**¿Qué es RabbitMQ?**

RabbitMQ es un broker de mensajería de código abierto, distribuido y escalable, que sirve como intermediario para la comunicación eficiente entre productores y consumidores.

RabbitMQ implementa el protocolo mensajería de capa de aplicación AMQP (Advanced Message Queueing Protocol), el cual está enfocado en la comunicación de mensajes asíncronos con garantía de entrega, a través de confirmaciones de recepción de mensajes desde el broker al productor y desde los consumidores al broker.

En una forma simplificada, **en RabbitMQ se definen colas que van a almacenar los mensajes que envían los productores hasta que las aplicaciones consumidoras obtienen el mensaje y lo procesan.** Esto nos permite diseñar e implementar sistemas distribuidos, en los cuales un sistema se divide en módulos independientes que se comunican entre sí a través de mensajes.

Podemos decir entonces que **RabbitMQ en su rol de intermediario, tiene como labor asegurarse de que los mensajes que un productor envía, se enruten al consumidor correcto.**

**¿Qué componentes podemos encontrar en RabbitMQ?**

**1. Exchange**

Este componente es el encargado de recibir los mensajes enviados al broker por un productor y depositarlos en la cola adecuada de acuerdo a una llave de enrutamiento (routing key). Esto significa que **el productor no envía los mensajes directamente a la cola, sino que los envía a un exchange con una llave de enrutamiento.**

De esta forma, si un productor quiere enviar un mensaje a diversas colas, no tiene que enviarlo a cada una de ellas, sino que el exchange se encarga de distribuir este mensaje a cada una de las colas. Conozcamos ahora los diferentes tipos de exchange: d**irect, topic y fanout.**

* El exchange direct: toma la llave de enrutamiento que viene en el mensaje y lo lleva a la cola que está asociada a este exchange y a esta llave de enrutamiento.
* El exchange topic: lleva el mensaje a las colas que cumplan con un patrón en la llave de enrutamiento. Por ejemplo, si tenemos la cola Q1 asociada al exchange EXCH1 con la llave de enrutamiento **solicitud.credito.consumo** y la cola Q2 asociada al mismo exchange con la llave de enrutamiento **solicitud.credito.libranza**, un mensaje que es enviado al exchange EXCH1 con llave de enrutamiento **solicitud.credito.\*** será enviado a ambas colas.
* El exchange fanout: envía el mensaje a todas las colas asociadas con el exchange, sin importar la llave de enrutamiento.

**2. Routing Key**

Es la llave que utiliza el exchange para saber a donde enrutar un mensaje, y a su vez es la misma que usa la cola para asociarse con un exchange.

**3. Cola**

Es el componente que guarda los mensajes provenientes de los exchange y los envía a los consumidores que están escuchando por estos mensajes.

**4. Binding**

Es la asociación entre una cola y un exchange a través de una llave de enrutamiento.

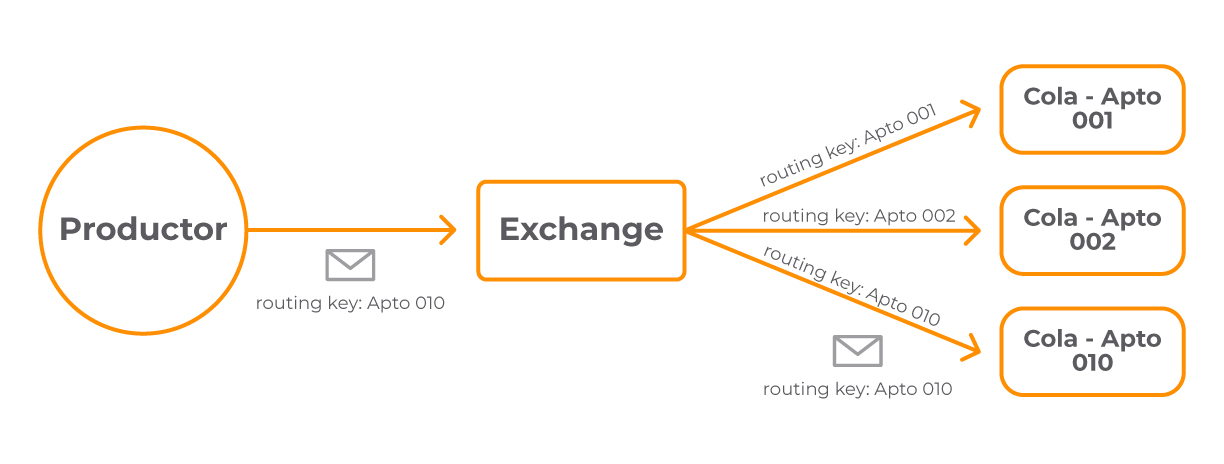
**5. Virtual Host**

División lógica de los componentes de servidor de RabbitMQ en la cual estos se agrupan para simplificar su administración y control. Debemos tener en cuenta que no pueden existir colas con el mismo nombre en un mismo virtual host, pero sí en diferentes virtual host.

Ahora veamos un ejemplo  de la cotidianidad para entender estos conceptos. Imaginemos los casilleros donde suelen dejar la correspondencia de los apartamentos de un edificio. Cada uno de los casilleros tiene una marcación con el número del apartamento en la que el portero del edificio deposita las encomiendas cuando alguien se las entrega. Para nuestro ejemplo **los casilleros representan las colas, el número del apartamento es la llave de enrutamiento, el portero hace las veces de exchange y la persona que entrega la encomienda es el productor**.

