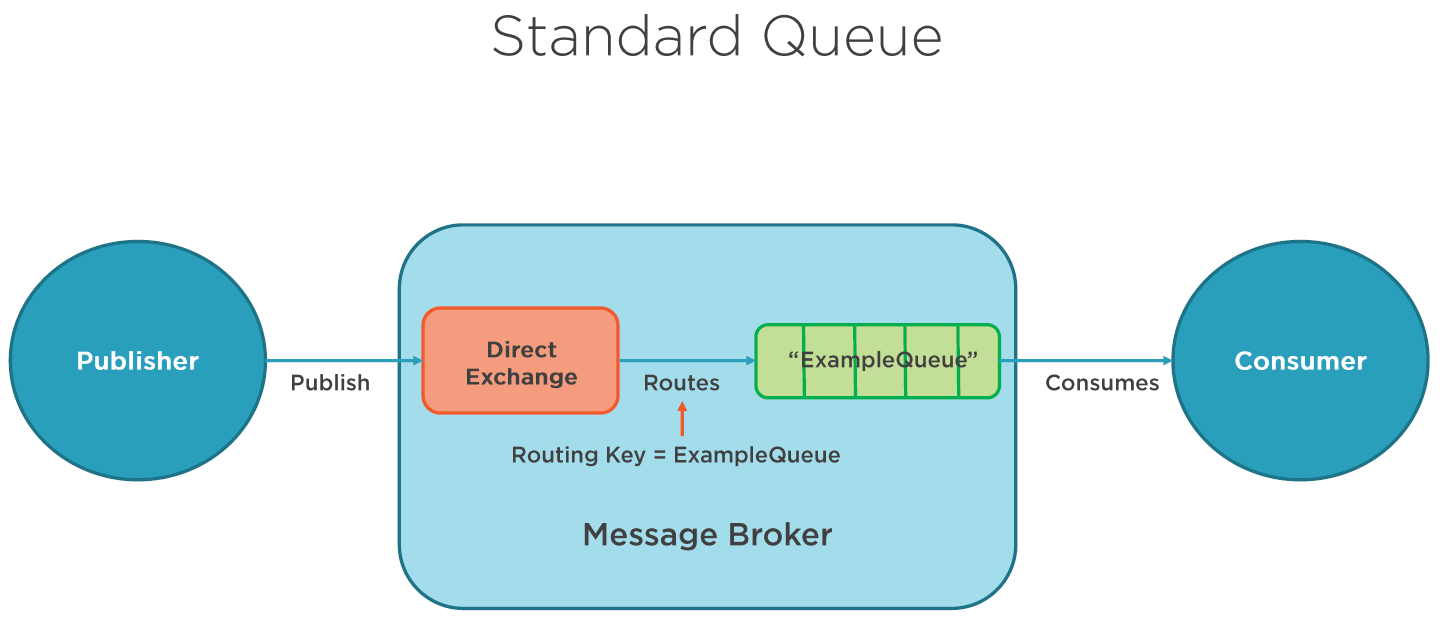


* When an application processes message, a processing may or may not succeed. If the processing fails for any reason (Eg: database timeout), then the consumer application can reject the message. When this happens, the consumer application can ask the broker to discard the message or requeue it.
