International Journal of Research Studies in Computer Science and Engineering (IJRSCSE)
Volume 6, Issue 2, 2019, PP 24-28
ISSN 2349-4840 (Print) & ISSN 2349-4859 (Online)
DOI: http://dx.doi.org/10.20431/2349-4859.0602004
www.arcjournals.org

Distributing Messages Using Rabbitmq with Advanced Message Exchanges

Madhu M P[1], Sunanda Dixit[2*]

[1]*M. Tech Student Department of Information Science and Engineering Dayananda Sagar College of Engineering, Bangalore, INDIA*

[2]*Associate Professor Department of Information Science and Engineering, Dayananda Sagar College of Engineering, Bangalore, INDIA*

Corresponding Author:** ***Sunanda Dixit, *Associate Professor Department of Information Science and Engineering, Dayananda Sagar College of Engineering, Bangalore, INDIA*

Abstract: *RabbitMQ is an open-source message-broker software that originally implemented the Advanced Message Queuing Protocol (AMQP) and has since been extended with a plug-in architecture to support Streaming Text Oriented Messaging Protocol (STOMP), Message Queuing Telemetry Transport (MQTT), and other protocols.*

The RabbitMQ server program is written in the Erlang programming language and is built on the Open Telecom Platform framework for clustering and failover. Client libraries to interface with the broker are available for all major programming languages. In this paper we have focused on AMQP protocol and different types of exchanges used by RabbitMQ.

Keywords: *AMQP, Direct exchange, Fanout exchange, Topic exchange.*

1. INTRODUCTION

RabbitMQ is an open-source messaging system that allows to integrate applications together by using messages and queues. RabbitMQ implements the AMQP. The underlying RabbitMQ server is written in the Erlang programming language which was originally designed for the telecoms industry by Ericsson. Erlang supports distributed, fault-tolerant applications and is, therefore, an ideal language to use to build a message queuing system.

While RabbitMQ is built with Erlang, it also supports many other development platforms via client libraries. Because of Erlang's heritage of working with telecoms networks, it is the ideal platform for handling messages for enterprise applications.

The RabbitMQ server is a message broker that acts as the message coordinator for the applications that we want to integrate. This means that it can give the systems a common platform for sending and receiving messages.

Rabbit MQ contains many features to make your systems integration as painless as possible. These include:

- **Reliability**: With RabbitMQ being built on top of Erlang, the message broker is already built on top of solid, high-performance, reliable, and durable foundations. Messages can be persisted to disk to guard from lost messages in the event that a server is restarted, and we can send message delivery acknowledgements to a sender so they can be sure that the message has been received and stored.
- **Routing**: RabbitMQ works by passing your messages through exchanges before they are stored in a queue. There are different exchange types which let you perform routing, but we can also work with more complex routing scenarios by binding exchanges together.
- **Clustering and High Availability**: To increase the reliability and availability of RabbitMQ, It can cluster several servers together on a local network which forms a single logical message broker. Queues can also be mirrored across multiple servers in a cluster so that, in the advent of a server failure, you will not lose any messages.

- **Management Web User Interface (UI)**: RabbitMQ comes with a browser-based UI that lets us to manage users and their permissions, exchanges, queues, and more. This tool is an excellent window into your RabbitMQ servers.
- **Command-Line Interface**: In addition to the Management UI, there is a command-line tool called "rabbitmqctrl" and "rabbitmqadmin." These command-line tools offer the same level of administration as the web UI, with the added advantage that allows us to incorporate RabbitMQ administration in scripts.
- A major benefit of using RabbitMQ is the fact that it is a cross-platform system. It can run Erlang and the RabbitMQ server on various platforms including Windows Servers, Linux and Unix systems such as (Debian/Ubuntu, Fedora, and Solaris), Mac OS X, and on various cloud platforms such as Amazon EC2 and Microsoft Azure. Rabbit MQ also has client libraries that support many different programming environments such as Microsoft .NET, Java, Erlang, Ruby, Python, PHP, Perl, C/C++, Node.JS, Lisp, and more.
- This cross-platform nature of RabbitMQ means we could have clients written in different programming languages easily sending and receiving messages from a RabbitMQ server hosted in any of the supported environments.
- RabbitMQ is built to the AMQP Version 0-9-1. AMQP is a networking protocol that enables client applications to communicate with messaging middleware broker servers. In our case, the client will be our .NET application and the server is the RabbitMQ server or cluster of servers.

