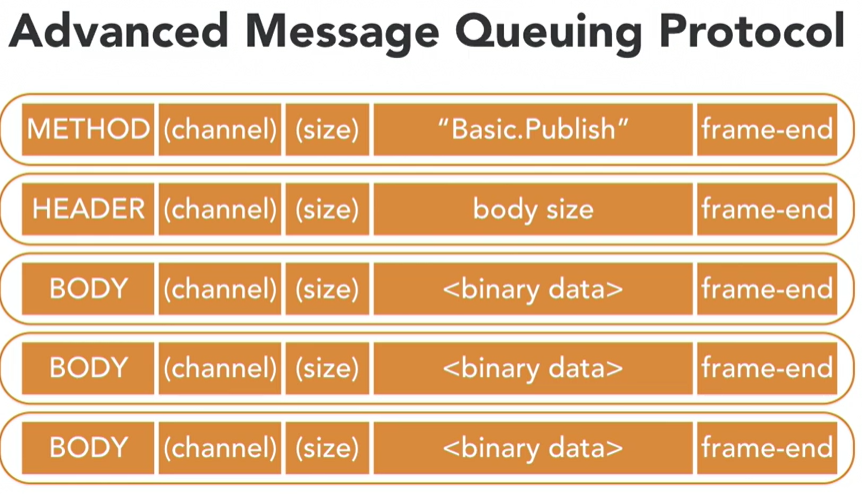
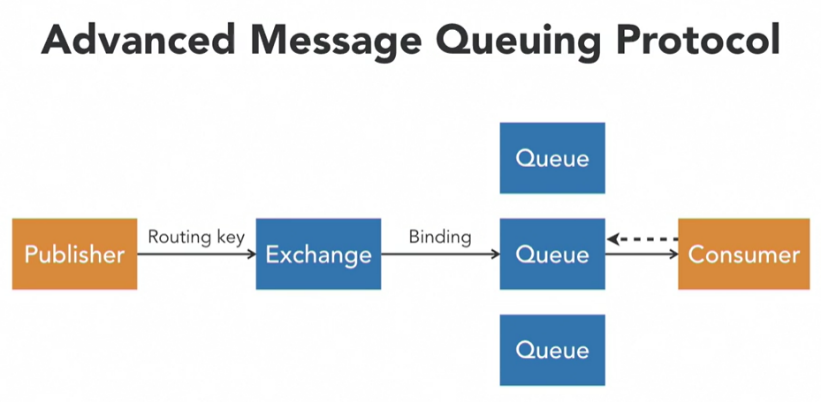
AMQP is a binary protocol.



The publisher connects to a message broker, in our case RabbitMQ, and it then publishes this message to an exchange. The publisher can also send a routing key along with the message. The exchange then passes this message on to the queues. The exchange will use certain rules to determine which queue to route the message to. These rules are called **bindings**, and they can use the routing key that the publisher included. Finally, the message is sent to the receiving application called the consumer. The broker will push the message to a subscribed consumer, but it's also possible to have the consumers fetch, or pull, the messages on the mound. Keep in mind that multiple applications can be subscribed to the same queue, but only one of them will receive a single message. As a last step, this consumer will send a message acknowledgment back to the message broker. This signals the message broker to delete the message from the queue.

Two configuration options that exchanges and queues have in common. Both can be defined as **durable or transient**. A durable queue or exchange will survive a RabbitMQ restart, and transient ones will not. Keep in mind that this doesn't mean that any undelivered messages are persisted unless you configure **persistence**, RabbitMQ keeps your messages in memory. This means they will be lost when the service restarts. It's also possible to tell an exchange or a queue to **auto-delete** itself if nothing is connected to it anymore. For example, a queue could automatically be removed when the last consumer disconnects.