* If any one consumer/application subscribes the queue, you need to make sure that you do not create an infinite message delivery loop by rejecting and requeuing the message from the same consumer over and over again

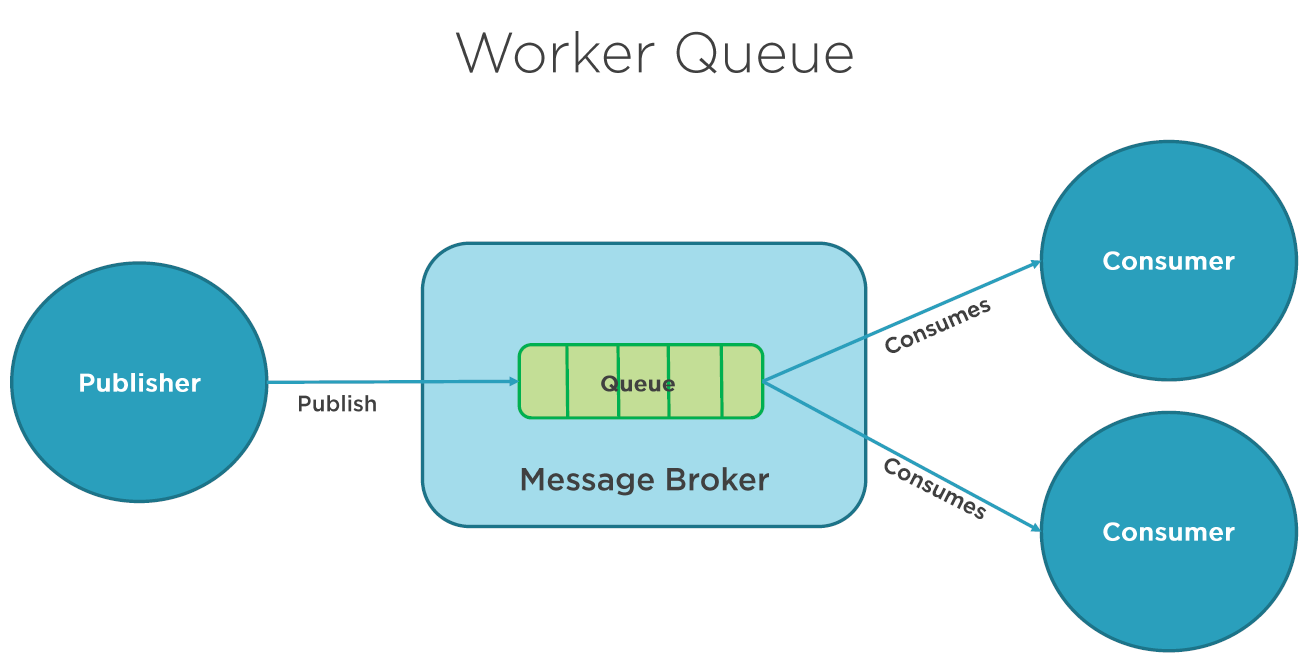