Udhayashankar.S et al**.** [1] have presented the message passing interface chatting and file transmission.

Philippe Dobbelaere et al. [2] compared Kafka and RabbitMQ which are open source and publish subscribe system. They have also framed the arguments for establishing comparison framework based on the core functionalities of publish subscribe systems.

Paolo Bellavista et al.[5] have presented the publish and subscribe system for academic and industrial interest . They have presented the ability of publish/subscribe infrastructures to offer cost-effective, scalable, and quality-aware data distribution in emerging wide-scale and highly dynamic communication environments. IoT algorithms and technologies have been summarised [8].

2. ADVANCED MESSAGE QUEUING PROTOCOL

AMQP [3] (Advanced Message Queuing Protocol) is a messaging protocol that enables conforming client applications to communicate with conforming messaging middleware brokers.

The AMQP 0-9-1 Model has the following view of the world: messages are published to exchanges, which are often compared to post offices or mailboxes. Exchanges then distribute message copies to queues using rules called bindings. Then AMQP brokers either deliver messages to consumers subscribed to queues, or consumers fetch/pull messages from queues on demand.

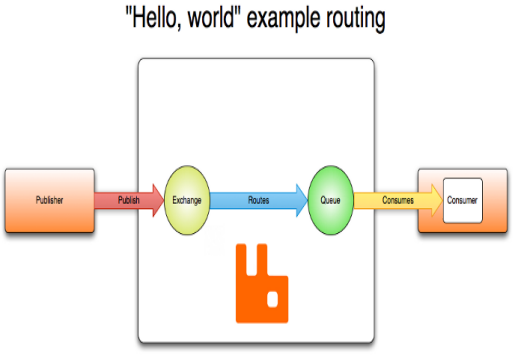

Fig1. *Routing message example*

When publishing a message, publishers may specify various message attributes. Some of this meta-data may be used by the broker, however, the rest of it is completely opaque to the broker and is only used by applications that receive the message.

Networks are unreliable, and applications may fail to process messages therefore the AMQP model has a notion of message acknowledgements: when a message is delivered to a consumer the consumer notifies the broker, either automatically or as soon as the application developer chooses

to do so. When message acknowledgements are in use, a broker will only completely remove a message from a queue when it receives a notification for that message (or group of messages).

In certain situations, when a message cannot be routed, messages may be returned to publishers, dropped, or, if the broker implements an extension, placed into a so-called "dead letter queue". Publishers choose how to handle situations like this by publishing messages using certain parameters.

Queues, exchanges and bindings are collectively referred to as AMQP entities.

3. DIFFERENT TYPE OF EXCHANGES

3.1. Default Exchange

The default exchange[4] 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.

Example, when if we 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. 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.

3.2. 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. Here is how it works:

- A queue binds to the exchange with a routing key K
- When a new message with routing key R arrives at the direct exchange, the exchange routes it to the queue if K = R Direct exchanges are often used to distribute tasks between multiple workers in a round robin manner.

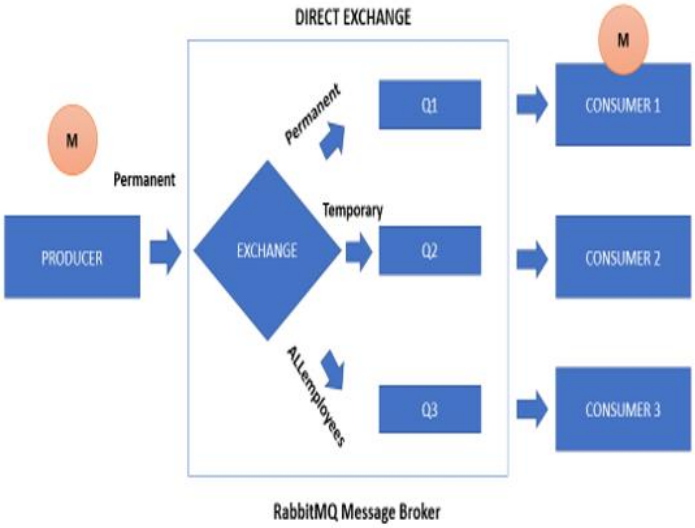

Fig2. *Direct Exchange*

In this figure direct exchange with three queues bound to it. The first queue is bound with binding key Permanent, the second with Temporary and the third with the binding key ALLemployees. A message published to the exchange with a routing key Permanent will be routed to queue Q1. Messages with a routing key of Temporary will be routed to Q2

3.3. Fanout Exchange

A fanout exchange [7] routes messages to all 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.

