Haute école d'ingénierie et d'architecture Fribourg
Hochschule für Technik und Architektur Freiburg

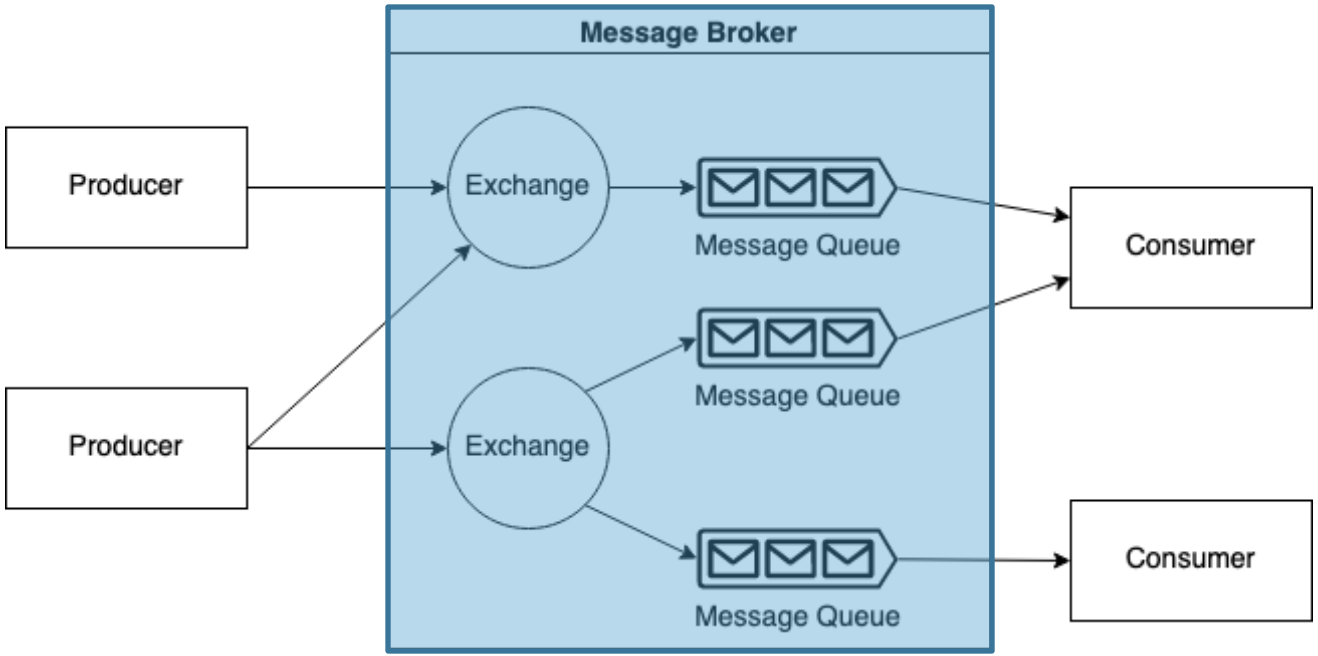
Messaging services
RabbitMQ

Hes·so
Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz

Messaging services – Msg broker

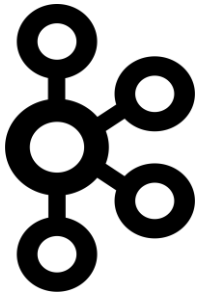
- Services that exchange messages are called **message broker**
- **Message broker** is an architectural platform that does the following with messages:
  - Validation
  - Transformation
  - Routing
- Based on telecommunication protocols → *Distributed computing*
  - Decoupling the different stakeholders of a distributed environment

Message broker – Big picture



Represents mainly the queue(s)