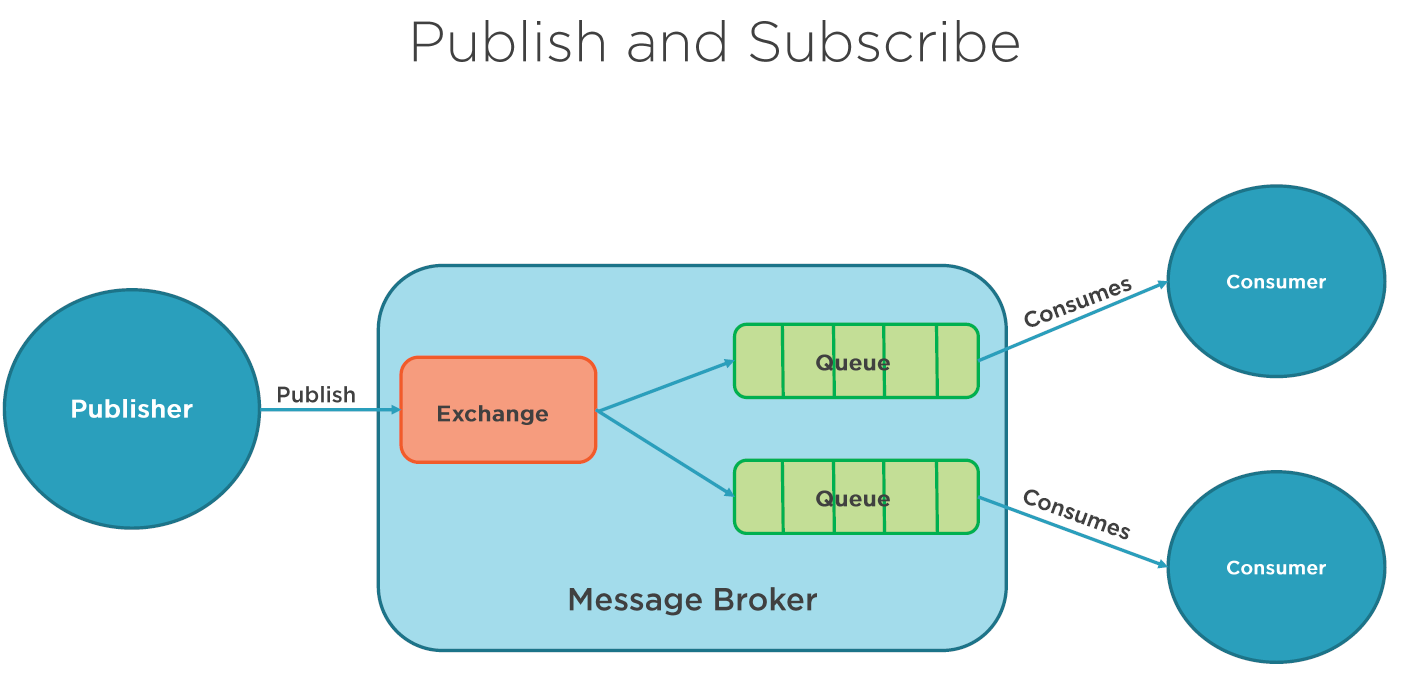


* It has one producer and one consumer.
* It shows Producer post messages directly to queue instead through an exchange, but there will a default exchange under the hood.

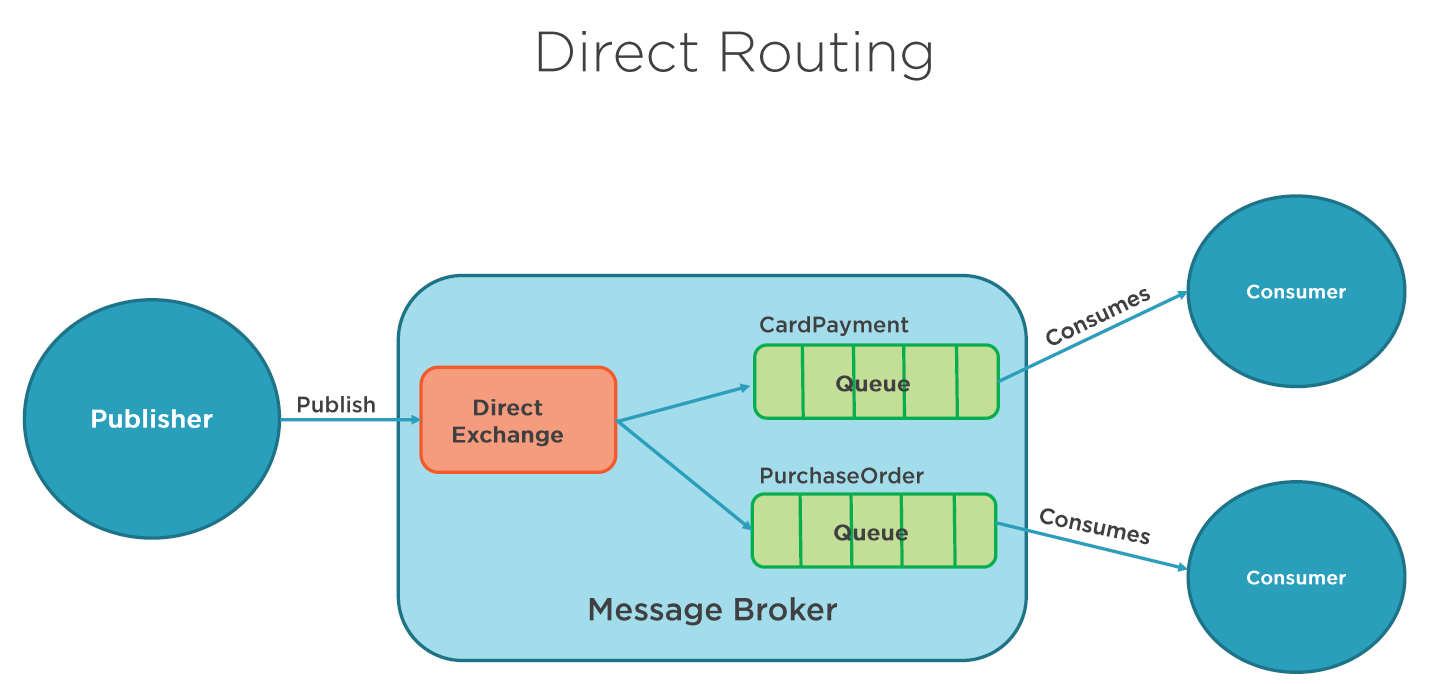


