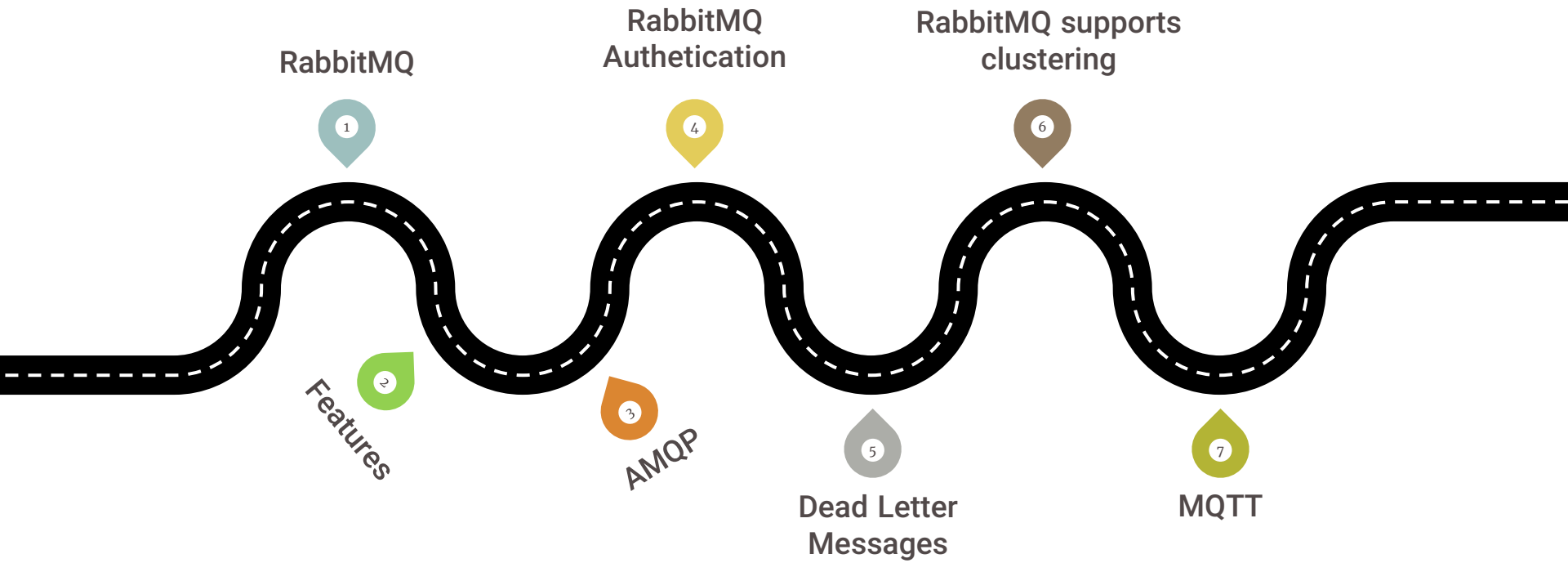
Roadmap


RabbitMQ
1
Features
2
AMQP
3
RabbitMQ
Authetication
4
Dead Letter
Messages
5
RabbitMQ supports
clustering
6
MQTT
7

RabbitMQ

این یک پروژه open source است که توسط VMware ایجاد شده و اکنون توسط Pivotal نگهداری می شود.

t is written in Erlang and designed for high-throughput, low-latency messaging.

- Messaging applications

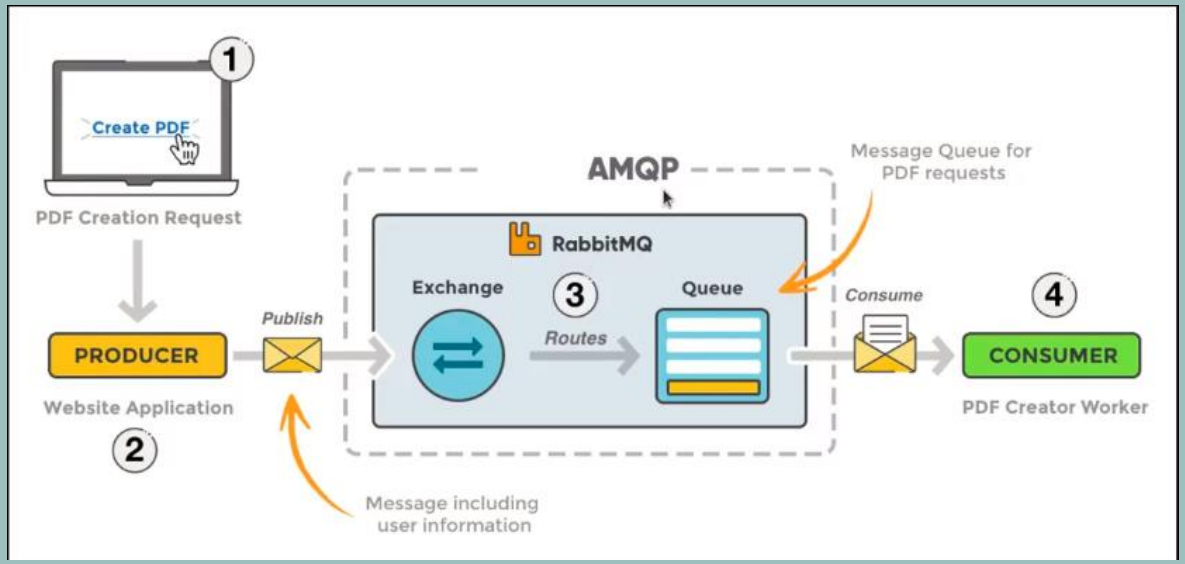
- Microservices architectures

- Event-driven architectures

RabbitMQرا می توان در برنامه های مختلفی استفاده کرد که در آن داده ها باید بین سیستم های مختلف رد و بدل شوند. برخی از نمونه ها عبارتند از:

RabbitMQ

یک نرم افزار برای انتقال پیام بین سیستم ها یا به عبارتی **message-broker software** که با استفاده از اون می تونیم بین سیستم های مختلف پیام ارسال کنیم و عملیات صف بندی به خوبی انجام بدیم.



Other Features

Multiplatform

Multiprotocol

Management UI

Widely supported

Flexible routing

High availability

Plugins

Tracing

Free and open source

Commercial options available

Push-based approach

از clustering و صف های آینه ای برای دسترسی بالا و تحمل خطا پشتیبانی می کند.

Push-based message/data consumption

Push-based message/data consumption یک مدل مصرف داده است که در آن broker منبع داده/پیام به طور فعال پیام‌ها را بدون درخواست صریح consumer ها به consumer ها ارسال می‌کند. Consumer ها فقط باید با subscribing علاقه نشان دهند.

مواردی که در این مدل خواهیم داشت

1 - تحویل پیام به صورت Real-time - معمولاً، داده ها به محض در دسترس بودن به consumer ها ارسال می شوند.

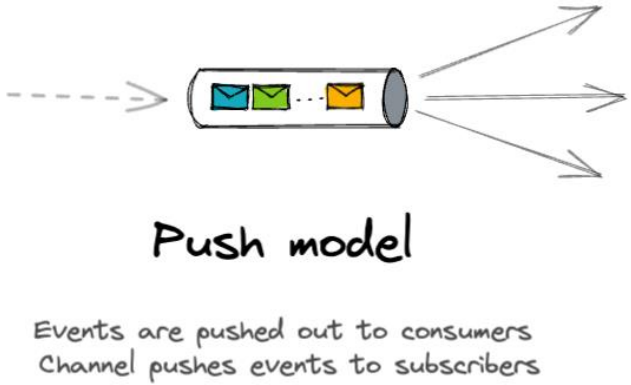
2- consumer ها ساده‌تر با مصرف منابع کمتر

3 - Consumers don't need to bother about latency or implementing an efficient polling loop as the broker handles it

از سوی دیگر چالش هایی نیز دارد

consumer ها باید بتوانند با سرعت تولید پیام ها هماهنگ باشند.
consumer ها باید برای دریافت پیام ها در دسترس باشند.
از آنجایی که broker سرعت انتقال پیام را کنترل می‌کند، زمانی که داده‌ها سریع‌تر از مصرف آن تولید می‌شوند، می‌تواند consumer را تحت تأثیر قرار دهد.

RabbitMQ: Push-based approach



RabbitMQ مبتنی بر push model و از طریق stops overwhelming که بر روی مصرف کننده تعریف شده است، consumer ها را تحت فشار قرار نمی دهد.
این می تواند برای پیام رسانی با تاخیر کم استفاده شود. هدف از push model توزیع پیام ها به صورت جداگانه و سریع است تا اطمینان حاصل شود که کار به طور یکنواخت موازی شده و پیام ها تقریباً به ترتیبی که در صف رسیده اند پردازش می شوند. با این حال، در مواردی که یک یا چند مصرف کننده «died» و دیگر پیامی را دریافت نمی کنند، این می تواند مشکلاتی ایجاد کند.

نکته مهمی که وجود داره اینه که RabbitMQ (Message broker)
که در ادامه توضیح خواهیم داد ، بصورت native از هر دو پترن
pub/sub -1
Messaging Queue -2
پشتیبانی میکنه. و در حالت Pub/Sub از هر دو حالت durable و
ephemeral پشتیبانی میکنه. این consumer هست که می‌تونه
تصمیم بگیره که کدوم حالت انتخاب بکنه.

Protocols in RabbitMQ

AMQP 0-9-1 (Advanced Message Queuing Protocol v0.9.1)

MQTT (Message Queuing Telemetry Transport)

STOMP (Simple Text Oriented Message Protocol)

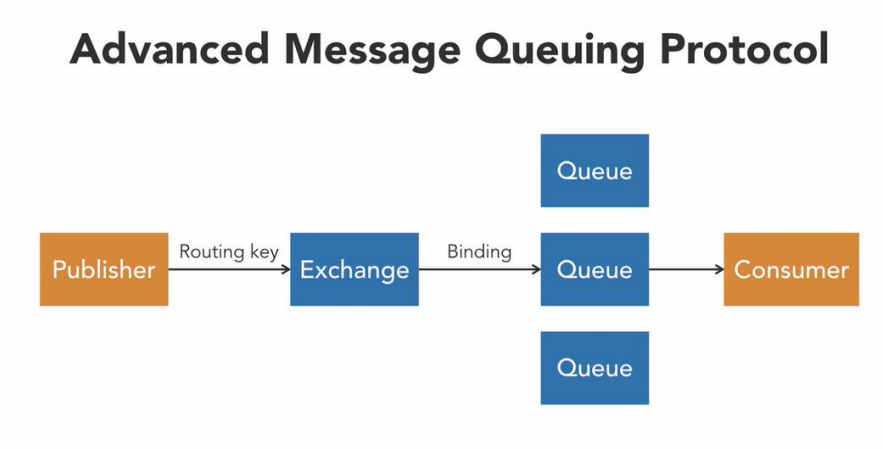
AMQP 1.0 (Advanced Message Queuing Protocol v1)

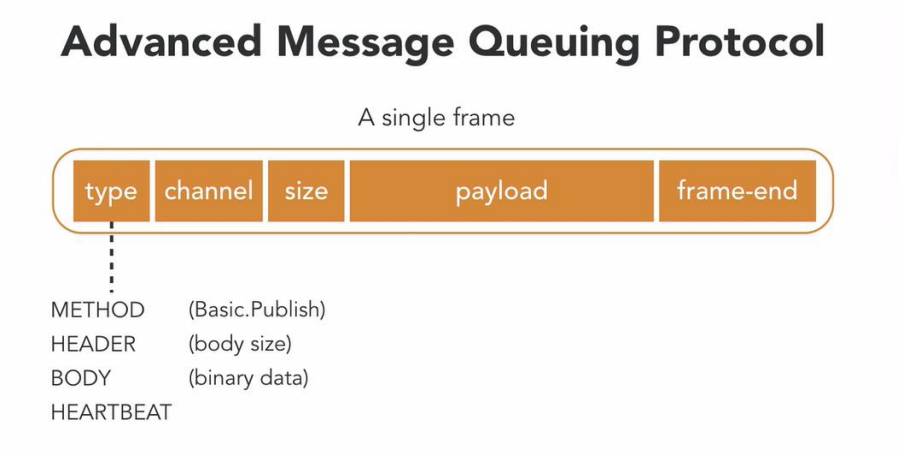
AMQP

Advanced Message Queuing Protocol

یه پروتکل Messaging یا پیام رسانی است که اجازه میده نرم افزارها به عنوان یه client به سیستم های messaging broker وصل بشن و با اون **تعامل** داشته باشن.

queue و **exchange** و **binding** همگی جز ماهیت های تشکیل دهنده پروتکل **AMQP** هستن که بهشون **AMQP Entities** گفته میشه.





