RabbitMQ

Abhishek Gupta[1,*]

[1] School of Informatics and Computing, Bloomington, IN 47408, U.S.A.
[*] Corresponding authors: abhigupt@iu.edu

project-001, February 22, 2017

Back in time when we had a multi-threaded applications, IPC was the mechanism where different modules in application used to share data, messages and maintain common data structures. Today we have a distributed applications which runs modules as micro services. These modules are running in their own container or virtual machine, so how do we pass messages across these services something line IPC? We need a mechanism similar to IPC, which can communicate to all other services. Also, it can be used by other services to send and receive messages. If you compare with IPC mechanism, we need something very similar. Here we cannot use IPC for obvious reasons. When we look for messaging, we look for certain features: asynchronous messaging, large scale, reliability, clustering, multi-protocol, highly available, fault tolerant. RabbitMQ[1] fulfills these requirements and provide a distributed, persistent, highly available, fault tolerant messaging system which can scale as data grows.

© 2017 https://creativecommons.org/licenses/. The authors verify that the text is not plagiarized.

Keywords: Cloud, I524

https://github.com/cloudmesh/sp17-i524/blob/master/paper1/S17-IO-3005/report.pdf

1. INTRODUCTION

RabbitMQ based[1] off AMQP (Advanced Message Queuing Protocol[2]) Which defines how client applications connect to message brokers. Message brokers receive messages from producers and make it available to consumers. The producer posts messages to exchanges and broker agent reads these messages from exchange and writes to queues. Consumers can further read messages from the queue. It also supports a notion of acknowledgements where consumers can post an acknowledgement back to broker and which in turn can post messages back to the producer.

Looking back at the history[3] of development of rabbitMQ, it started with IBM MQseries in 1993, 1997 Microsoft MQ, 2001 java messaging service. It was in 2003 where JPMorgan created the first version AMQP which became the base for RabbitMQ technologies formed in 2006.

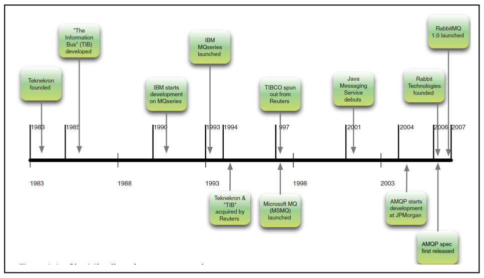

"The Information Bus" (TIB) developed
IBM MQseries launched
RabbitMQ 1.0 launched
Teknekron founded
IBM starts development on MQseries
TIBCO spun out from Reuters
Java Messaging Service debuts
Rabbit Technologies founded
1983
1985
1990
1993
1994
1997
2001
2004
2006
2007
1983
1988
1993
1998
2003
Teknekron & "TIB" acquired by Reuters
Microsoft MQ (MSMQ) launched
AMQP starts development at JPMorgan
AMQP spec first released

Fig. 1. Timeline for evolution of RabbitMQ [3]

2. ARCHITECTURE

[3]At high level, producer publishes the message to the broker. Consumer consumes or subscribes the messages from the broker. Broker is the middle man who has the knowledge of exchanges and queues. It maintains the bindings between exchanges and queues. The consumers read the messages from the queue. Following table shows different exchanges supported by RabbitMQ and corresponding default exchanges[1] .

Name	Default pre-declared names
Direct exchange	(Empty string) and amq.direct
Fanout exchange	amq.fanout
Topic exchange	amq.topic
Headers exchange	amq.match (and amq.headers in RabbitMQ)

3. TYPE OF EXCHANGES

The sections below explains various types of exchanges supported by RabbitMQ.

3.1. Default Exchange

It is created by the broker and all queues are bound to default exchange unless specified separately. For example we have a queue called demoqueue, all messages assigned to demoqueue will be routed by default exchange.

3.2. Direct exchange

It is used to route messages to queue with a given routing key. For example a message queue has routing key K. A message with same routing key K will be delivered to same message queue.

3.3. Fanout exchange

Delivers messages to all queues that are bound to a given exchange ignoring the routing key. For example if there are 5 queues bound to a exchange E. Messages to exchange E are delivered to all 5 queues.

3.4. Topic exchange

It delivers the messages to a given queue, not just based on the routing key but a pattern specified in the message called topic. It can be useful in scenario when consumer/application decides to which topic(s) it is interested in. Further, it can subscribe to those topics(or messages).