- Because a fanout exchange delivers a copy of a message to every queue bound to it, its use cases are quite similar:
- Massively multi-player online (MMO) games can use it for leader board updates or other global events
- Sport news sites can use fanout exchanges for distributing score updates to mobile clients in near real-time
- Distributed systems can broadcast various state and configuration updates
- Group chats can distribute messages between participants using a fanout exchange.

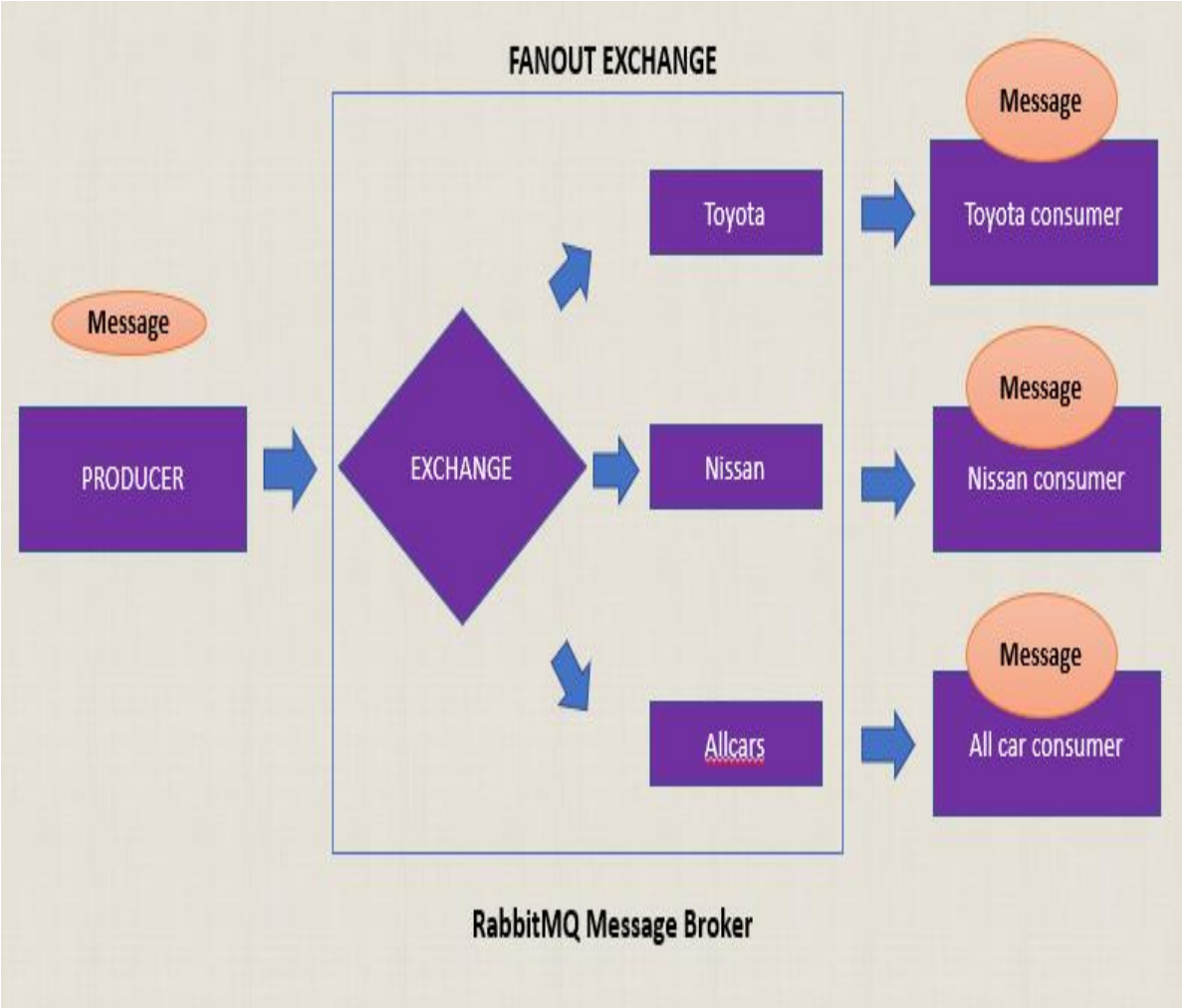

Fig3. *Fanout Exchange*

3.4. Topic Exchange

Messages sent to a topic exchange can't have an arbitrary routing key - it must be a list of words, delimited by dots. The words can be anything, but usually they specify some features connected to the message. A few valid routing key examples: "stock.usd.nyse","nyse.vmw","quick.orange.rabbit". There can be as many words in the routing key as you like, up to the limit of 255 bytes.