پروتکل AMQP ساختار چارچوب های خود را تعریف می کند. پیام ها بسته های داده ای هستند که از طریق شبکه ارسال می شوند. در بایت اول، پروتکل AMQP نوعی پیام را ذخیره می کند(مشخص می کند). این می تواندMETHOD , HEADER, BODY, HEARTBEAT . در بایت بعدی، channel ذخیره می شود. این کانال یک ارتباط مجازی(virtual connection) است بعد از آن نوبت به اندازه (size) می رسد. این بایت شامل عددی است که نشان دهنده اندازه بار فریم است. سپس، payload بسته به نوع قابی که ارسال می کنیم متفاوت خواهد بود تعداد بایت های استفاده شده در اینجا برابر با اندازه ای است که در بلوک قبلی قرار داده شده بود. Payload method حاوی مقداری خواهد بود که نشان دهنده عملی است که می خواهیم انجام دهیم در مورد ارسال یک پیام می تواند حاوی چیزی مانند Basic.Publish باشد. در نهایت، یک بایت انتهای کادر (frame-end) را نشان می دهد.

بنابریا نهنگامی که ما یک پیام واحد را بر روی پروتکل AMQP ارسال می کنیم، بسیاری از این فریم ها را انتقال خواهیم داد. یک فریمMETHOD ، یک فریمHEADER ، و تعداد زیادی فریمBODY .

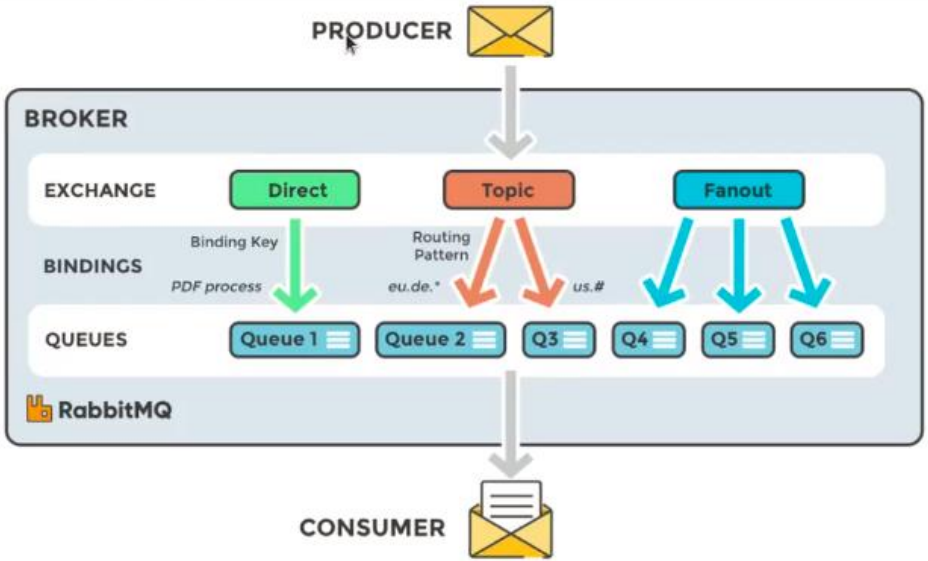
مدل کلی پروتکل AMQP

messageهای ارسال شده از نرم افزار فرستنده (**publisher**) پیام ها مستقیما به صف **publish** نمی شوند و **ابتدا** توسط exchange دریافت می شوند. بعد **exchange** یه **کپی** از پیامی که دریافت کرده رو بر اساس یسری قوانین و المان برای **queue** میفرسته (اگه بخوام بهتر بگم پیام ها رو هدایت میکنه برای queueکه به این هدایت کردن در اصطلاح فنی **route** میگیم).

نکته ای که باید گفت این است که صف ها در RabbitMQ از روش **FIFO (First In, First Out)**، پیروی می کنند.

First In, First Out یا به اختصار **FIFO** مکانیسمی در مدیریت داده‌ها بر اساس ترتیب زمانی است که از آن طریق ریکوئست‌های مرتبط با ساختار داده های Queue و Stackمدیریت می‌شوند بدین صورت که **اولین** درخواستی ارسالی **اول** از همه هندل خواهد شد (نقطهٔ مقابل FIFOرویکرد Last In, First Outیا به اختصار **LIFO**است بدین شکل که جدیدترین درخواست‌ها اول از همه هندل خواهند شد.)
برای درک بهتر این موضوع، می‌توان قرار گرفتن خودروها پشت چراغ راهنمایی را مد نظر قرار داد بدین صورت که اولین خودرویی که پشت چراغ قرار می‌گیرد همواره پس از سبز شدن چراغ اول از همه از صف خارج خواهد شد و شروع به حرکت خواهد کرد (در چنین موقعیتی سَر صف اصطلاحاً Head و ته صف Tailنامیده می‌شود.)

routing key and binding



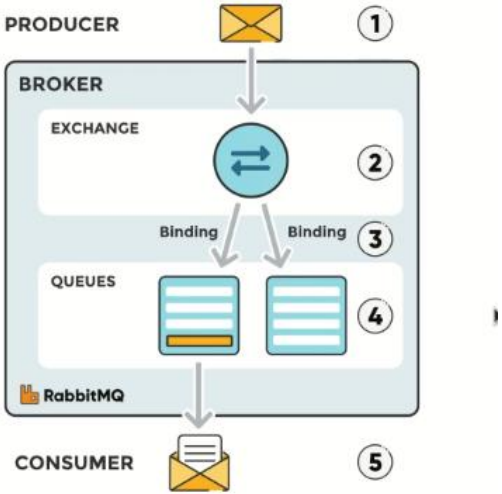
routing key یکی از attribute های header است که توسط **فرستنده** پیام مشخص میشه. exchange برای هدایت پیام به queue ازش استفاده میکنه. **تاثیر** routing key در هدایت پیام ها به **نوع** **exchange** بستگی داره اگر:

Exchange از نوع **fanout** باشه اصلا این attribute **اهمیتی ندارد** اما زمانی که exchange از نوع **direct** یا **Topic** باشد این attribute خیلی مهم است برای هدایت پیام به queue چون مشخص می کند به **چه** **queue** پیام منتقل شود(به عنوان آدرس مقصد پیام در نظر بگیرید).در واقع **binding** با استفاده از آن با توجه به نوع exchange پیام ها رو به queue موردنظر منتقل می کند

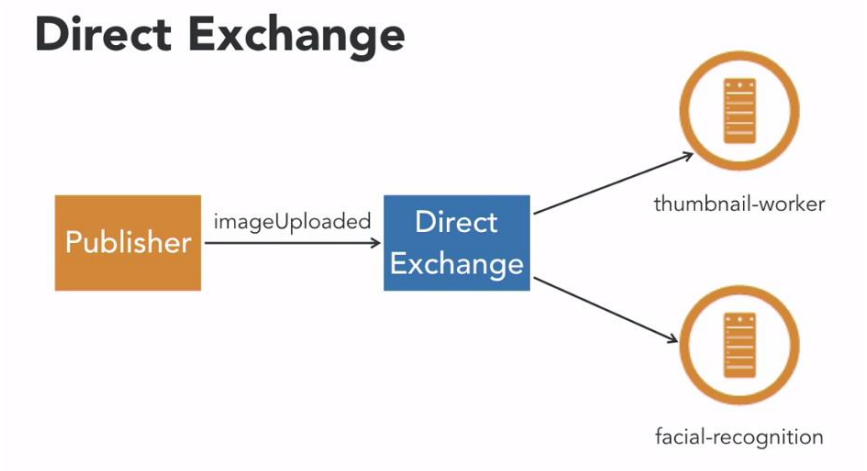
Exchangeها و انواع آن ها

Exchangeیک درگاه ورود به RabbitMQبرای پیام هاست. وظیف آنها دریافت پیام از Producer ها و Pushکردن انها روی Queue در RabbitMQبسته به **نوع** exchange، پیام ها در صف های متفاوتی مسیردهی می شوند. Exchange Type یک صف احتیاج دارد که به حداقل یک Exchangeمقید شده باشد تا بتواند پیام‌ها را دریافت کند.

چهار نوع وجود دارد:



direct exchange : پیام به **Queue** هایی هدایت می شود که binding key آن ها دقیقاً با **routing key** پیام مطابقت دارد.(routing key ویژگی پیام است که توسط Producer به پیام اضافه می شود.) به عنوان مثال، اگر صفی که binding key آن برابر با pdfprocess است به exchange متصل باشد، پیامی که routing key آن برابر با pdfprocess باشد، به آن **Queue** هدایت می شود.



An example here is a system where a user can upload an image and you can have a message sent out with a routing key image uploaded for example. The direct exchange might pass the message to a queue for an application that will create a thumbnail. And another queue might be for an application that does some facial recognition.

Sample code python
Sender=publisher
Exchange=direct

sender.py 1 ● receiver.py ● sender.py 2 ●

2 > sender.py > ...

```python
import pika

# todo connect to server rabbitmq
connection = pika.BlockingConnection(pika.ConnectionParameters(host='localhost'))
# create channel
ch1 = connection.channel()

# Declare queue, created if needed, This Method created or checks a queue
# parameter => durable => durable queues are persisted to disk and thus Survive broker reastart.
# Durabilty of a queue dose not make messages that are routed to that queue durable.
# parameter => auto_delete => Delete after consumer cancels or disconnects
# paramets => queue => The queue name should unique
ch1.queue_declare(queue='first', durable=True)

message = 'the test message'
ch1.basic_publish(exchange='',
                  routing_key='first',
                  body=message,
                  properties=pika.BasicProperties(delivery_mode=2, headers={'name': 'ali'}))
print('message send')
# close connection
connection.close()
```

ایجاد queue ها
داخل channel

exchange=‘ ‘
یعنی از نوع direct

Auto-delete: پارامتری دیگری از تابع queue_declare که اگر مقدار آن برابر با True باشد محض اینکه ارتباط آخرین jlqueue با exchange قطع میشه، exchange حذف میشه

برای اینکه بخواهیم زمانی که سرور ما ریستارت میشه channel هامون از بین نره durable=True قرار میدهیم این اتفاق باعث می شود که اطلاعات کانال ها علاوه بر رم در داخل هارد نگه داشته شود این مقدار به صورت default برابر با false

برای اینکه بخواهیم زمانی که سرور ما ریستارت میشه پیغام هامون حفظ بشه و از بین نره این property باید ست کنیم

(در basic publish که اگر مقدار یک را قرار بدهیم روی هارد(disk) ذخیره نمی کنه پیام ها رو و اگر 2 قرار دهیم روی هارد(disk) ذخیره می کنه پیام ها رو)

Queue and Exchange Configuration

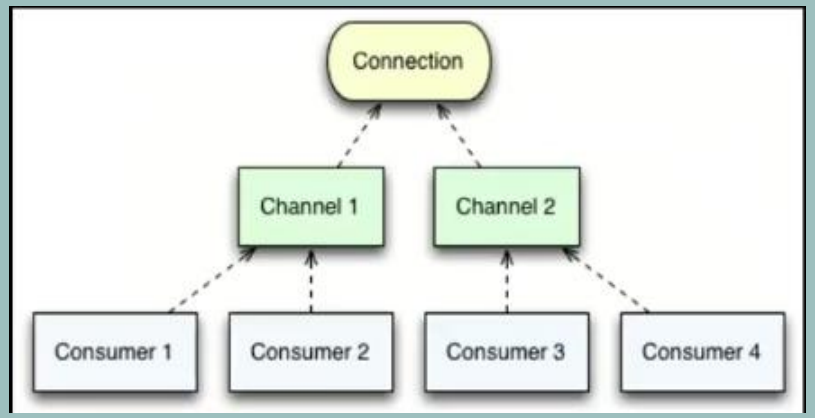
1 –Durability = True

2 –Durability = False

3 –Auto–delete

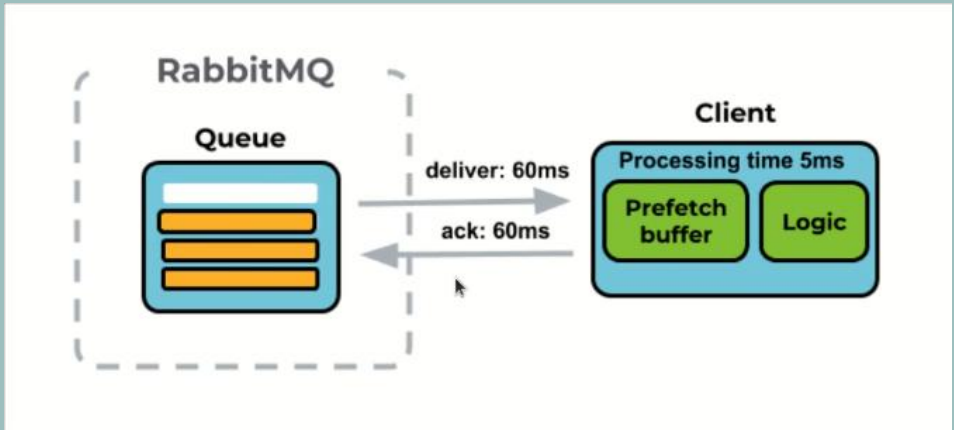
اجازه دهید به دو گزینه تنظیمات اشاره کنم که Exchange و صف ها در آن ها مشترک هستند.. یک **Queue** یا **Exchange** (durable) از راه اندازی مجدد RabbitMQ جان سالم به در خواهد برد اما در مورد **Queue** های گذرا این اتفاق نخواهد افتاد. به خاطر داشته باشید که این بدان معنا نیست که پیام‌های تحویل‌نشده باقی می‌مانند، مگر اینکه تداوم(persistence) را تنظیم کنید، RabbitMQ پیام های شما را در حافظه نگه می دارد. این بدان معناست که با راه اندازی مجدد سرویس، (پیام ها) از دست خواهند رفت.