Some message brokers



Apache Kafka

Link: Type of brokers

RabbitMQ

Link : https://www.rabbitmq.com/tutorials

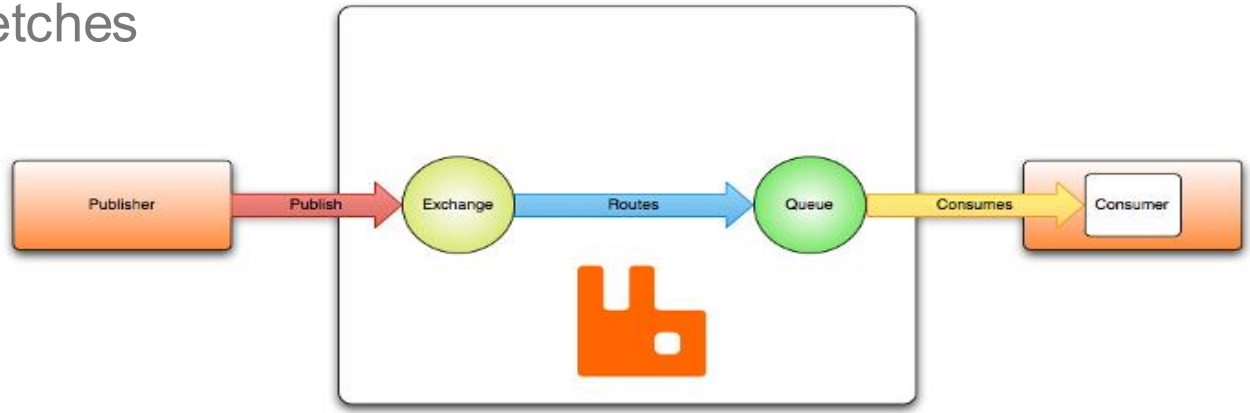
Our guinea pig

open source initiative

RabbitMQ

RabbitMQ – AMQP 0-9-1

- **A**dvanced **M**essage **Q**ueueing **P**rotocol
- Messages are published to ***exchanges*** *(like a mailbox, post office)*
- Exchange distribute the messages copies to ***queues*** using rules (called *bindings*)
- Broker finally delivers the messages
- ..or customers fetches the messages



RabbitMQ – AMQP 0-9-1 (cont.)

- Message attributes (message meta-data)
  - Can be used also by the broker (e.g. filtering)
- In unreliable networks, messages can be acknowledged → *message acknowledgments*
  - The consumer informs (acknowledge) the broker about delivery
  - E.g. message removed from queue only if acknowledged
- Routing problems
  - Messages returned to sender, messages are dropped, or placed in a *dead-letter* queue

https://www.rabbitmq.com

Key take away messages

- Messaging services (RabbitMQ) are message brokers
  - Exchange messages (validate, transform, route)
  - In a distributed environment
- Published to **exchanges**, distributed from **queues**
- RabbitMQ
  - Different languages
  - AMQP : Msg attributes, acknowledgements, routing management

Haute école d'ingénierie et d'architecture Fribourg
Hochschule für Technik und Architektur Freiburg

Messaging services
RabbitMQ - Hello world

Hes·so
Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz

RabbitMQ example – *Hello world*

- Have a look at https://www.rabbitmq.com/tutorials/tutorial-one-python.html
- Install Docker RabbitMQ image (https://www.rabbitmq.com/docs/download)
- A RabbitMQ service is running on
  - **concurp1.isc.heia-fr.ch** (Port 5072)
- Receiving messages works with **callback functions**
  - The callback function will be executed once the receiver gets a new message
    - A thread-safe solution will be seen later

Hello world – Connection

```
QUEUE = 'concurp-queue1'

conn_param = pika.ConnectionParameters(
        host='localhost',
        port=5672,
        # credentials=cred_param)

connection = pika.BlockingConnection(conn_param)

channel = connection.channel()

channel.queue_declare(queue=QUEUE)
```

Default credentials are used if no `pika.credentials` is given
https://pika.readthedocs.io/en/stable/modules/credentials.html#plaincredentials

Different callbacks can be defined already here (on_open, on_open_error, …)
https://pika.readthedocs.io/en/stable/modules/connection.html