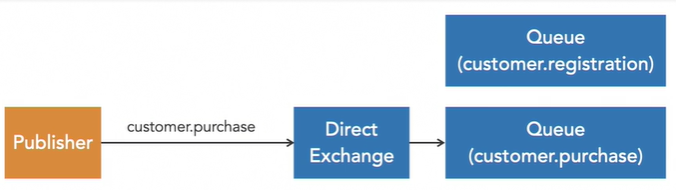
**Exchange types -**1. Direct

2. Topic

3. Fanout

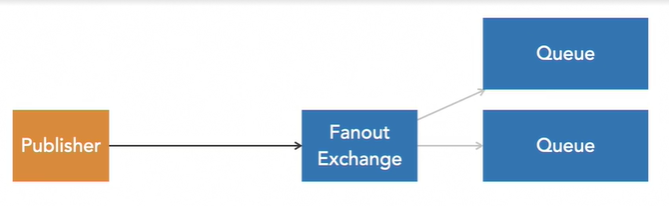
4. Headers

**Direct Exchange**



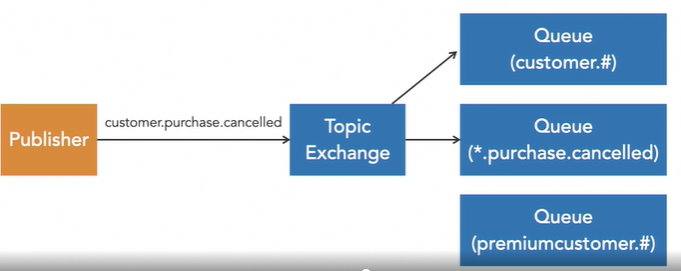
Route messages to all queues that have been connected to this exchange with the same routing key as the routing key that the publisher used. In above image, the publisher used the routing key customer.purchase. Let's assume there are two queues that have been bound to this exchange. One with the routing key customer.purchase and the other with customer.registration. Direct Exchange will now route the message to the queue with customer.purchase as a routing key. The other queue will not receive the message.

**Fanout Exchange –**



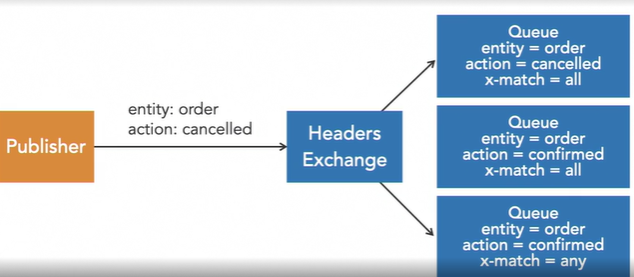
 Messages are routed to all bound queues and the routing key is just ignored. Every queue will receive a copy of the message.

**Topic Exchanges**



Route messages based on patterns in the routing key. In this example, we'll send a message and add the routing key customer.purchase.cancelled. Now if a binding was declared with a hash, this symbol matches zero or more words in the routing key. You can also use an asterisk, which will match a single word. A queue that uses premiumcustomer.# in this example will not match and the message will not be routed to this queue.

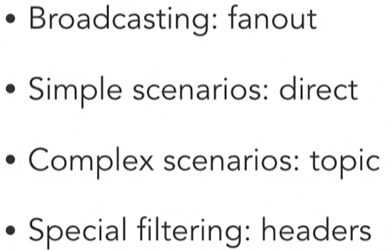
**Headers Exchange** –



will ignore the routing key and instead look at the headers that were sent with the message. Let's assume we have a publisher that sent a message with the headers, entity: order and action: cancelled. When we bind a queue to this exchange, we need to decide if we want to match all or any of these headers. A queue that has a binding with the arguments, entity = order, action = cancelled, and x-match = all will receive this message. The x-match = all arguments means all headers must match. With a queue that has a binding with the arguments, entity = order, action = confirmed, and x-match is all will not receive this message. The action argument differs from the header sent with the message. Finally, picture a queue that has a binding with the arguments, entity = order, action = confirmed, and x-match = any. This queue will receive the message. Even though not all headers match the arguments of the binding, at least one of them does. The entity header matches and that's enough.

Use cases of exchange –

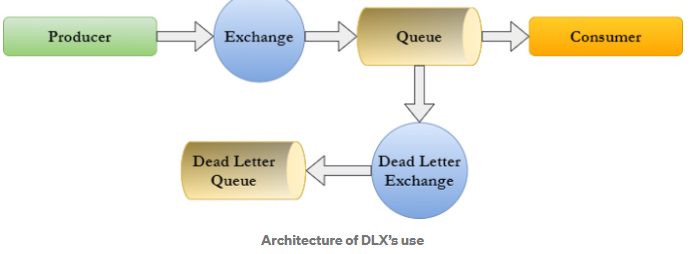
It’s the consumer which will decide the type of exchange.



Dead Letter Exchanges (DLXs) –

Consumers of the queues may not be able to process some notifications or even the queue itself may reject messages due to some events. Dead Letter Exchanges keeps those messages and reprocess them some other time. In below scenarios DLXs might be required.

1. **The message has expired due to the TTL (Time to Live):** in RabbitMQ, the argument x-message-ttl can be added to the queue to set how long a message can live. When the message’s TTL expires, the message is discarded.
2. **The message was reject by the consumer:**that can happen when, for some reason, the message could not be processed, so the consumer negatively acknowledged (Nack) it with the requeue flag set to false.
3. **The queue has reached its length capacity:**in RabbitMQ, the argument *x-max-length*can be added to the queue to set how many ready messages it can contain. When the capacity is reached, the next messages are dropped.