به سراغ channel برویم: ممکنه که برنامه ما نیاز به چندین اتصال TCPبه سرور داشته باشد. ایجاد کردن چندباره connectionبه سرور هدر دادن منابع سخت افزاری است. channelها یک رابط بین سرور و consumerها هستند. چندین consumerبه یک channelوصل شده و channelفقط یکبار به سرور وصل میشود. consumerها با اتصال به channelبا سرور کار میکنند.



ما دو جور acknowledge :

Auto acknowledge: به محض دریافت پیام توسط consumer پیام رو از داخل صف حذف می کنه چه تسک انجام بشود یا انجام نشود به دلایل مختلف اما برعکس manual acknowledge در آخر که تسک انجام شده است یه ack ارسال میکنه به queue و پیام را از داخل صف حذف می کند.



Sample code python
receiver=consumer
Exchange=direct

receiver.py 2 ● | receiver.py 1

2 > receiver.py > callback

```python
import pika
import time

# create connection for connect to server rabbitmq
connection = pika.BlockingConnection(pika.ConnectionParameters(host='localhost'))
# create channel
ch2 = connection.channel()

# Declare queue, created if needed, This Method created or checks a queue
ch2.queue_declare(queue='first', durable=True)


def callback(ch, method, properties, body):
    print(f'Received {body}')
    time.sleep(9)
    print(properties.headers)
    print('Done')
    #manualacknowledge
    ch.basic_ack(delivery_tag=method.delivery_tag)


ch2.basic_qos(prefetch_count=1)
# consume to the broker and binds messages for the consumer_tag to the consume callback
ch2.basic_consume(queue='hello', on_message_callback=callback, )

print('waiting message')
# start consuming
ch2.start_consuming()
```

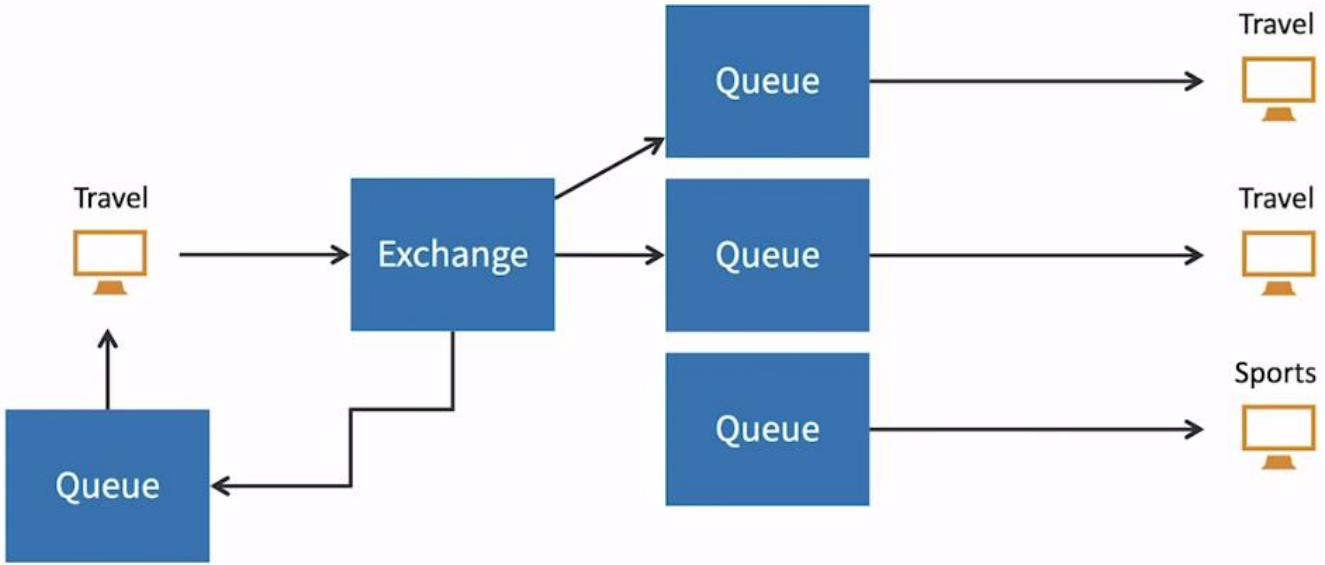
یکی از آرگومان های callback function <---
method
Redelivered که اعلام می کنه یک تسک قبلا فرستاده شده به یک cunsumer یا نه

Delivery_tag یک مقدار مثبت برای شناسایی پیام هاست.
یک عدد که consumer به صف بر می گرداند که برای هر پیام منحصر به فرد و صف با توجه به این عدد پیام مربوطه را حذف می کند

manual acknowledge

تابع که وصل میشه به صف تا کاری بکند

Chat Rooms

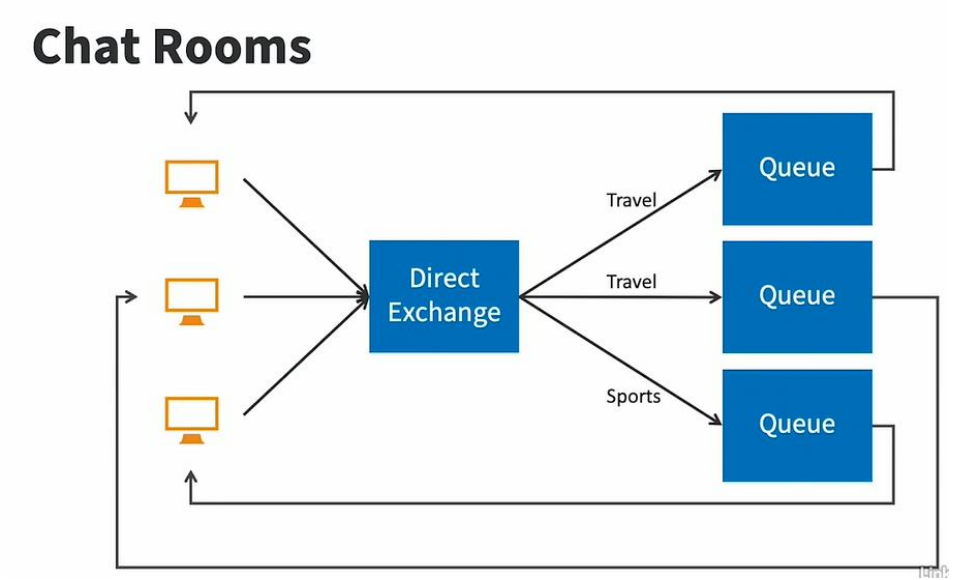

Travel
Exchange
Queue
Queue
Queue
Queue
Travel
Travel
Sports

Chat Rooms

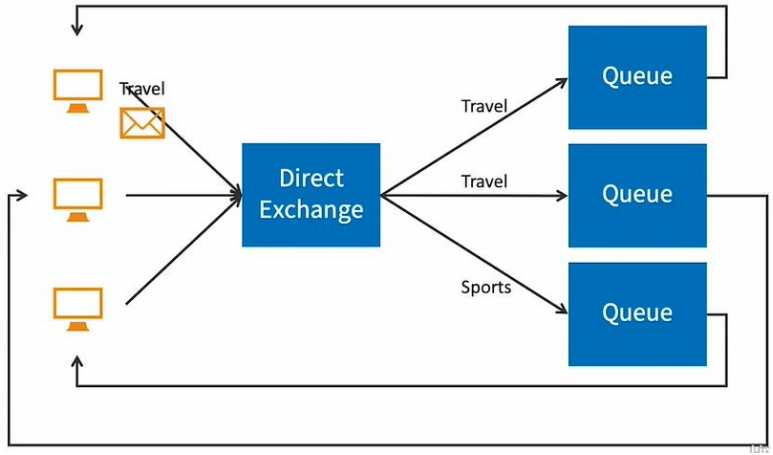
No fanout → Direct exchange

Keep chatroom selection simple

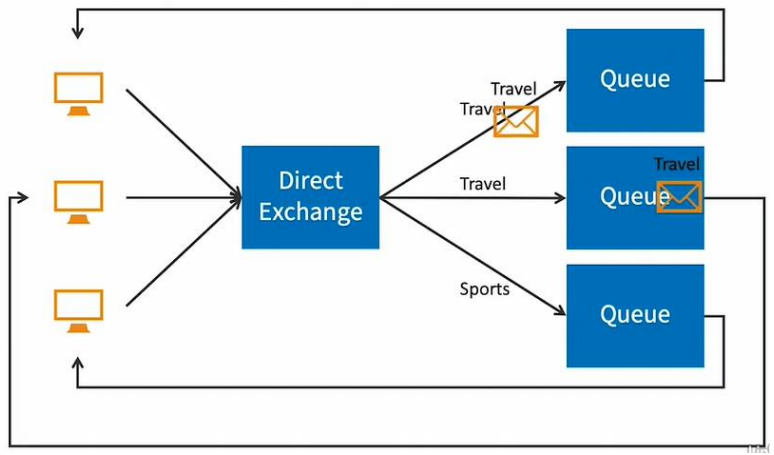
Route to correct queues



Chat Rooms

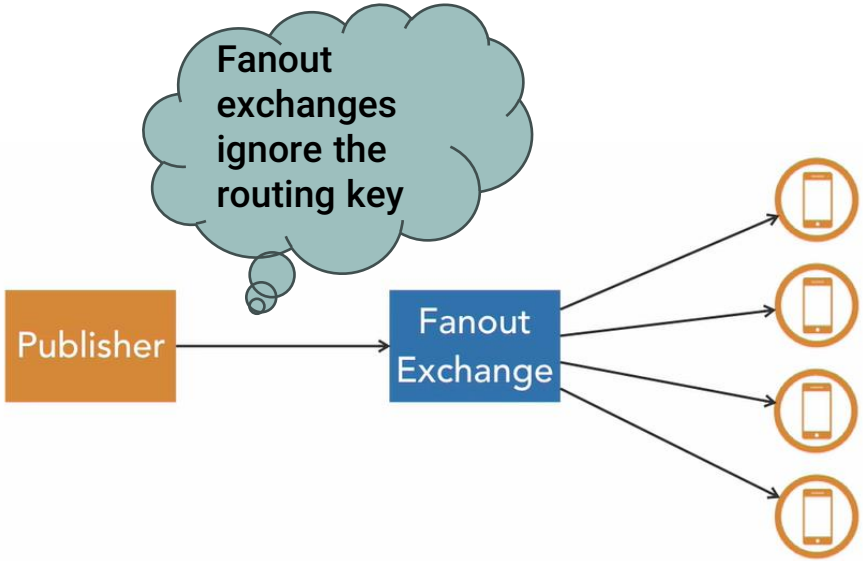


Chat Rooms



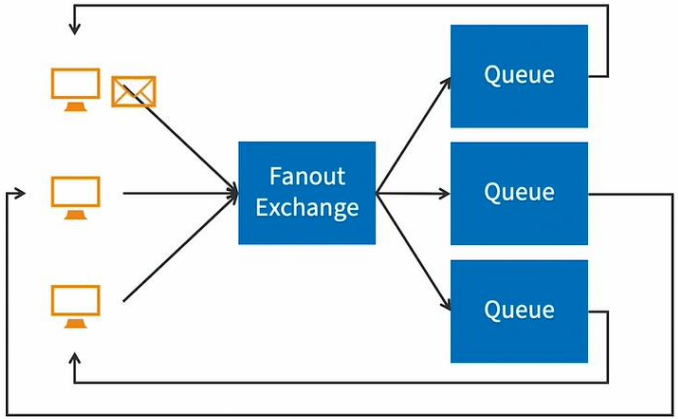
Fanout exchange:

پیام های دریافتی را به تمامی صف های متصل، منتقل می کند و **توجه ای به routing key ها ندارند.**

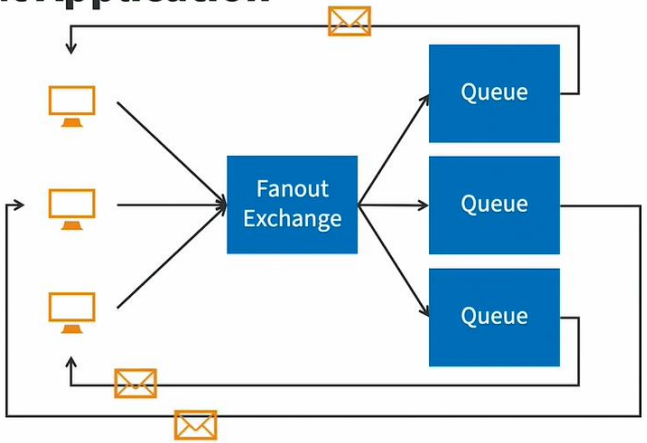


A good example is a service announcement that needs to be sent to all connected to mobile applications. Other examples include **games that need to send out leaderboard updates or distributed systems that have to notify components of configuration changes**.

Chat Application



Chat Application



Sample code python
Sender=publisher
Exchange=fanout

sender.py | receiver.py

3 > sender.py > ...

```python
import pika

 # todo connect to server rabbitmq
connection = pika.BlockingConnection(pika.ConnectionParameters(host='localhost'))

# create channel
ch = connection.channel()

# exchange fanout
ch.exchange_declare(exchange='logs', exchange_type='fanout')


# routing key = '' becuase exchange type's fanout
ch.basic_publish(exchange='logs', routing_key='', body="exchange fanout")
print('message send')
# close connection
connection.close()

```

Sample code python
receiver=consumer
Exchange=fanout

sender.py | receiver.py

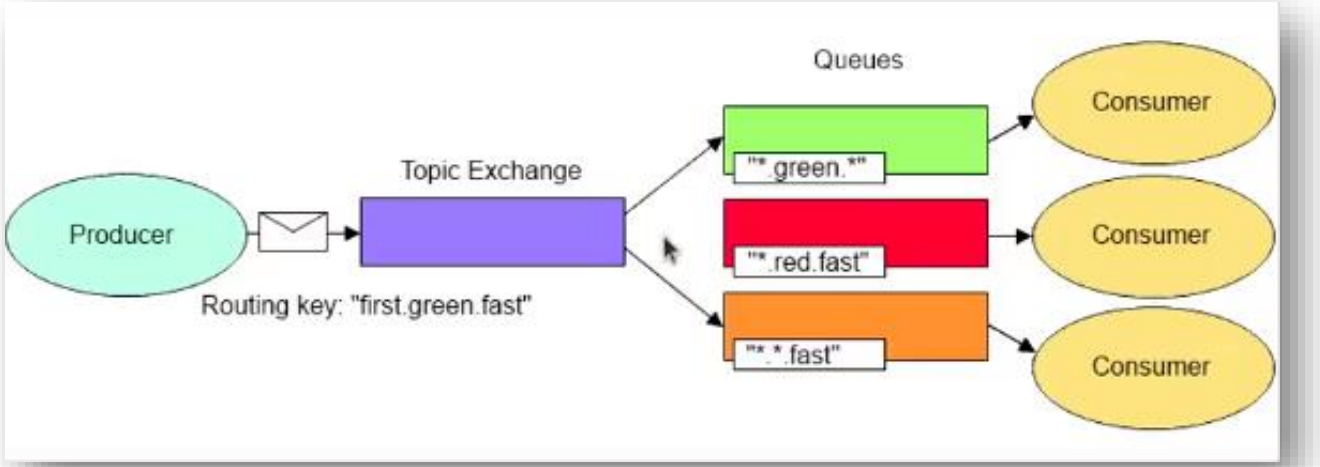
3 > receiver.py > ...

```python
import pika

# todo connect to server rabbitmq
connection = pika.BlockingConnection(pika.ConnectionParameters(host='localhost'))

#create channel
ch = connection.channel()
#create exchange fanout
ch.exchange_declare(exchange='logs', exchange_type='fanout')

# exclusive default False => once the consumer connection is closed, the queue should be deleted.
# name queur generated random by rabbitmq
result = ch.queue_declare(queue='', exclusive=True)

# binding
ch.queue_bind(exchange='logs', queue=result.method.queue)


def callback(ch, method, properties, body):
    print(f'Received {body}')


print('waiting')
ch.basic_consume(queue=result.method.queue, on_message_callback=callback, auto_ack=True)
# start consuming
ch.start_consuming()
```

برابر True باشد به یک exchange یا یک صف گفته زمانی که مقدار exclusive ایم که اگر دیگر هیچ چیزی به آن متصل نبود، خود به خود حذف شود. برای مثال، زمانی که آخرین مصرف کننده قطع می شود، یک صف می تواند به طور خودکار حذف شود.

Topic Exchange

نوعی از exchange است که routing key پیام های ارسالی رو با الگوی مشخص شده توسط queueها مطابقت میده (الگوی binding) ، اگه routing key پیام با الگوی اعلام شده توسط queueمطابقت داشت برای اون queueیه کپی از پیام ارسال شده، ارسال خواهد شد.

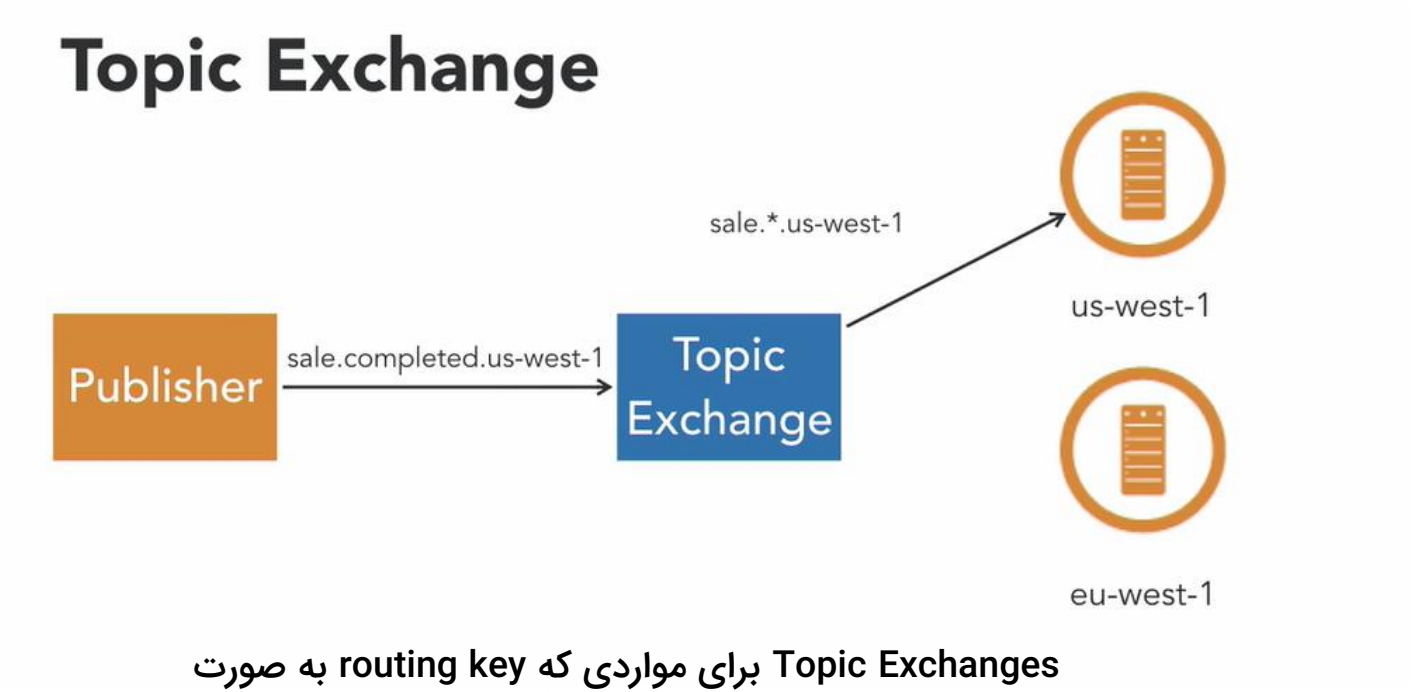


example

Background tasks

Logging events In a certain category

Messaging categories



Topic Exchanges برای مواردی که routing key به صورت پویا ساخته می شود و مصرف کنندگان (consumers) می خواهند پیام های خاصی را فیلتر کنند، ایده آل هستند.

routing keyمیتونه شامل لیست ترتیبی از کلمات باشه. هر کلمه توی این لیست با علامت ' . ' (بخونید: دات یا نقطه) از هم جدا میشه. مثلا اگه میخوایم مشخص کنیم که پیام ارسالی از نوع مقاله های کاربیت هستش میتونیم از این ساختار به عنوان routing keyپیام استفاده کنیم:

مقاله eskazemi درباره زبان برنامه نویسی پایتون	eskazemi.article.lang.python
خبر های eskazemi درباره زبان برنامه نویسی پایتون	eskazemi.news.lang.python
خبر های eskazemi درباره زبان message broker rabbitmq	eskazemi.news.mq.rabbitmq

برای مشخص کردن الگوی مورد نیاز در routing patternمیتونه از علامت های زیر استفاده کنیم:
علامت # توی routing patternنشون میده که از اون کلمه به بعد برای انطباق با الگو مهم نیست و هرچیزی بعدش باشه یا نباشه قابل قبوله:

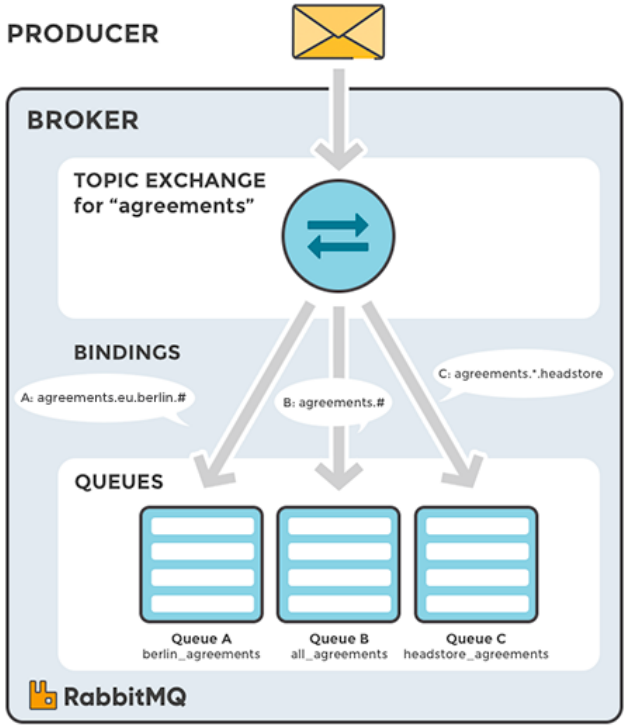
تمام خبر های کاربیت	Karbit.news.#

علامت * توی routing patternنشون میده که هر مقداری بجای علامت * قابل قبوله (یادتون باشه index قرارگیری * در مقدار routing keyبرای الگو مهمه):

تمام خبر های کاربیت که مربوط به پیام رسان هاست

Karbit.news.mq.*

الگوی بالا رو به این شکل نیز می توان خوند تمام پیام هایی که کلمه اول اونها karbit است و کلمه دوم اونها news است وکلمه چهارم اون هر چیزی که بود مهم نیست