* When “ExampleQueue” is declared, RabbitMQ broker will bind it to default exchanege with “ExampleQueue” as routing-key



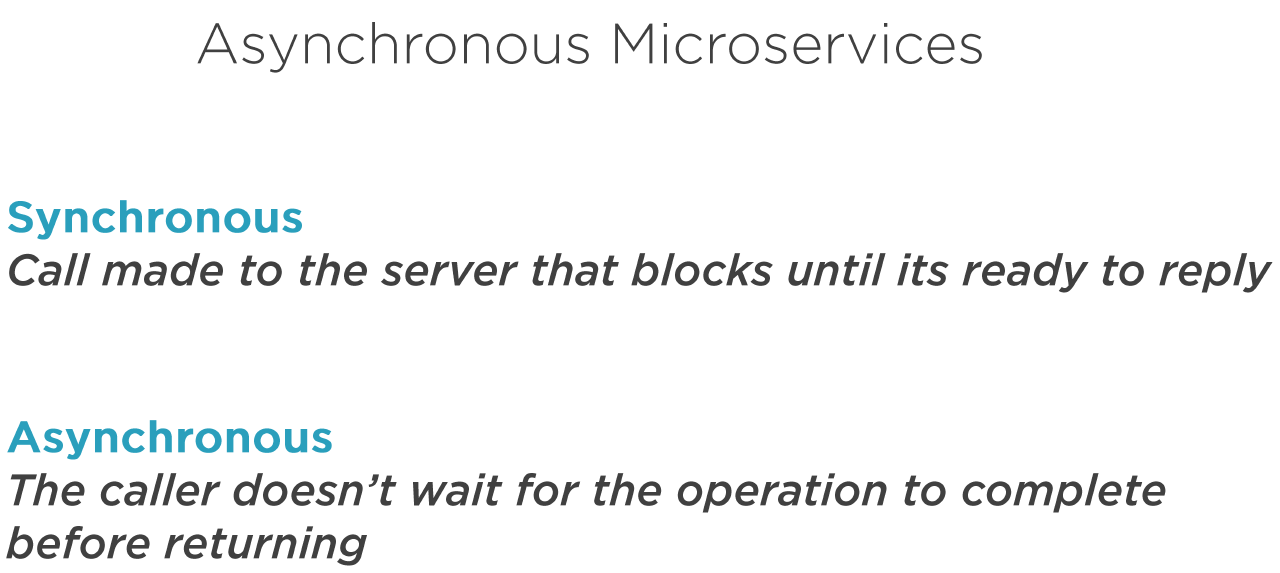


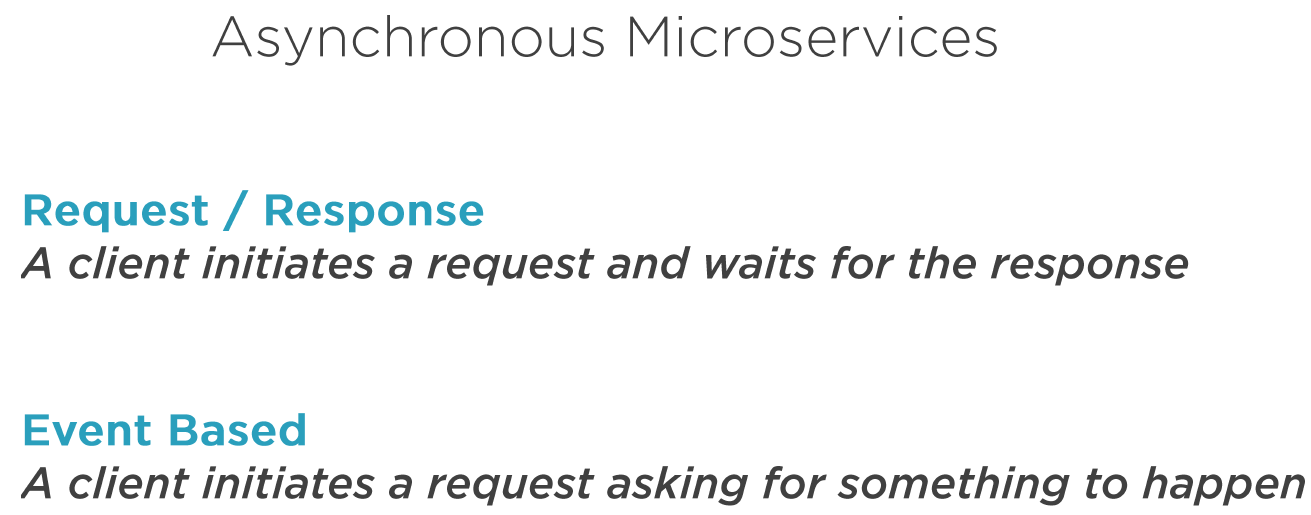