Installing RabbitMq

1. Install Erlang with administrative rights <https://www.erlang.org/downloads>
2. Install RabbitMq with administrative rights <https://www.rabbitmq.com/install-windows.html>
3. Ensure Erlang and RabbitMq versions are compatible with each other
4. Open command prompt and go to C:\Program Files\RabbitMQ Server\rabbitmq\_server-3.9.12\sbin folder
5. Run below command –

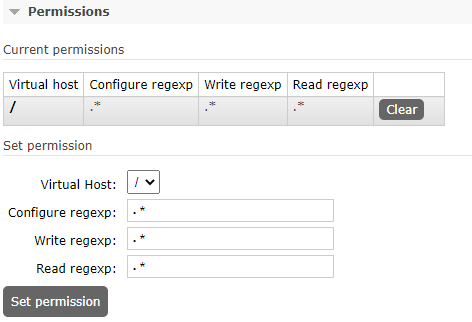
rabbitmq-plugins enable rabbitmq\_management

1. Login to RabbitMq dashboard <http://localhost:15672/>

Username and password - guest/guest

**Authentication and Authorization –**

1. In dashboard, goto admin tab and create a new user. Guest user works only with localhost
2. Click on the user and provide the required permissions –



1. Configure host, port, username, password in application.properties.

**@RabbitListener at class level -**

If we are using @RabbitListener annotation at the class level, then we need to add @RabbitHandler annotation for the required method. Based on the payload type appropriate method will be invoked.

If we are publishing the message from the Rabbitmq UI dashboard, then typeid needs to be set with the appropriate packagename.classname, else exception will be thrown –

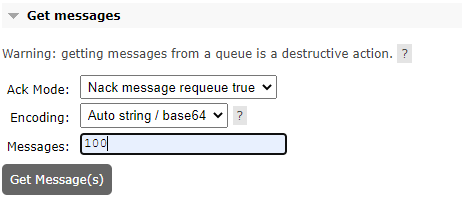
Book.java is the class name

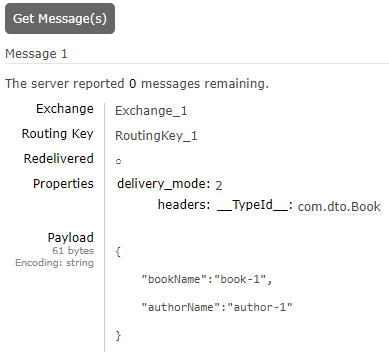
\_\_TypeId\_\_ = com.dto.Book

Note: use **content\_type .** content-type is not supported

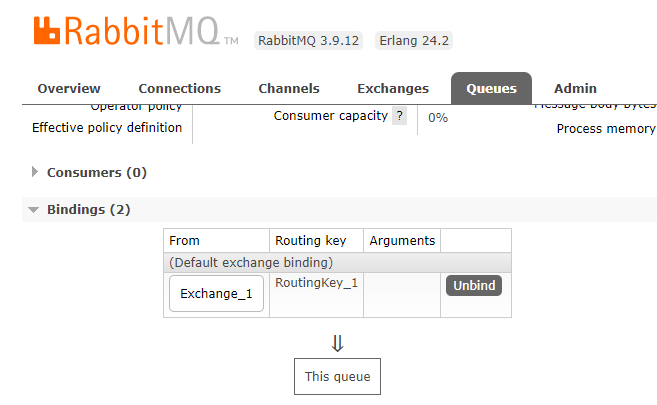


Goto queues, click on Queue name, click on Get messages to see the published messages.

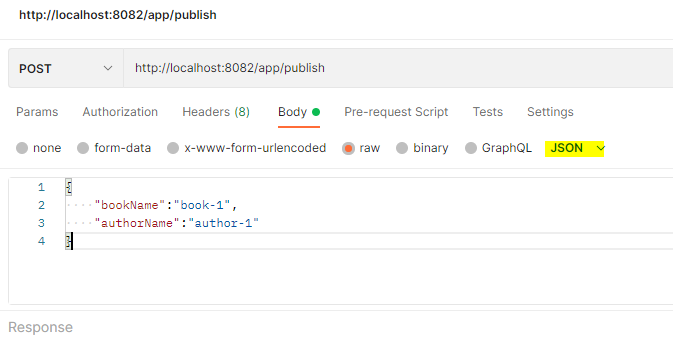




**Queue binding –**



http://localhost:8082/app/publish



{

    "bookName":"book-1",

    "authorName":"author-1"

}

……………………………………………………………………………………………..

{

    "bookName":"Test-deadLetter",

    "authorName":"author-1"

}

Dead letter exchange ( DLX )

Dead letter routing Key (DLK)