Sample Code Python
Sender=publisher
Exchange=Topic

sender.py

topic > sender.py > ...

```python
import pika

# todo connect to server rabbitmq
connection = pika.BlockingConnection(pika.ConnectionParameters(host='localhost'))

# create channel
ch = connection.channel()

# exchange topic
ch.exchange_declare(exchange='topic_logs', exchange_type='topic')

# messages and pattern
message = {
    "error.warning.important": 'This is important message',
    "info.debug.notImportant": 'This is not important message',
}

for k, v in message.items():
    ch.basic_publish(exchange='topic_logs', routing_key=k, body=v)

print('send')
# close connection
connection.close()

```

Sample Code Python
receiver=consumer
Exchange=Topic

sender.py | info_receiver.py | error_receiver.py

topic > info_receiver.py > ...

```python

# todo connect to server rabbitmq
connection = pika.BlockingConnection(pika.ConnectionParameters(host='localhost'))
# create channel
ch = connection.channel()

# declare exchange topic
ch.exchange_declare(exchange='topic_logs', exchange_type='topic')
result = ch.queue_declare(queue='', exclusive=True)
# get name queue
qname = result.method.queue

# binding key
binding_key = '#.notImportant'
# binding
ch.queue_bind(exchange='topic_logs', queue=qname, routing_key=binding_key)

print('Waiting for message')

# method callback
def callback(ch, method, properties, body):
    print(f'Received {body}')

# consumer
ch.basic_consume(queue=qname, on_message_callback=callback, auto_ack=True)
# start consuming
ch.start_consuming()

```

Sample Code Python
receiver=consumer
Exchange=Topic

sender.py | info_receiver.py | error_receiver.py

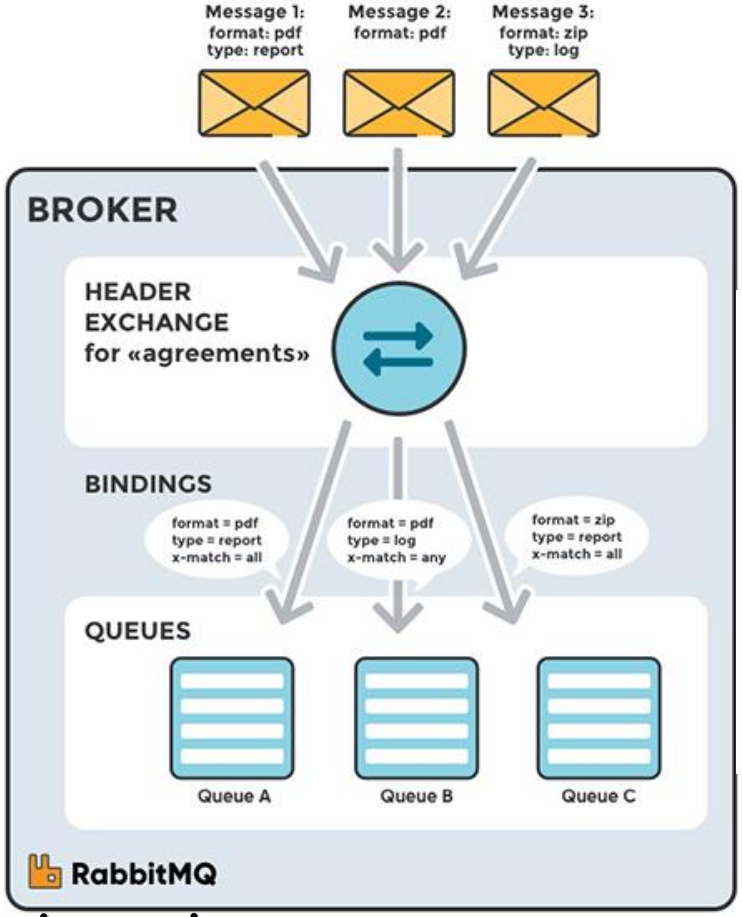
topic > error_receiver.py > ...

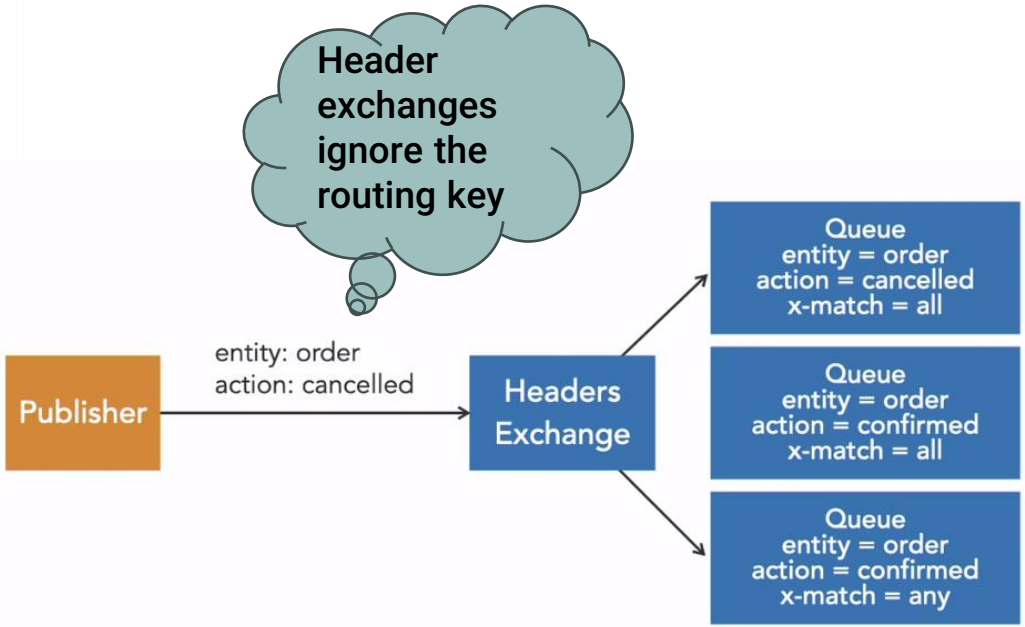
```python
import pika

# todo connect to server rabbitmq
connection = pika.BlockingConnection(pika.ConnectionParameters(host='localhost'))
# create channel
ch = connection.channel()

# create exchange topic
ch.exchange_declare(exchange='topic_logs', exchange_type='topic')
result = ch.queue_declare(queue='', exclusive=True)
qname = result.method.queue

# binding key => use *
binding_key = '*.*.important'
# binding
ch.queue_bind(exchange='topic_logs', queue=qname, routing_key=binding_key)
print('Waiting for message')

# callback function => save error log
def callback(ch, method, properties, body):
    with open('error_logs.log', 'a') as lg:
        lg.write(f'{str(body)}' + '\n')

# consumer
ch.basic_consume(queue=qname, on_message_callback=callback, auto_ack=True)
# start consumer
ch.start_consuming()

```

Headers Exchange

- رفتار آن خیلی شبیه به Topic Exchange است با این تفاوت که تصمیم گیری بر اساس attributeهای دیگه ایه که توی header پیام ها مشخص میکنیم.
- توی این مدل از exchange هر queue یه binding از یک یا مجموعه ای از attribute های header پیام های ارسالی رو برای مقایسه و هدایت پیام ها به exchange معرفی میکنه. بعلاوه یه مقدار با عنوان **x-match** هم به **exchange** ارسال میکنه که مقدارش میتونه یکی از دو مقدار **any** یا **all** باشه که به صورت پیش فرض **all**.
- **all** برای exchange مشخص میکنه که تمام attribute های مشخص شده توسط queue باید توی header پیام ارسال شده وجود داشته باشه بعلاوه مقادیر مشخص شده توسط queue باید با مقادیر ارسال شده در header **یکسان** باشه.
- اما any به exchange میگه حتی اگه **یکی از attribute های مشخص شده** توسط queueدر header پیام ارسال شده وجود داشت و مقدار اون با هم مطابقت داشت، به queueهدایت بشه و queue علاقمند به دریافت پیام است.

Headers Exchange





Exchange Type Use Cases

مصرف کنندگان (consumers) هستند
که تعیین می کنند از کدام
نوع exchange باید استفاده کنید

Choosing Your Exchange

Broadcasting: fanout

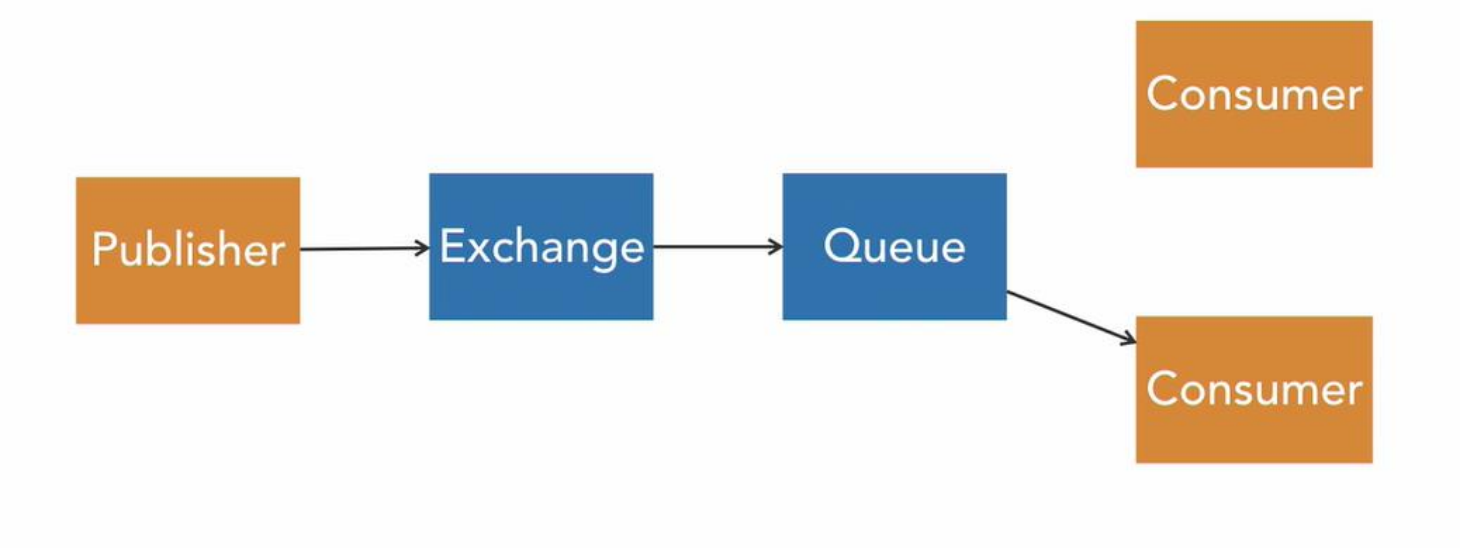
Simple scenarios: direct

Complex scenarios: topic

Special filtering: headers

Load Balancing between Consumers

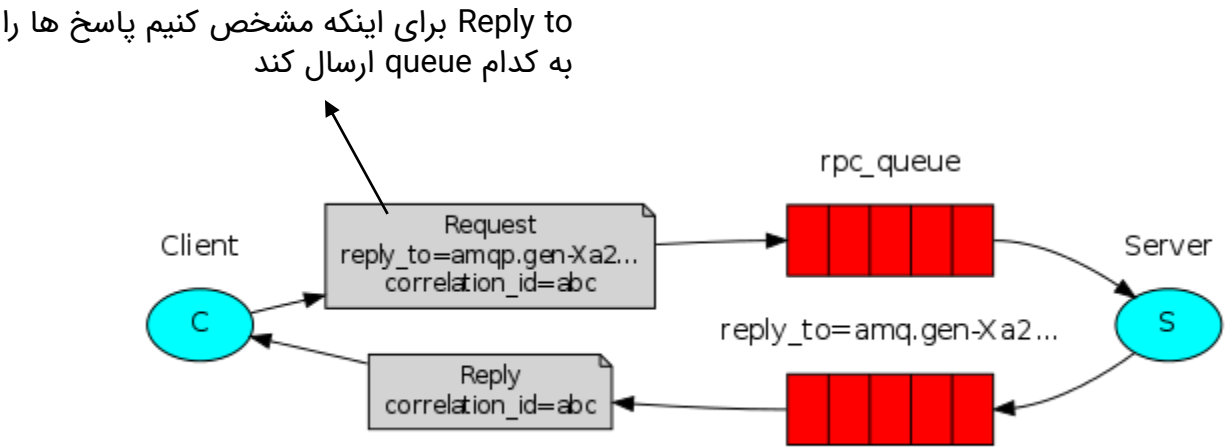
RabbitMQ will load balance between consumers. This means only one consumer will receive the message. The next message will go the other consumer.