Declares or creates a queue
https://pika.readthedocs.io/en/stable/modules/channel.html#pika.channel.Channel.queue_declare

Hello world – Publishing

```
the_body = 'Hello world from Python!'

channel.basic_publish(exchange='',
                      routing_key=QUEUE,
                      body=the_body)

connection.close()
```

https://pika.readthedocs.io/en/stable/modules/channel.html#pika.channel.Channel.basic_publish

code-snippets/255

Based on pika Python library

Hello world – Consuming

```
def callback_func(ch, method, properties, body):
    print(f"Channel: {ch}")
    print(f"Method: {method}")
    print(f"Props: {properties}")
    print(f"Received body: {body.decode()}")



channel.basic_consume(queue=QUEUE,
                      on_message_callback=callback_func,
                      auto_ack=True)

channel.start_consuming()       # start the consuming loop
```

https://pika.readthedocs.io/en/stable/modules/channel.html#pika.channel.Channel.basic_consume

https://pika.readthedocs.io/en/stable/modules/adapters/blocking.html?highlight=start_consuming_loop#pika.adapters.blocking_connection.BlockingChannel.start_consuming

RabbitMQ with Pika – Exercise

Exercise 22, Question 1-3 → Link

- Let's install the `pika` library for the Python environment
- Let's install the RabbitMQ image on Docker
- Let's play with RabbitMQ
  - Hello world
  - Distributed tasks

Haute école d'ingénierie et d'architecture Fribourg
Hochschule für Technik und Architektur Freiburg

Messaging services
RabbitMQ – AMQP Exchange types

Hes·so
Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz

AMQP – Programmable Protocol

- Entities and routing schemes are primarily **defined by applications themselves**, not a broker administrator
  - Accordingly, provision is made for protocol operations that declare queues and exchanges, define bindings between them, subscribe to queues and so on
- This gives application developers a lot of freedom but also requires them to be aware of potential definition conflicts
- Applications declare the AMQP entities that they need, define necessary routing schemes and may choose to delete AMQP entities when they are no longer used

https://www.rabbitmq.com

RabbitMQ-AMQP – Exchange types

- The AMQP protocol in RabbitMQ provides different **exchange types**:
  - Direct Exchange
  - Fanout Exchange
  - Topic Exchange
  - Headers Exchange
- Exchange **attributes**:
  - Name
  - Durability (survive broker restart) vs. transient
  - Auto-delete (delete exchange, when last queue is unbound)
  - Arguments

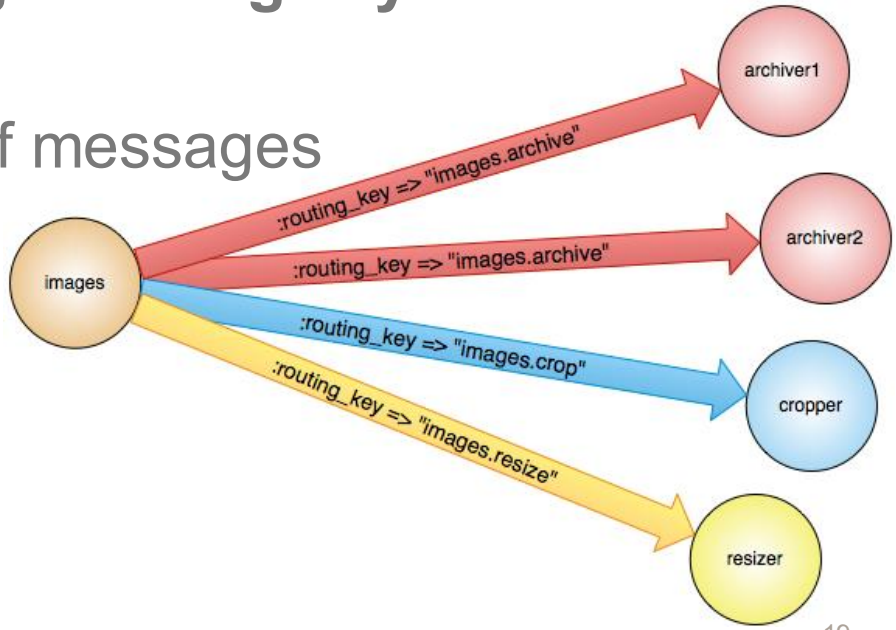
https://medium.com/trendyol-tech/rabbitmq-exchange-types-d7e1f51ec825