* In worker Queue consumers can share the load of messages from the queue.this means messages are distributed between multiple consumers.
* In Pub-Sub , the messages are sent from exchange to all consumers that are bound to the exchange.this means messages are not picked up by the multiple consumers , instead all consumers that are interested in receiving the messages.
* Unlike standard and worker queue, here we setup explicit exchange instead of default.
* FANOUT exchanges: it routes messages to all of the queues that are bound to it, and routing-key is ignored.

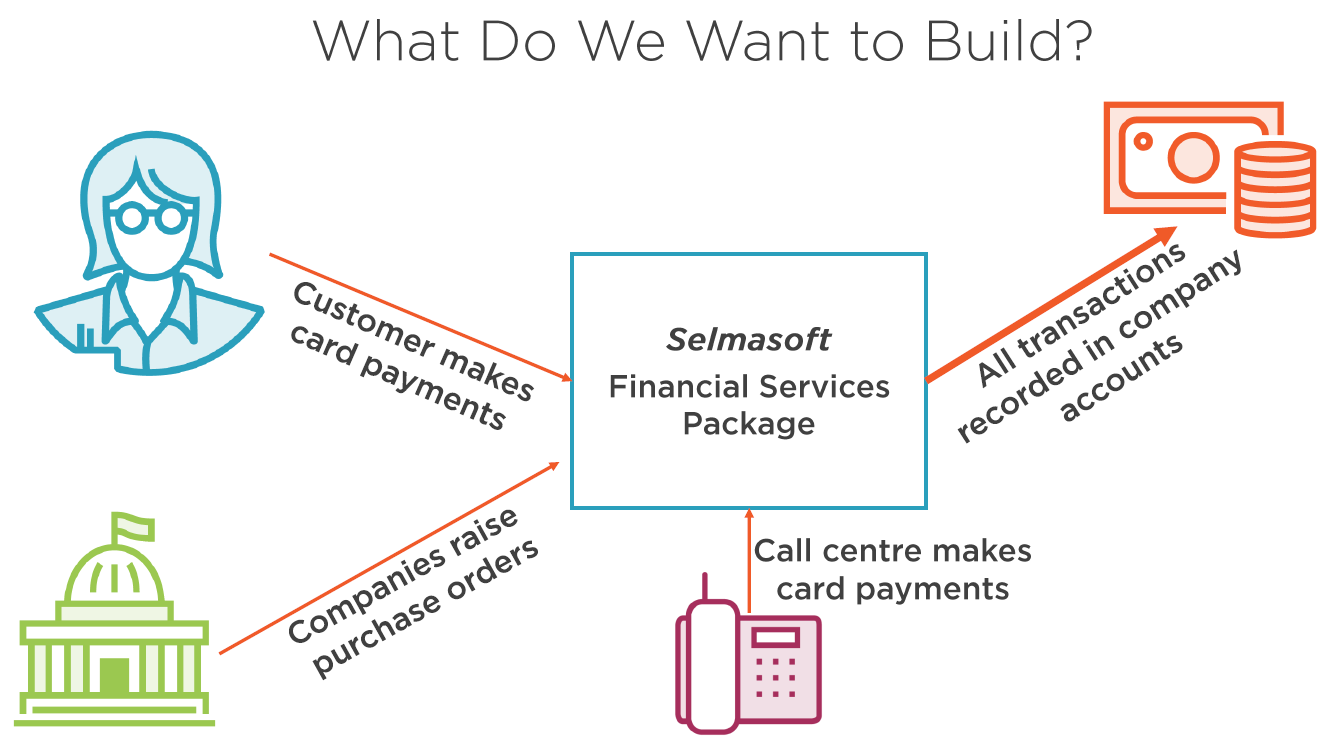


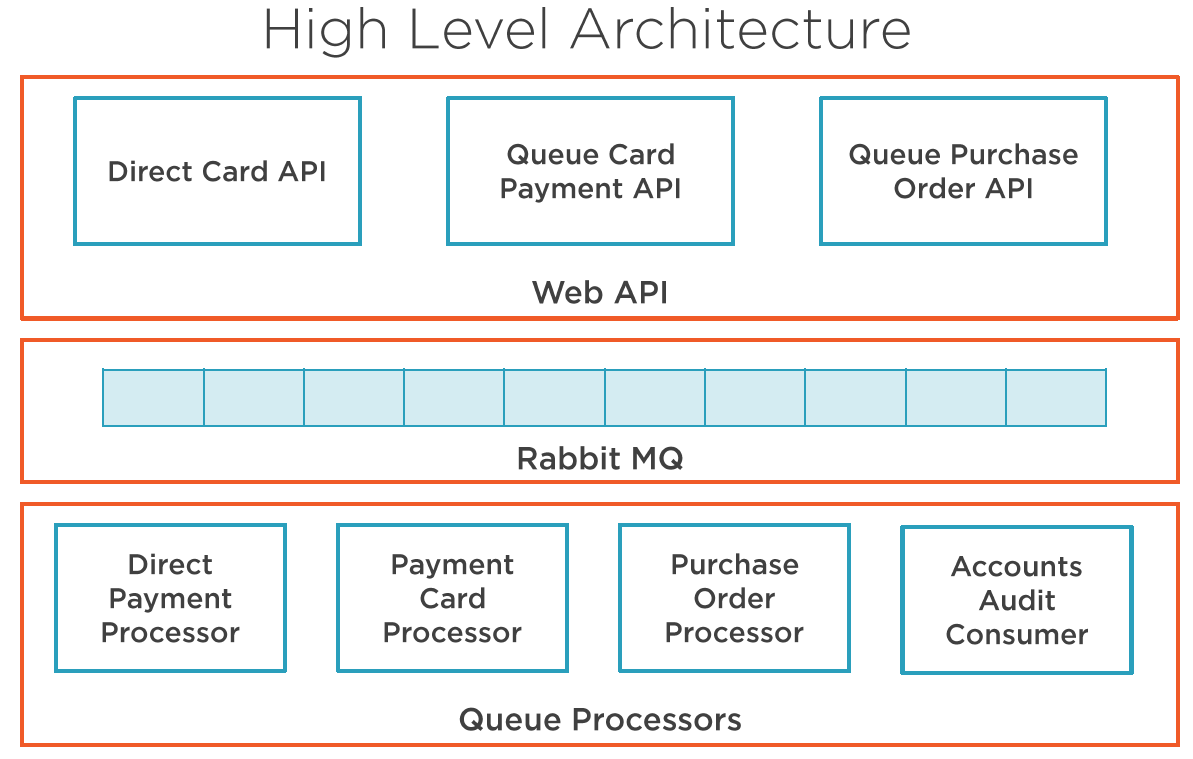
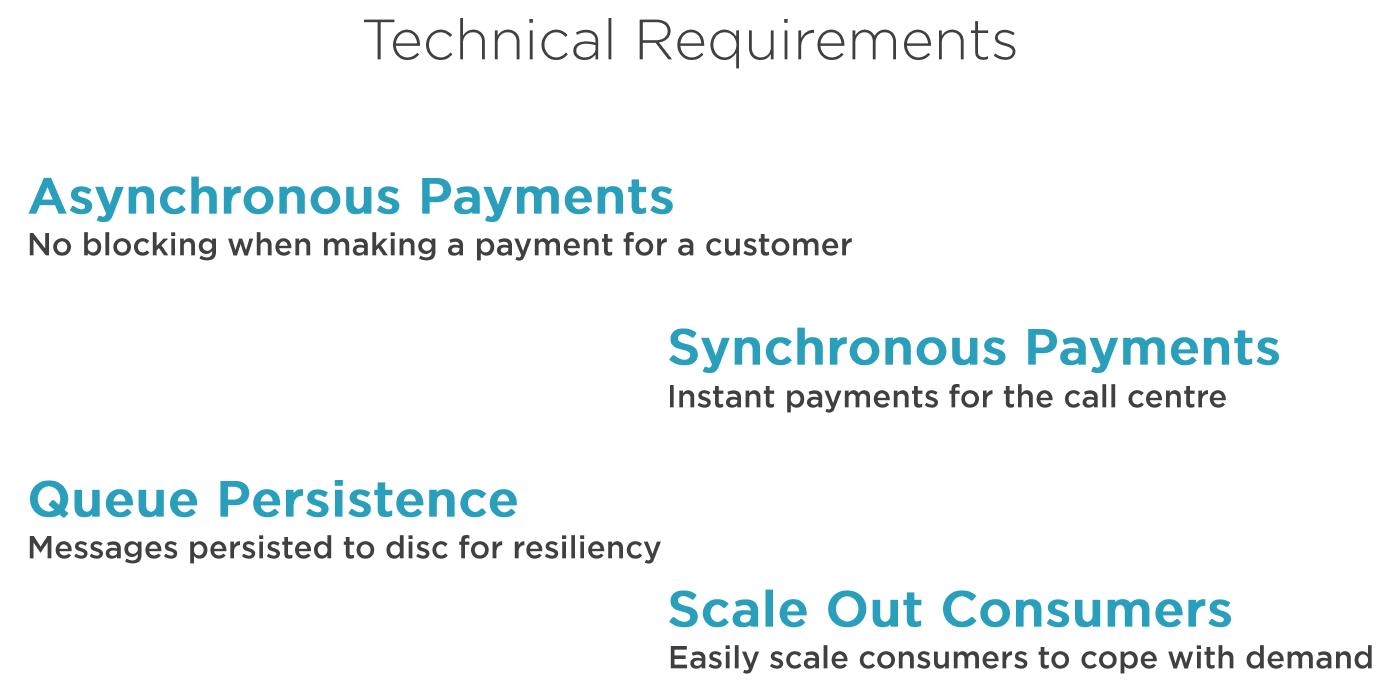
* It is similar to publish subscribe, but its fundamentally different.