One thing to keep in mind is that RabbitMQ will load balance messages between consumers, not queues. If the two applications are consuming the same queue, they will receive the messages one after the other.

rpc

ممکنه که ما بخواهیم برنامه ای بسازیم که یک درخواست رو به سرور بفرسته و بعد از اینکه سرور اطلاعات رو پردازش کرد پاسخ رو برگردونه، برای اینکار می توانید از rpc استفاده کنید در این حالت دو تا صف ایجاد کنیم که یکی برای ذخیره درخواست ها و دیگری برای ذخیره پاسخ ها استفاده می شود در ضمن هر درخواست باید یک id (correlation_id) منحصر به فرد داشته باشد که پاسخ اش با پاسخ بقیه درخواست ها قاطی نشود
نکته: در اینجا ما clinet و server هردو نقش:consumer, producer دارن.



Sample Code Python
client=consumer and publisher
Exchange=direct

client.py server.py

rpc > client.py > ...

```python
import pika
import uuid
# client has consumer rule and publisher rule
class Sender:
    def __init__(self):
        self.connection = pika.BlockingConnection(pika.ConnectionParameters(host='localhost'))
        self.ch = self.connection.channel()
        # replay queue
        result = self.ch.queue_declare(queue='', exclusive=True)
        self.qname = result.method.queue
        self.ch.basic_consume(queue=self.qname, on_message_callback=self.on_response, auto_ack=True)
    def on_response(self, ch, method, proper, body):
        if self.corr_id == proper.correlation_id:
            self.response = body

    def call(self, n: int):
        # check response
        self.response = None
        # correlation_id => should string
        self.corr_id = str(uuid.uuid4())
        # routing key => queue requests
        send.ch.basic_publish(exchange='', routing_key='rpc_queue',
                              properties=pika.BasicProperties(reply_to=self.qname, correlation_id=self.corr_i
                              body=str(n))
        while self.response is None:
            self.connection.process_data_events()
        return int(self.response)
# create instance from Sender class
send = Sender()
response = send.call(20)
print(response)
```

Sample Code Python
server=consumer and publisher
Exchange=direct

client.py ● server.py ×

rpc > server.py > ...

```python
import pika
import time

 # todo connect to server rabbitmq
connection = pika.BlockingConnection(pika.ConnectionParameters(host='localhost'))
# create channle
ch = connection.channel()
# declare queue
ch.queue_declare(queue='rpc_queue')

def callback(ch, method, proper, body):
    # get n from client
    n = int(body)
    print('processing message')
    time.sleep(4)
    # increase n
    response = n + 1
    # publish response to reply to queue with correlation_id
    ch.basic_publish(exchange='', routing_key=proper.reply_to,
                     properties=pika.BasicProperties(correlation_id=proper.correlation_id),
                     body=str(response))
    ch.basic_ack(delivery_tag=method.delivery_tag)


ch.basic_qos(prefetch_count=1)
# consumer
ch.basic_consume(queue='rpc_queue', on_message_callback=callback)
print('Waiting for message')
# start consuming
ch.start_consuming()

```

Authentication
And
Authorization
In RabbitMQ

ما به روشی نیاز داریم تا مطمئن شویم کدام برنامه‌ها به RabbitMQ متصل می‌شوند. همچنین مبنایی برای تعیین اینکه یک برنامه مجاز به انجام چه کاری است اینجاست که احراز هویت (authentication) و مجوز (Authorization) وارد عمل می شود.

Authentication vs. Authorization

Authentication: who are you?

Authorization: what are you allowed to do?

RabbitMQ Authentication

این امکان وجود دارد که به وسیله نام کاربری و رمز عبور(username/password) و مجوزهای دسترسی مشخص شده به RabbitMQ وصل شد.

استفاده میکنه از X.509 certificates

Authentication Back Ends

- RabbitMQ
- LDAP (Lightweight Directory Access Protocol)
- HTTP (Hypertext Transfer Protocol)

By default, RabbitMQ creates a username guest with a password guest.

This user can only connect from a local host. This is useful for quick testing or development.

But in production, it's better to remove this user and create a new user for every application that must connect to RabbitMQ.

Also, don't forget to create one or more users to log into the management UI.

In summary

Best Practices

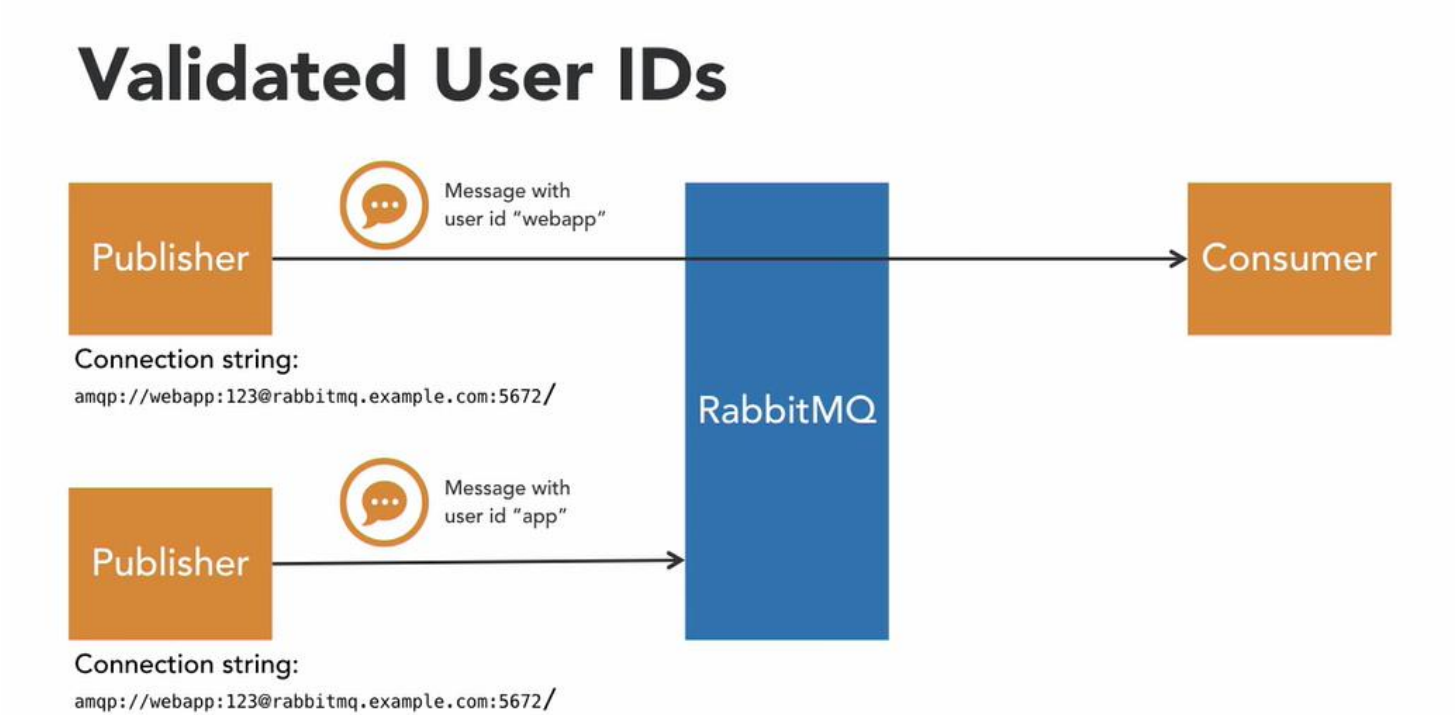
- Remove “guest:guest” user
- User per application
- Users for management UI

Point:

What are the resources in RabbitMQ that can be manipulated and are covered by authorization?

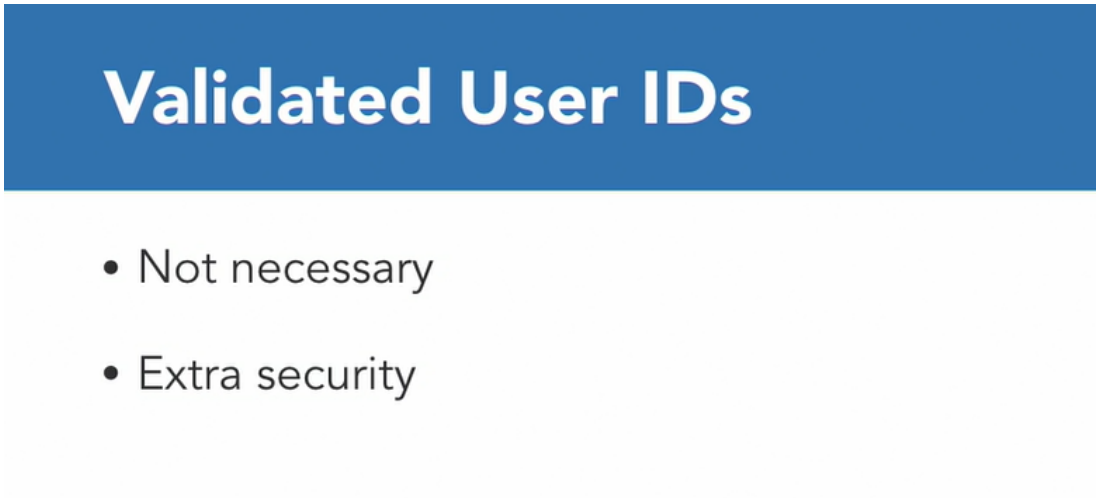
Exchanges and queues.

Bindings are part of a queue or exchange (depending on the operation).



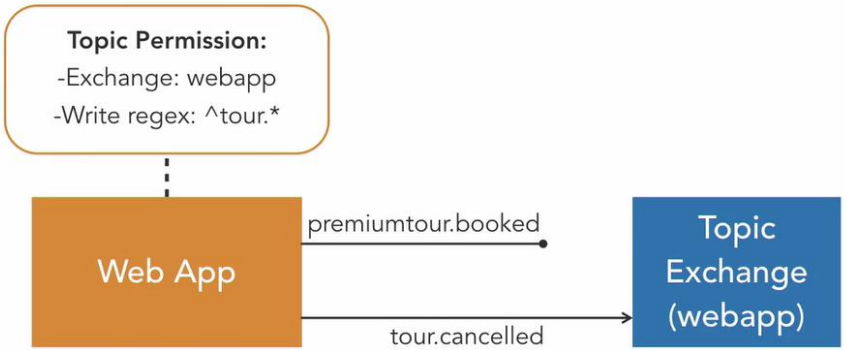
وقتی پیامی از RabbitMQدریافت می‌کنیم، به‌طور پیش‌فرض نمی‌دانیم کدام کاربر یا برنامه این پیام را منتشر کرده است. اینجاست که validated user IDsوارد عمل می شوند. هنگام انتشار پیام از طریق RabbitMQ، می‌توانید شناسه کاربری اضافه کنید. این نام کاربری برنامه شما است که برای ارتباط با RabbitMQاستفاده می‌کنید. اگر شناسه کاربری با نام کاربری در رشته اتصال مطابقت نداشته باشد، پیام رد می‌شود. این بدان معناست که RabbitMQشناسه کاربری در پیام را تأیید نمی‌کند. به این ترتیب، مصرف‌کننده می‌تواند مطمئن باشد که پیام از یک کاربر خاص نشات گرفته است.

به طور کلی ، حذف UserId اشکالی ندارد. وقتی ویژگی تنظیم نشده باشد RabbitMQ هویت ناشر را برای مصرف کننده فاش نمی کند اما می تواند **یه لایه امنیتی** اضافی کند.



Chat Users

Topic Permissions



The topic permissions are defined for a specific exchange, and the rooting key of the message is matched against the regular expression that you have defined in the right regex field. In this example, the web app is allowed to send the tour.cancelled message, but RabbitMQ will refuse the premiumtour.booked message because the rooting key does not match its right regular expression. With these options, you should be able to restrict publishers and consumers to allow only the operations they need. This really allows you to thoroughly secure your RabbitMQ instance.