https://www.rabbitmq.com/tutorials/amqp-concepts.html

https://www.rabbitmq.com/tutorials/amqp-concepts.html

Exchange Types – Direct Exchange

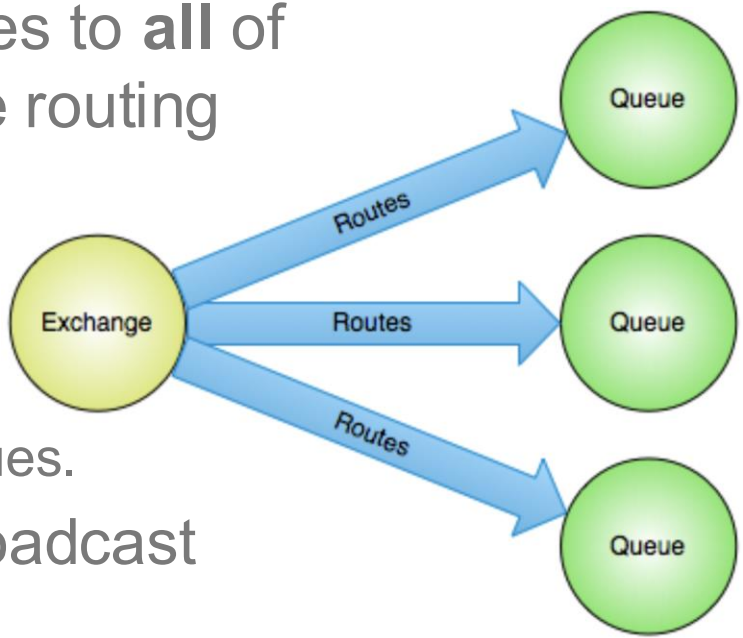
- A direct exchange delivers messages to queues based on the message **routing key**
  - The keys must match
- Ideal for the unicast routing of messages (although they can be used for multicast routing as well



https://www.rabbitmq.com/tutorials/amqp-concepts.html

Exchange Types – Fanout Exchange

- A fanout exchange routes messages to **all** of the queues that are bound to it, 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 broadcast routing of messages

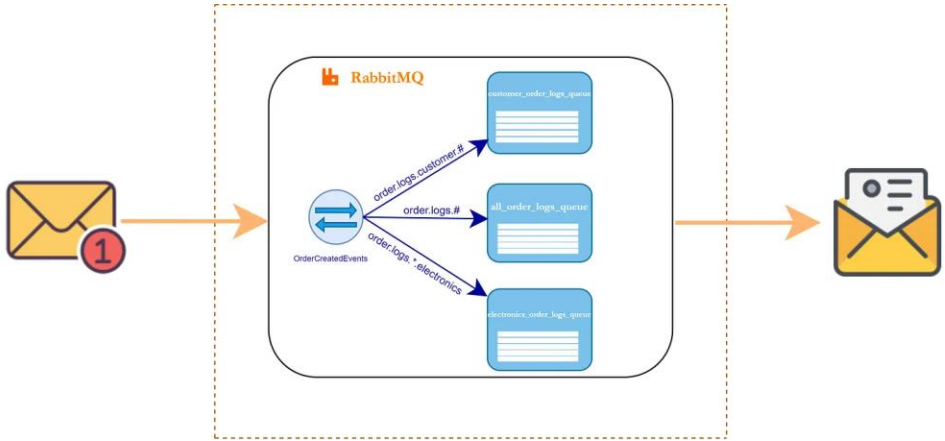


https://www.rabbitmq.com/tutorials/amqp-concepts.html

Exchange Types – Topic Exchange (Publisher/Subscriber)

- 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
- The topic exchange type is often used to implement various **publish/subscribe** pattern

variations. Topic exchanges are commonly used for the multicast routing of messages.