3.5. Header exchange

This exchange ignores the routing key but uses the header parameter to decide which messages goes to which queue. A message is matched when a value in header matches with the one specified in the queue binding. Exchanges have other important attributes apart from routing key and exchange type for example name, durability, auto-delete, arguments etc. Durability allow messages to persist on disk in case of broker restarts. Auto-delete deletes the exchange, once all queue have finished using the exchange.

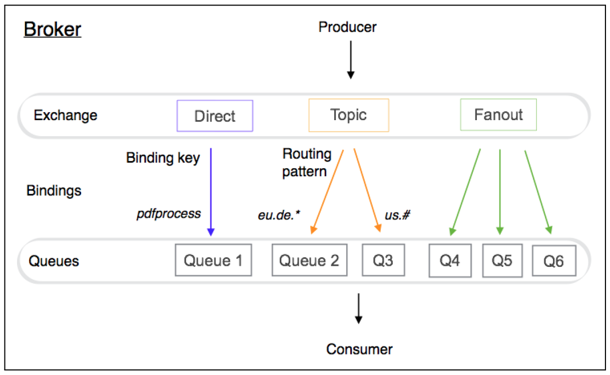


Fig. 2. Different type of exchanges supported by RabbitMQ [4]

4. BINDINGS

Bindings are the rules that define how exchanges will route messages to the queue. To route all messages from exchange E to queue Q, Q has to be bound to Exchange E. Bindings use routing key as one of the criteria to route messages to a queue. However, routing key is optional and not always applicable.

5. CONSUMERS

Messages from message queue are eventually used or consumed by consumers. Consumers can use push or pull mechanism to consume these messages. Push API have messages delivered to the consumer whereas a Pull API is used to fetch messages from the queue.

6. MESSAGE ACKNOWLEDGEMENT

It has a built-in mechanism to send and receive acknowledgements. Producer can send messages and wait to acknowledgement in the response queue from consumer. Consumer can receive messages and post acknowledgement to response queue. Here request queue and request exchange can be used to send and receive messages. Whereas response queue and response exchange can be used to send and receive acknowledgements. This mechanism makes RabbitMQ robust in case of failures.

7. CLUSTERING

[3]To achieve high availability and making sure producers and consumers send and receive data without knowing about node failures, clustering was introduced. RabbitMQ follows OTP(open telecom platform) framework provided by erlang to achieve high availability. RabbitMQ by default doesn't replicate the content of queues i.e. all queues are stored on one node in the cluster. To achieve clustering rabbitmq keeps track of metadata for queue, exchange, binding and vhost. In case of cluster, it only stores all information about the queue like metadata, state and contents on one node rather than all nodes in the cluster. However, it stores metadata and pointer to actual data on each node in the cluster. This is to optimize storage space and performance.

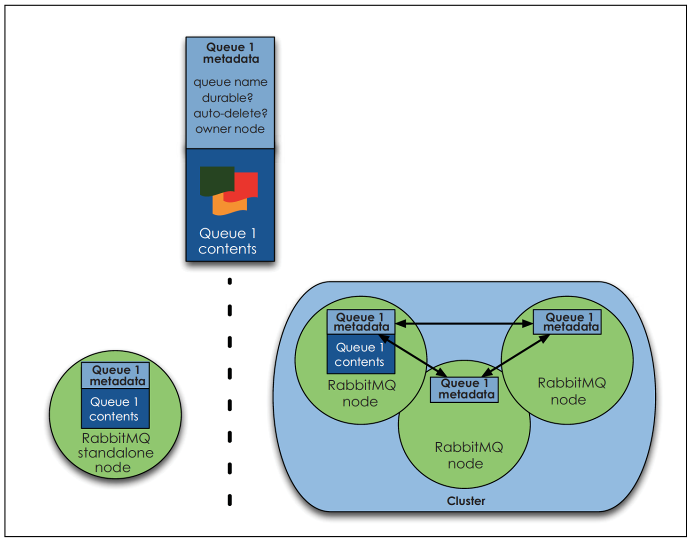


Fig. 3. Shows queue metadata for a cluster vs single node [3]

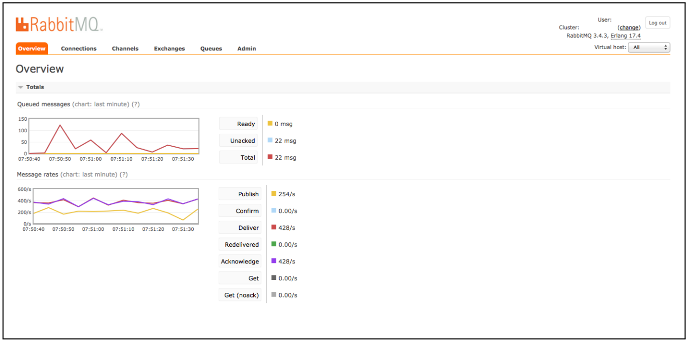
On the other hand exchanges are just bindings to the queues. So when you send a message to rabbitmq exchange, the channel checks the routing key of the message and compares it against the queue bindings. Further it sends the message to appropriate queue. Since, exchanges are just lookup tables for queue bindings, they are replicated across all nodes on the cluster.