مکانیسم تضمین دریافت پیام توسط دریافت کننده

- از اونجایی که شبکه قابل اطمینان نیست و ممکنه به هر دلیلی قطع بشه و یا نرم افزار ممکنه در حین دریافت پیام از کار بیوفته پروتکل AMQP روشی رو برای اطمینان از دریافت پیام توسط consumer یا دریافت کننده داره که بهش **message acknowledgement** گفته میشه. وقتی پیام به دست دریافت کننده میرسه، دریافت کننده، message broker رو با خبر میکنه و بهش اعلام میکنه که پیام رو دریافت کرده. این کار به دو روش انجام میشه، یا برنامه نویس این کار رو بر عهده میگیره و لابلای دستوراتش این کار رو بصورت دستی انجام میده یا عملیات بصورت اتوماتیک انجام میشه. بعلاوه کل این عملیات میتونه بصورت تک به تک برای هر پیام اتفاق بیوفته یا برای گروهی از پیام ها.
- message broker زمانی پیام رو بصورت کامل از queue پاک میکنه که دریافت کننده خبر دریافت پیام رو برای message broker ارسال کرده باشه.
- اما حالت های خاصی هم ممکنه بوجود بیاد مثلا message broker نتونه پیام رو routeکنه یعنی پیامرسان نتونه پیام رو از exchangeبه queueهدایت کنه. وقتی message broker نتونه پیام رو به queue هدایت کنه اونوق ممکنه پیام به publisher یا ارسال کننده بازگشت داده بشه، یا اگه message broker برای این موضوع پیاده سازی انجام داده باشه، پیام ها به جایی ارسال بشن به اسم **dead letter queue**
- یادتون باشه که publisher یا همون ارسال کننده مشخص میکنه که اگه پیامرسان نتونست پیام ها رو به queue مورد نظر routeکنه چه اتفاقی برای پیام ها بیوفته.

مفهوم **Acknowledgments and Confirms**: شاخص‌هایی برای مشخص کردن اینکه پیام دریافت شد یا پردازش شد. Acknowledgementsمیتواند در هر دو طرف استفاده شود؛ برای مثال، یک Consumer میتواند به سرور اطلاع بدهد که پیام دریافت یا پردازش شد، و سرور هم میتواند همچنین گزارشی را به Producer بدهد.

اکثر سیستم های پیام رسانی بالغ راهی برای مقابله با پیام هایی دارند که به جایی نمی رسند. این پیام ها را deadletters می نامند.

در RabbitMQپیام ها زمانی می توانند به dead letters تبدیل شوند که یکی از موارد زیر رخ دهد:

Conditions for Dead Letters

- Rejection or negative acknowledgment without requeue
- Message expiration due to TTL (time to live)
- Queue length limit exceeded

پیام توسط مصرف کننده رد یا تایید منفی می شود و مصرف کننده به RabbitMQ می گوید که پیام را دوباره ارسال نکند

یا پیام منقضی می شود به دلیل یک گزینه زمانی (زمان انقضا)که در صف یا روی پیام مشخص شده است.

دلیل آخر می تواند این باشد که صف از حد مجاز فراتر رفته است

What Happens to Dead Letter Messages?

. این پیام های dead letter از طریق **dead letter exchange** منتشر می شوند.
این یک تبادل منظم است که می توانید هنگام اعلام صف مشخص کنید.
RabbitMQ همچنین چند **header اضافی** به پیام اضافه خواهد کرد که به شما اجازه می دهد مشخص کنید از کجا آمده و **چرا و چه زمانی dead** است.

x-death یک header به نام x - Death اضافه می شود.

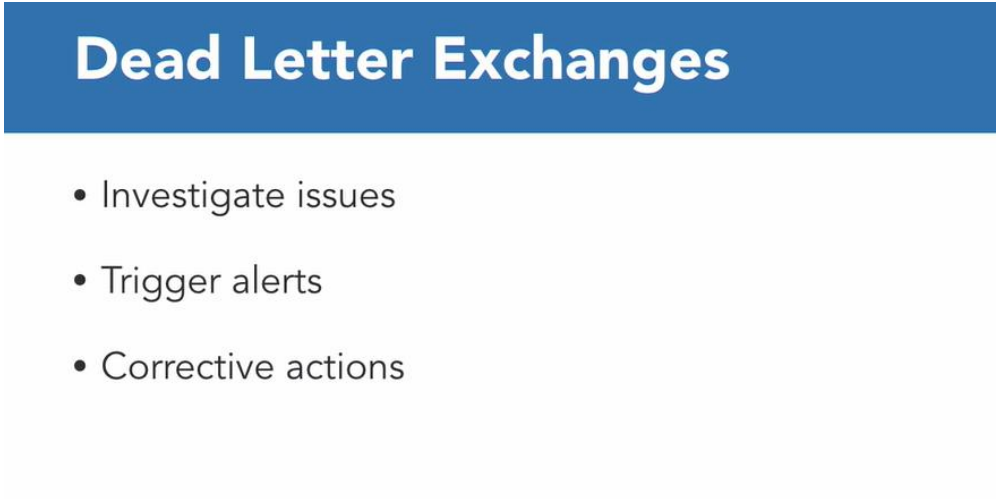
- queue
- reason
- time
- exchange
- routing-keys

The first time a message is dead-lettered RabbitMQ will also add three headers containing details of the original dead-lettering event. These values queues never change

Original Dead Letter Event Details

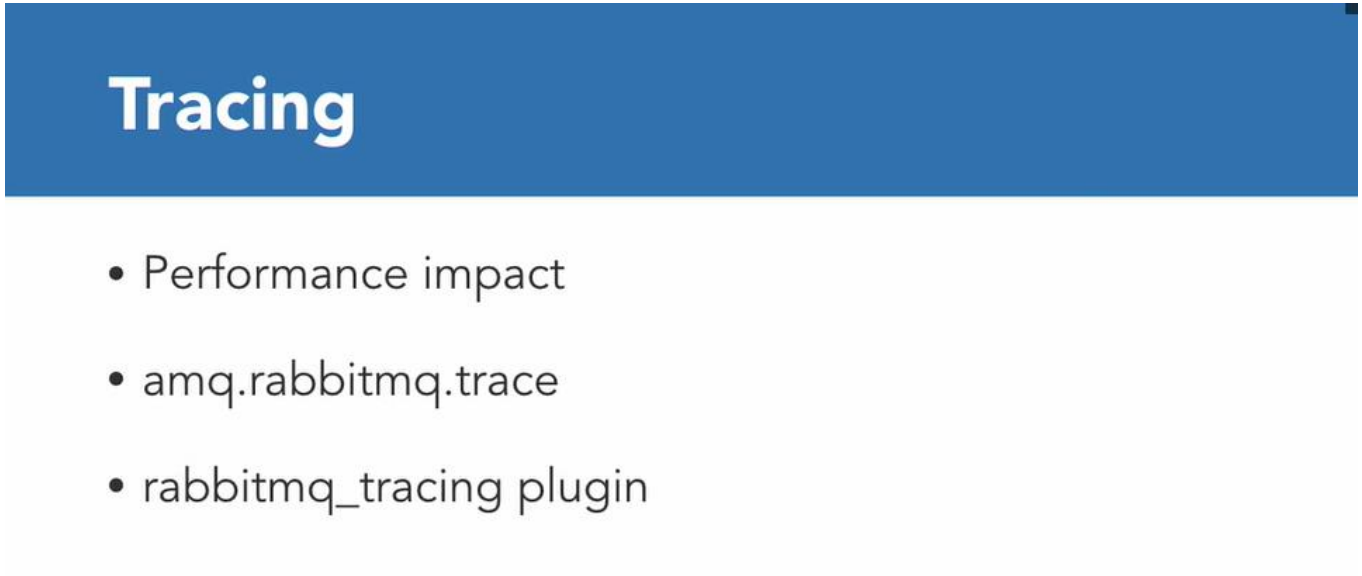
- x-first-death-reason
- x-first-death-queue
- x-first-death-exchange

Dead Letter **یک الگوی رایج در سیستم های پیام رسانی** است.
این قابلیت به شما اجازه می دهد تا مشکلات احتمالی (investigate potential issues) اپلیکیشن های خود را بررسی کنید. همچنین ممکن است برنامه هایی داشته باشید که پیام های dead letter را برای ایجاد هشدار (Trigger alert) یا اجرای اقدامات اصلاحی (execute corrective actions) مصرف می کنند.
همه این ها در تمام پیام های رد و بدل شده، یکی از ویژگی های مفید RabbitMQ هستند و برای هر سیستم پیام رسانی حیاتی هستند.



Tracing in RabbitMQ

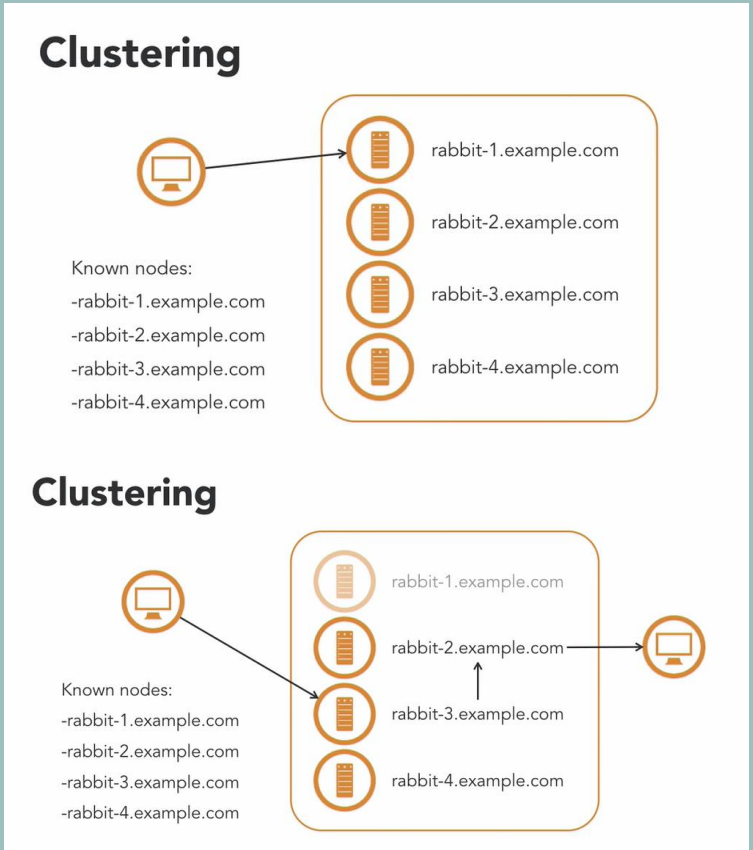
شما تنها باید در زمان توسعه یا هنگام اشکال زدایی از ردیابی در RabbitMQ استفاده کنید زیرا بر عملکرد**(Performance)** تاثیر می گذارد.
زمانی که فعال می کنید tracing اتفاقی که می افتد پیام های اضافی ارسال می شود به Exchange خاصی که مربوط به tracing



معایب RabbitMQ چیست؟

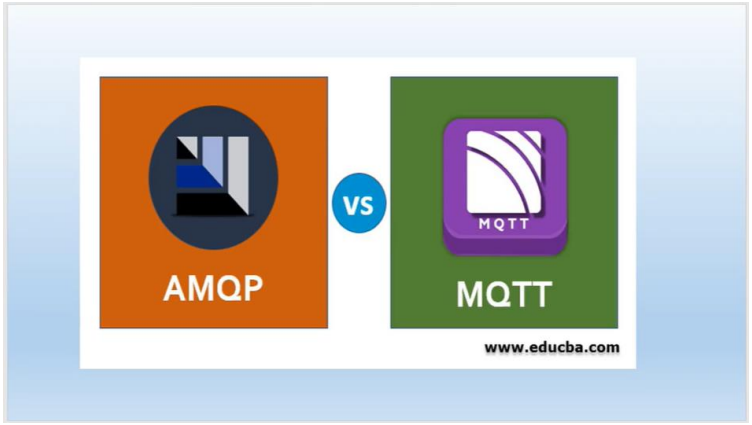
1. عیب یابی مشکلات می تواند دشوار باشد
2. برای کاربردهای با کارایی بالا مناسب نیست
3. ابزار مانیتورینگ داخلی ندارد

RabbitMQ supports clustering



This means that **you** can **handle heavy loads** but still appear to be a single logical broker to the client applications.
A client will have a list of node addresses that connect to a single node.
When a node **fails**, the client will **simply** be able to connect to **another node.**
Publishers and consumers don't have to be connected to the same node.
A RabbitMQ will ensure that messages are routed to the correct nodes.
In the case of a node failure in a cluster, any messages and queues on this node will be lost. **Highly available queues** are the answer to this.
If you need it In the case of a node failure, the other queues will contain any undelivered messages and applications will still receive them.