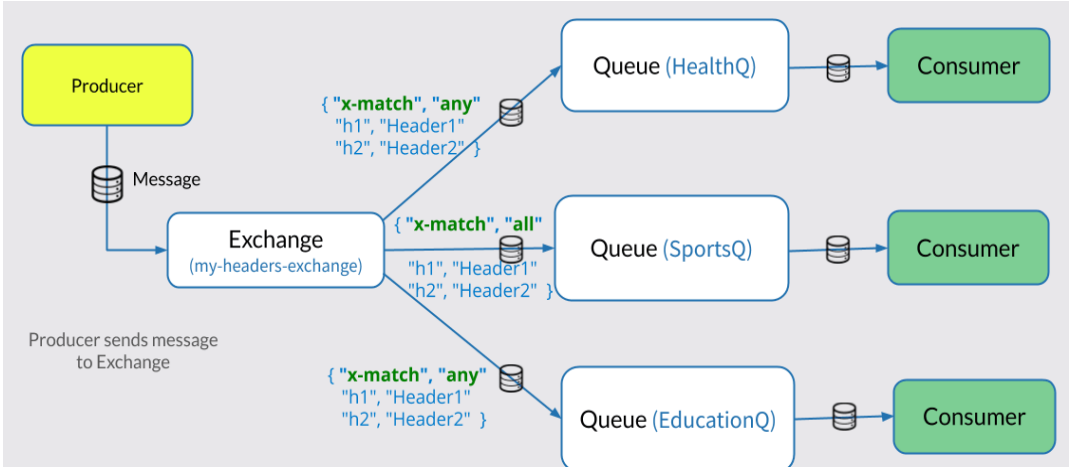


Source: https://medium.com/trendyol-tech/rabbitmq-exchange-types-d7e1f51ec825

https://www.rabbitmq.com/tutorials/amqp-concepts.html

Exchange Types – Header Exchange

- A headers exchange is designed for routing on multiple attributes that are more easily expressed as message headers than a routing key
- Headers exchanges ignore the routing key attribute
- Attributes used for routing are taken from the headers attribute. A message is considered matching if the value of the header equals the value specified upon binding



Key take away messages

- AMQP: Programmable protocol
  - Offers a lot flexibility
- Exchange types
  - Direct exchange (routing key)
  - Fanout exchange
  - Topic or Pub/Sub Exchange
  - Header exchange (different header attributes (routing keys) are considered

Haute école d'ingénierie et d'architecture Fribourg
Hochschule für Technik und Architektur Freiburg

Messaging services
RabbitMQ – Python Usage

Hes·so
Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz

RabbitMQ – Python examples

- Examples for the different RabbitMQ's exchange types in Python
  - Default exchange
    - Seen in previous slide deck (*hello world* example)
  - Fanout exchange (e.g. log distribution)
  - Direct exchange
    - Key binding
    - Routing

code-snippets/255

Pika

Philadelphia, PA https://pika.readthedocs.org

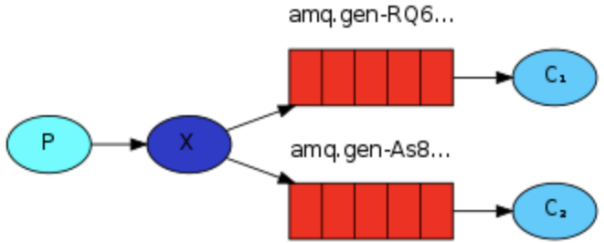
RabbitMQ – Fanout (sender)