The binding key must also be in the same form. The logic behind the topic exchange is similar to a direct one - a message sent with a particular routing key will be delivered to all the queues that are bound with a matching binding key. However, there are two important special cases for binding keys:

- * (star) can substitute for exactly one word.
- # (hash) can substitute for zero or more words.

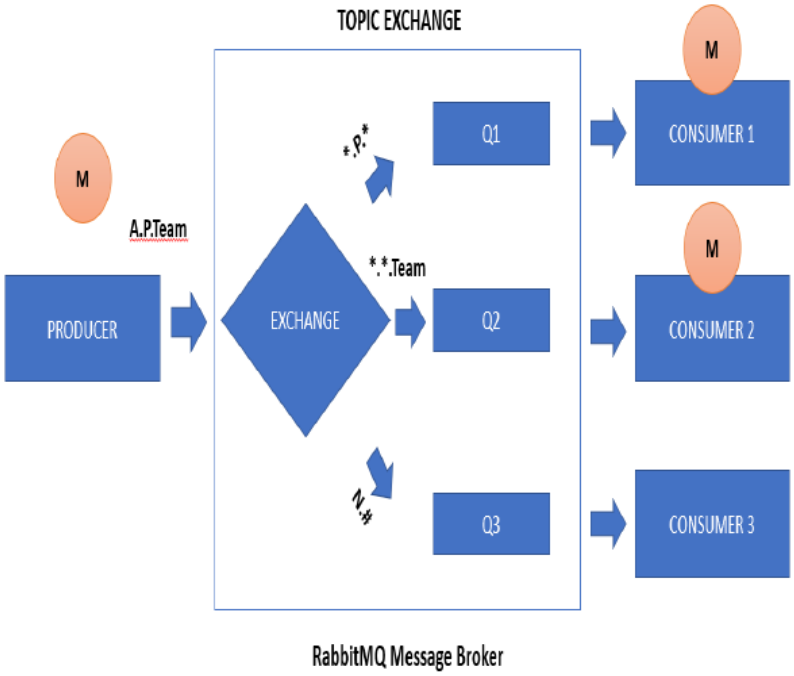

Fig4. *Topic Exchange*

4. CONCLUSION

RabbitMQ mainly acts as message broker for sending message. It is mainly useful for distributing the message and guaranteed delivery of the message as it will use AMQP protocol.

The user can select any one of the exchanges based on preference whether he wants to send a message to single subscriber or multiple subscriber. AMQP and different types RabbitMQ exchanges have been summarized.

REFERENCES

[1] Udhayashankar.S, Dr. K.P. Kaliyamurthie, Mathivilasini.S , "A Novel Security Technique in Message Passing Interface Systems"International Journal of Computer Science and Mobile Computing,2013.

[2] Philippe Dobbelaere, Kyumars Sheykh Esmaili, "Industry Paper: Kafka versus RabbitMQ",2017.

[3] AMQP,"https://www.cloudamqp.com/docs/amqp.html".

[4] Defaultexchange,"https://www.rabbitmq.com/tutorials/tutorial-four-python.html".

[5] Paolo Bellavista, Antonio Corradi, Andrea Reale, "Quality of Service in Wide Scale Publish–Subscribe Systems" IEEE Communications Surveys & Tutorials,2014.

[6] TopicExchange,"https://springbootdev.com/2017/11/12/spring-amqp-rabbitmq-topic-exchange-example-part-1-producer-application/".

[7] Fanoutexchange,"https://www.rabbitmq.com/tutorials/tutorial-three-python.html".

[8] Alvaro Videla ,Jason J.W. Williams "RabbitMQ in Action",2012

[9] Karimunnisa Begum, Sunanda Dixit, "Industrial WSN using IoT: A Survey", International Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT) – 2016

Citation: *Sunanda Dixit & Madhu M P, (2019). Distributing Messages Using Rabbitmq with Advanced Message Exchanges. International Journal of Research Studies in Computer Science and Engineering (IJRSCSE), 6(2), pp.24-28. http://dx.doi.org/10.20431/2349-4859.0602004*