MQTT



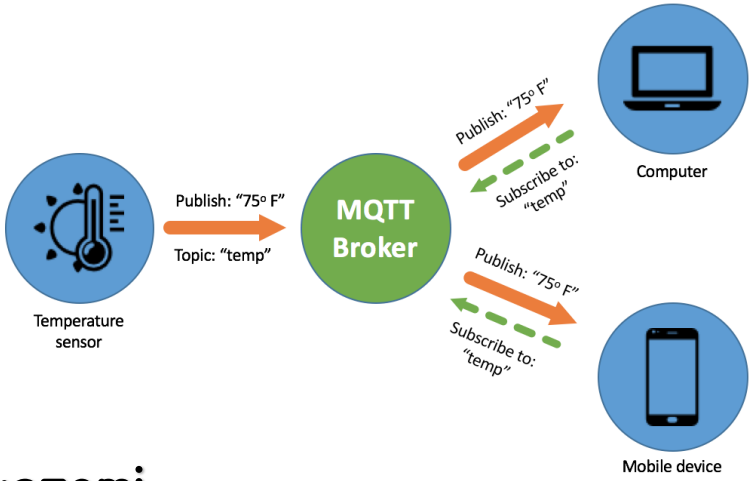
پروتکل MQTT چیست؟

پروتکل MQTTیکی از مهم ترین و پرکاربردترین **پروتکل ها در حوزه اینترنت اشیا** بشمار می رود.

MQTT یک پروتکل M2M یا machine to machine است.
ارتباط این پروتکل بر اساس ارتباطات کلاینت سروری می باشد و همچنین در لایه شبکه یا Network قرار دارد.
MQTT پروتکلی است که بر روی پروتکل TCPکار می کند و برای شبکه هایی بر روی TCPکار می کند مناسب می باشد.
ورژن دیگری از MQTTوجود دارد که برای شبکه های UDP مناسب است.

معماری : MQTT

MQTT از مدل Publish/Subscribبرای ارتباط خود استفاده می کند.
به این صورت که هر دیوایس یا سنسور در این پروتکل اطلاعات خود را به سرور MQTT که در اینجا یک Broker است، Publishمی کند به عنوان مثال یک سنسور دما اطلاعات خود را با استفاده از یک topicخاص تحت عنوان temp بر روی سرور Publishمی کند، در اینجا هر کسی که بخواهد به این اطلاعات دسترسی داشته باشد یا به اصطلاح Subscribکند، باید آن Topicخاص را که در اینجا temp است، داشته باشد تا بتواند اطلاعات را از سرور دریافت کند. در واقع دستگاه های یا Clientها در MQTTهم می توانند Publisherو هم Subscriberباشند.
در MQTTمحدودیتی برای تعداد Topicها وجود ندارد. همچنین هر دستگاه می تواند داده های خود را در چند Topicمختلف قرار دهد.



سرفصل پیام (Topic) چیست؟
سرفصل یک رشته حروف است که به سرور واسطه این امکان را میدهد تا پیامها را بر اساس آن برای هر مصرف کننده پالایش کند. هر سر فصل از یک یا چند زیر سرفصل تشکیل میشود که با کاراکتر فوروارد اسلش (/) از هم جدا میشوند.
مطابق نمونه زیر:
Home/Devices/Temperature
در این مثال Temperatureزیر سرفصل Devicesو Devicesزیر سرفصل Homeمیباشد.

پروتکل MQTT

- پروتکل MQTT یک پروتکل ارتباطی بر مبنای TCP/IPاست که به دلیل کم حجم بودن دیتای ارسالی و دریافتی در مقایسه با HTTP به پهنای باند کمتری برای برقراری ارتباط نیاز دارد.
- این خصوصیت MQTT آن را به پروتکل ارتباطی بسیار مناسبی در زمینه اینترنت اشیاء بدل میکند.
- جایی که تعداد زیادی از سنسورهای کم قدرت که اغلب با باتری کار میکنند وجود دارند.
- MQTT بر اساس مدل پیامرسانی انتشار و اشتراک (Pub/Sub) کار میکند که دارای خصوصیاتی از قبیل نگهداری پیام و آخرین وصیت (LWT) میباشد.
- رمزنگاری , احراز هویت و کنترل دسترسی به راحتی در این پروتکل قابل انجام است.
-
- MQTT با فراهم کردن ویژگی کیفیت خدمت (QoS) امکان ارسال پیام در بسترهای ارتباطی با کیفیت پایین و غیر قابل اعتماد را فراهم میکند. ساختار سرفصل پیام (Topic) در این پروتکل به صورتی است که به کاربر امکان ساماندهی پیامهای ارسالی و دریافتی را میدهد. برای استفاده از این پروتکل به یک سرور واسطه (Broker) نیاز هست که معروفترین آنها به نام Mosquitto به صورت متن باز در اختیار علاقمندان قرار دارد.

مفهوم QOS در MQTT:

مفهوم QOS به معنی مدیریت ارسال اطلاعات در شبکه است.

با استفاده از این می توان سرعت انتقال داده ها، زمان ارسال اطلاعات، حجم ارسال اطلاعات، نحوه ی ارسال اطلاعات و ... را کنترل کرد.

MQTTدارای سه نوع Qosاست:

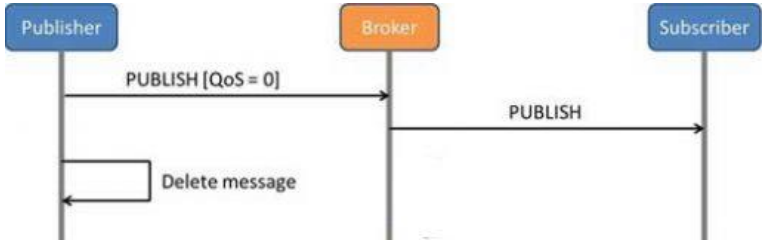
Qos0 :

ساده ترین حالت برقراری ارتباط در MQTT با Broker همین حالت می باشد.

در Qos0دستگاه Publisherکه اطلاعات را به Broker ارسال می کند هیچ نیازی به Ack (تائید) ندارد.

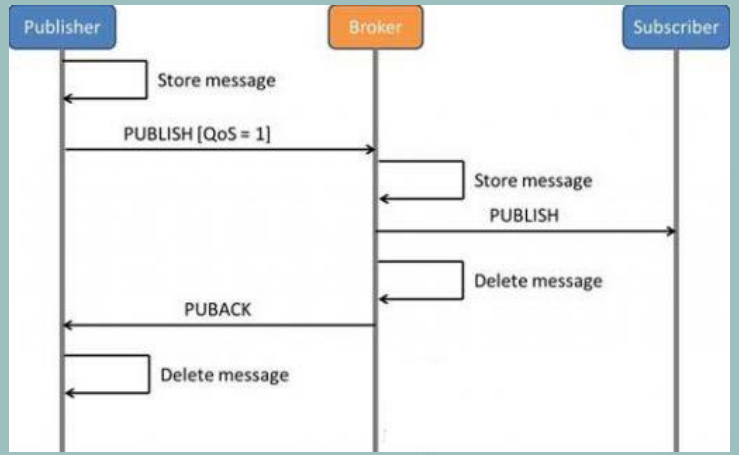
بنابراین هیچ اهمیتی ندارد که پیام به Broker رسیده باشد یا خیر، همین که در Qos0دستگاه Publisher بتواند Ackمربوط به شبکه TCP را دریافت کند فرض به دریافت اطلاعات از طرف Brokerمی کند. در این روش اگر ارتباط قطع شود و هنوز اطلاعات به آن دستگاه نرسیده باشد، بخشی از اطلاعات هیچ گاه به Brokerنمی رسد.

در ارتباطات ساده این روش قابل قبول است زیرا که **دارای مصرف انرژی پایینی** می باشد.



Qos1

در این روش دستگاه مطمئن می شود که حداقل یک بار پیام توسط سرور دریافت می شود.
به این صورت که Broker پس از دریافت پیام، یک پیام Puback به دستگاه Publisher ارسال می کند. سپس دستگاه پس از دریافت Puback مطمئن می شود که پیام به Brokerرسیده است و پیام را از صف ارسال خارج می کند. اگر به هر دلیلی پیام Puback ارسال نشود دستگاه دوباره پیام را ارسال خواهد کرد تا زمانی که پیام Puback دریافت شود. اما اگر پیام دوباره ارسال شود، Publisher یک Flagتکراری (DUP) ایجاد می کند.
در Qos1 این پرچم تنها برای اهداف داخلی استفاده می شود.
در هر بسته Publisher یک PachID یا شناسه بسته استفاده می شود تا بسته Publishشده را با Puback مطابقت دهد. هر کدام از Pubackها نیز یک PackIDدارند.



Qos2:

این حالت مطمئن ترین و امن ترین حالت ارسال پیام و البته کندترین حالت در MQTTاست.
در این Qos فرستنده یا Publisherمطمئن می شود که تنها یک بار پیام توسط Subscriberها دریافت شود. در Qos2از دو پیام req/resp بین Sender و Receiver برای اطمینان از ارسال و دریافت پیام استفاده می شود. فرستنده و گیرنده از شناسه بسته اصلی (Publish) برای هماهنگی و تحویل پیام استفاده می کنند. روال کار به این شکل است که وقتی **Broker** پیام Publishشده را از دریافت می کند، پیام دریافت شده پس از پردازش توسط **Broker** با ارسال یک پیام **pubrec** که برای تائید یک **Publish** است تائید می شود. هنگامی که publisher یک بسته **pubrec** دریافت می کند **publisher** می تواند با خیال راحت بسته اولیه Publish را حذف کند. **Publisher** بسته **Pubrec** را که **Broker** ارسال کرده، ذخیره می کند و با یک بسته **Pubrel** پاسخ می دهد.
پس از این که **Broker** پیام **Pubrel**را دریافت کرد می تواند تمام پیام ها و حالت های ذخیره شده را حذف کند و با یک بسته **Pubcomp** پاسخ بدهد.
Publisher هم پس از دریافت **Pubcomp** می تواند تمام پیام ها را حذف کند.

بنابراین:

در MQTTمیتوان کیفیت خدمت هر پیام ارسالی را تعیین کرد به طور کلی سه نوع کیفیت خدمت وجود دارد:
QoS Level 0 :
ساده ترین حالت برقراری ارتباط با brokerهست که نیازی به acknowledgmentنداره .
QoS Level 1 :
در این حالت سرور مطمئن میشه که حداقل یک بار یا بیشتر بسته به کلاینت رسیده و ACKاز کلاینت میگیره.
QoS Level 2 :
در این حالت brokerیک و فقط یک بسته به کلاینت میفرسته و مطمئن میشه که بسته رسیده.این حالتش خیلی توصیه نمیشه .
همچنین این پروتکل حالاتی رو تشخیص میده که ارتباط قطع شده یا به هر دلیلی کلاینت نمیتونه با brokerارتباط برقرار کنه و در این حالت brokerاقدام لازم رو انجام میده تا به خطا بر نخوریم. در همه این شرایط brokerبقیه کلاینت ها رو هم با خبر میکنه.

- در نوع ۱ پیام ارسالی حداکثر یک بار ارسال میشود و تضمینی برای دریافت آن توسط مشترک وجود ندارد به این معنی که اگر مشترک به سرور واسطه متصل نباشد و این پیام را دریافت نکند سرور واسطه دیگر مجددا این پیام را ارسال نخواهد کرد.
- در نوع ۲ پیام ارسالی حداقل یک بار ارسال میشود به این معنی که تا مشترک یا مشترکین پیام را دریافت نکنند سرور واسطه پیام را در نوبت های بعدی ارسال خواهد کرد. و ممکن است مشترک آن پیام را حتی بیش از یکبار دریافت کند.
- در نوع ۳ تضمین میشود که مشترک پیام را فقط یکبار دریافت کند. این امن ترین در عین حال کندترین روش ارسال پیام است زیرا که به سیستم تصدیق ۴ مرحله ای نیاز دارد.

چند نمونه از نرم افزار های Broker

.

تا کنون brokerهای متفاوتی برای این پروتکل نوشته شده که در زیر چند نمونه رو معرفی میکنیم .

Mosquitto اولین نسخه brokerنوشته شده با زبان Cو دارای تنظیمات مختلف.

Mosca نوشته شده با Nodejsو بسیار محبوب ، سریع و قابل توسعه و با تنظیمات راحت.

RSMBنسخه نوشته شده توسط کمپانی IBMو کمتر محبوب اما با تنظیمات بسیار عالی ، پیاده سازی شده با زبان C.

HiveMQ این نسخه یک بازیگر جدیده که بسیار خوب ظاهر شده.گفته میشه که توی این نسخه قابلیت رمزگذاری TLSپشتیبانی میشه.

in eskazemi