- Exchange type must be set to **`fanout`**
  - `routing_key` will be ignored for `fanout` exchange

```
channel.exchange_declare(exchange='name', exchange_type='fanout')
```

- Message sending

```
channel.basic_publish(exchange='name', routing_key='', body=payload)
```



https://www.rabbitmq.com/tutorials/tutorial-three-python.html

RabbitMQ – Fanout (receiver)

- Exchange type must be set to **`fanout`**
  - `routing_key` will be ignored for `fanout` exchange

```
channel.exchange_declare(exchange='…', exchange_type='fanout')
```

- The queue should be declared as **exclusive**
  - A queue will be exclusively and dynamically created. This queue then will also be automatically deleted after usage

```
result = channel.queue_declare(queue='', exclusive=True)
```

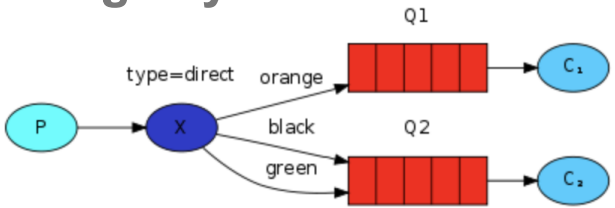
- The dynamically created queue name must be used for the **binding**

```
queue_name = result.method.queue
channel.queue_bind(exchange='…', queue=queue_name)
channel.basic_consume(
    queue=queue_name, on_message_callback=callback_func, auto_ack=True)
```

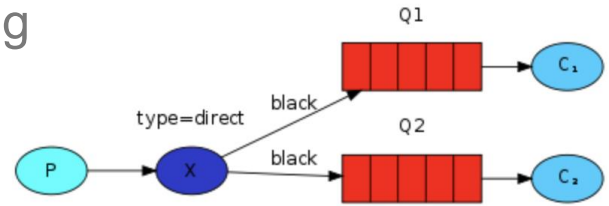
amq.gen-kRgkE2cg9NvAUj6keXA4mQ	classic	Excl

RabbitMQ – Direct exchange 1/2

- Exchange type must be set to **`direct`**
  - Direct exchange offers routing: The message goes to the queues whose **binding key** exactly matches the **routing key** of the message
    - Queue Q1 has one binding (orange)
    - Queue Q2 has two bindings (black, green)



  - Bind **multiple** queues with the same binding key (black)
    - In this case, the exchange behaves like a multicast/fanout



https://www.rabbitmq.com/tutorials/tutorial-four-python.html

RabbitMQ – Direct exchange 2/2

- Exchange type must be set to **`direct`**

```
channel.exchange_declare(exchange='…', exchange_type='direct')
```

- Publishing with the desired **routing key**

```
channel.basic_publish(exchange='…', routing_key='black', body=payload)
```

- Client's binding

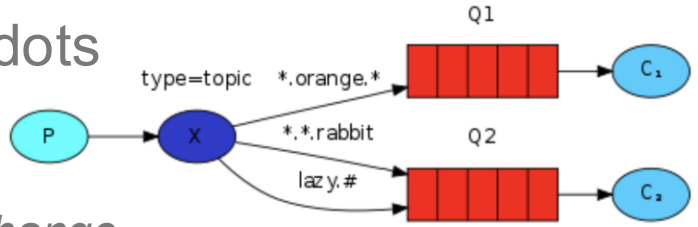
```
q_name = result.method.queue
ch.queue_bind(exchange='…', queue=q_name, routing_key='black')
```

Can be done multiple times for multiple bindings

https://www.rabbitmq.com/tutorials/tutorial-four-python.html

RabbitMQ – Topic exchange

- Exchange type must be set to **`topic`**
  - Allow routing on different criteria (e.g. severity, emitting source, etc.)
  - The routing_key must have a specific format (same as MQTT)
- Topics are given as a list, delimited by dots
  - E.g.: `<celerity>.<color>.<animal>`
  - The binding_keys must have the same format
  - The logic of the matching is like the *direct exchange*
- Special cases for binding
  - ***** (star) can substitute for exactly one word
  - **#** (hash) can substitute for zero or more words