8. MANAGEMENT

RabbitMQ provides all management using:

- web ui
- REST interface
- Rabbitmqctl command line utility

Web UI can be used by administrators to create user, monitor queues and exchanges, view statistics, add configurations etc. Similar functionality can be achieved by cli utility rabbitmqctl and REST API. Rabbitmqctl can be used to automate rabbitmq deployments and management. IT can also be used to write automated tests. REST Api can used for integrating with 3rd part UI and plugins. Using REST api you can monitor the number of connections, download or upload a configuration, list the nodes in the cluster, create or delete rabbitmq users, view or create virtual host, set permission for a user etc. You can also list all current APIs using following url: http://localhost:55672/api with some api documentation and explanations.



Ready	🟨 0 msg
Unacked	🟦 22 msg
Total	🟥 22 msg

Publish	🟨 254/s
Confirm	🟦 0.00/s
Deliver	🟥 428/s
Redelivered	🟩 0.00/s
Acknowledge	🟪 428/s
Get	⬛ 0.00/s
Get (noack)	⬜ 0.00/s

Fig. 4. RabbitMQ management UI showing messages queued and message data rates
[4]

9. LICENSING

RabbitMQ is licensed under Mozilla Public License(MPL) and GPL v2. [1]

10. USE CASES

RabbitMQ messaging can be useful in applications which require asynchronous messaging for example an application initiates a task by posting a message to RabbitMQ, it doesn't have to wait for the task to get completed. Rather it can periodically check the status of task. It can be useful to scale up as the data volume grows by adding additional nodes to RabbitMQ cluster. With distributed applications where applications run as micro-services in a container or virtual machine, RabbitMQ is useful to communicate and share data between different services. RabbitMQ can be managed separately with its management UI, CLI tool and REST api interface. This decouples the messaging layer from application and makes the overall design very robust.

11. CONCLUSION

RabbitMQ is an open source platform and provides a robust messaging platform for applications. It provides simple manageability decoupling with the application. RabbitMQ can scale well when application demand increases and can handle more data by adding more nodes to the cluster. Based on test [5]conducted on RabbitMQ with set of 4K(4096) and 16K(16384) messages, the performance of RabbitMQ on multi node cluster decreases in comparison with single node cluster to reach a threshold and then becomes stable. This decrease in performance can be primarily accounted due to replication between nodes in the cluster.These tests were conducted on combination of single publisher and single subscriber, multiple publishers and single subscriber, multiple publishers and multiple subscribers but there is no concrete conclusion to these test and numbers.

ACKNOWLEDGEMENTS

Special thanks to Professor Gregor von Laszewski, Dimitar Nikolov and all associate instructors for all help and guidance related to latex and bibtex, scripts for building the project, quick and timely resolution to any technical issues faced. The paper is written during the course I524: Big Data and Open Source Software Projects, Spring 2017 at Indiana University Bloomington.

REFERENCES

[1] Pivotal, "RabbitMQ, components," Web Page, accessed: 2017-02-02. [Online]. Available: https://www.rabbitmq.com/
[2] S. Vinoski, "Advanced message queuing protocol," *IEEE Internet Computing*, vol. 10, no. 6, 2006. [Online]. Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=4012603
[3] A. Videla and J. J. Williams, *RabbitMQ in action*. Manning, 2012.
[4] Lovisa Johansson, "Rabbitmq for beginners - what is rabbitmq?" WEb Page, May 2015, accessed: 02-15-2017. [Online]. Available: https://www.cloudamqp.com/blog/2015-05-18-part1-rabbitmq-for-beginners-what-is-rabbitmq.html
[5] B. Jones, S. Luxenberg, D. McGrath, P. Trampert, and J. Weldon, "Rabbitmq performance and scalability analysis," *project on CS*, vol. 4284, 2011. [Online]. Available: https://people.cs.vt.edu/butta/cs4284/spring2011/butta/RabbitMQPaper.pdf