* In FANOUT exchange , all of the messages are routed to each of the subscribers. There routing-key is ignored even if sent.
* Here in DIRECT exchange, the routing-key is used and direct messages to specific consumer.
* In this eg: producer posts 2 different types of messages 1.card payment, 2.purchase order

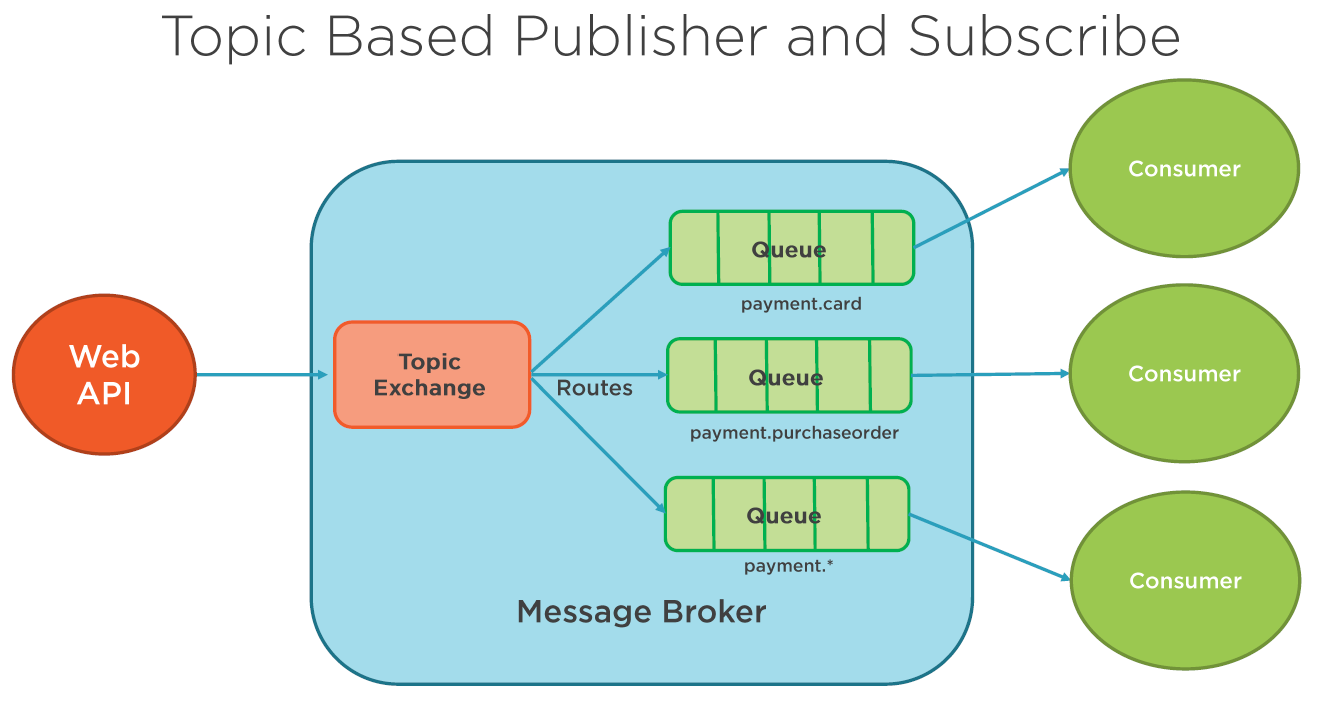
EG:











Routing-key here is list of works separated by dots.(logical feature). length Limit 255 bytes

\* > substitute of exactly 1 word, # > substitute of 0 or more words