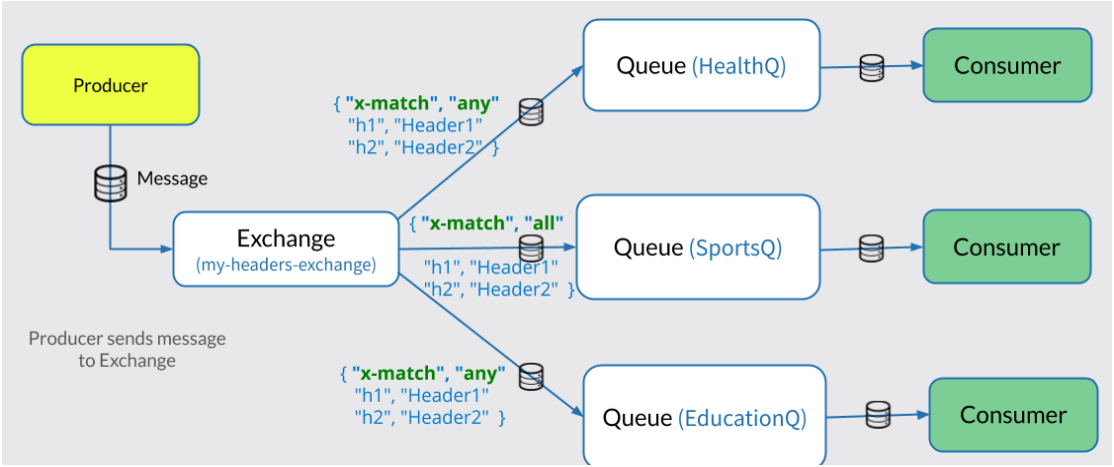

```
- *.orange.*
- lazy.#
- #
- quick.black.*
- *.*.rabbit
```

https://www.rabbitmq.com/tutorials/tutorial-four-python.html

RabbitMQ – Header exchange 1/2

- Exchange type must be set to **`header`**
  - Header exchange offers routing: The message goes to the queues whose **binding arguments** matches (part or all) the **header** of the message
  - Bind type **`all`**: like an AND function
  - Bind type **`any`**: like a OR function

```
bind_args = {
  'x-match': 'all',
  'h1': 'Header1',
  'h2': 'Header2'
}
```



RabbitMQ – Header exchange 2/2

- Producer must set header(s)

```
channel.basic_publish(…,
properties=pika.BasicProperties(headers={‘h1’:
‘Header1’, …}, …)
```

- Consumer must filter on header(s)

```
channel.queue_bind(…, arguments=
  {‘x-match’: ‘any’,
   ‘h1’: ‘Header1’,
 …},
…)
```

RabbitMQ – Default (direct) exchange for task distribution

- Standard asynchronous message passing (`hello world Example`)
- Default exchange is based on a direct exchange where queue_name = key_name
- Task distribution → some modifications are needed:
  - No automatic acknowledgment (in receiver)

```
channel.basic_consume(…, auto_ack=False, …)
```

  - acknowledgment only after task completion (in callback function)

```
ch.basic_ack(delivery_tag=method.delivery_tag)
```

  - don't dispatch a new message to a worker queue until it has processed and acknowledged the previous one. Instead, the *exchange* will dispatch it to the next worker queue that is not still busy (→ has an empty channel) → avoid a round-robin distribution

```
channel.basic_qos(prefetch_count=1)
```

Key take away messages

- Exchange types
  - Direct exchange (routing key)
    - Multiple binding
  - Fanout exchange
    - Sort of broadcasting
  - Topic exchange
    - Word list for matching
      - MQTT style
  - Header exchange
    - Binding args set as headers
    - Filtering on these args (any, all)