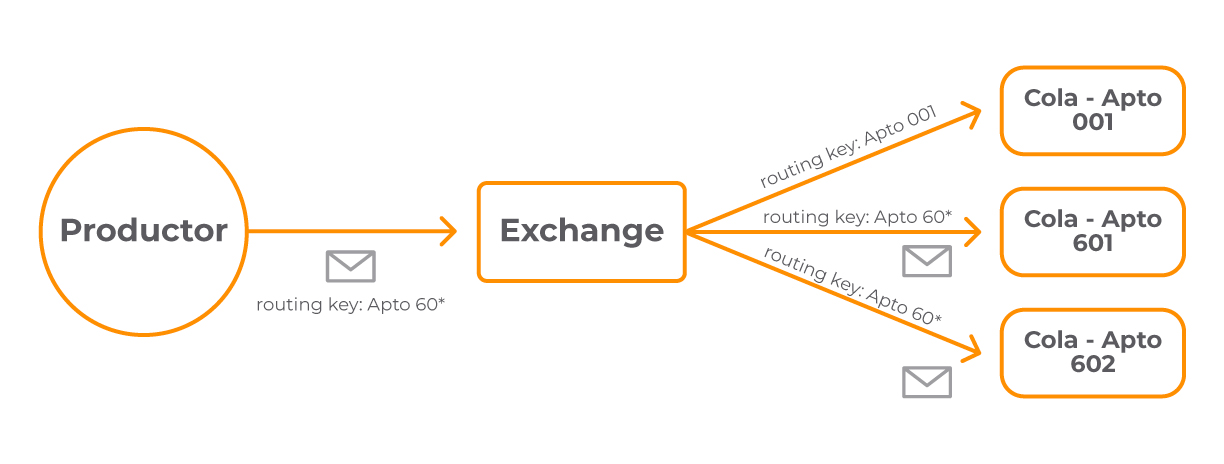
Teniendo en cuenta el ejemplo, un exchange directo se da cuando una encomienda va para el apartamento 010, el productor se la entrega al portero (exchange directo), quien revisa la etiqueta de la encomienda para establecer a qué apartamento va destinado y depositarlo en su respectivo casillero.

**En la siguiente imagen se presenta el envío de un mensaje a un exchange directo:**



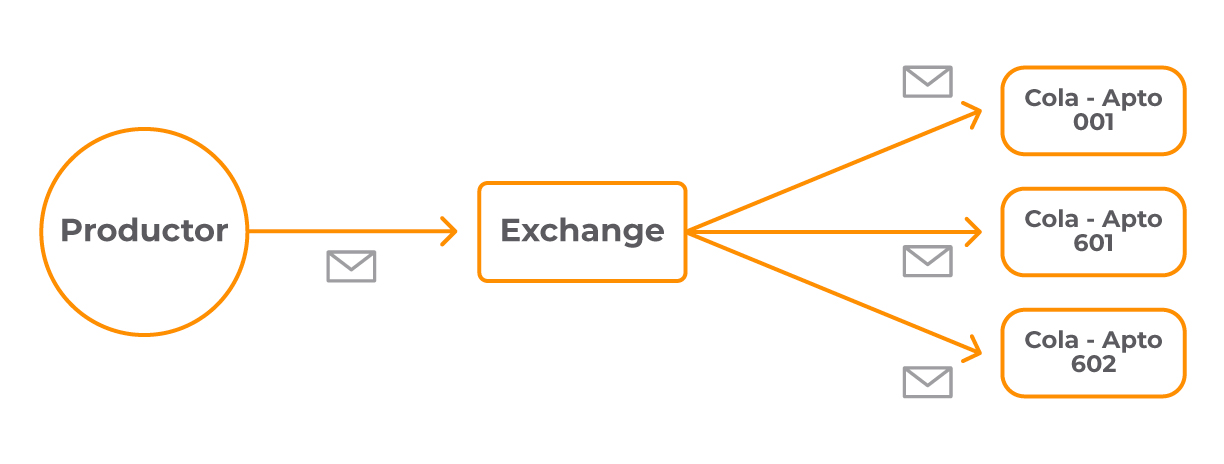
Por otro lado, un exchange de tipo topic se da cuando el administrador del edificio desea enviar una carta solo a los apartamentos del último piso del edificio (601,602), el portero (exchange) recibe las cartas y las depositas en los casilleros (cola) de los apartamentos del último piso (601, 602).

##### En la siguiente imagen se presenta el envío de un mensaje a un exchange de tipo topic:



Un exchange de tipo fanout, se da cuando el administrador del edificio envia un comunicado a todos los apartamentos. En este caso el portero (exchange) deposita el mensaje en cada casillero sin preocuparse en revisar el número del apartamento, sino en que cada casillero reciba una copia del mensaje.

##### En la siguiente imagen se presenta el envío de un mensaje a un exchange de tipo fanout.



## Beneficios de utilizar RabbitMQ

Además de permitir la **integración de diferentes aplicaciones a través de mensajes de forma asíncrona (desacoplamiento en tiempo) y desde diversas ubicaciones (desacoplamiento en espacio)**, RabbitMQ nos ofrece otros beneficios que lo han hecho muy popular dentro del mundo de los brokers de mensajería:

#### 1. Confiabilidad

RabbitMQ incorpora varias características que le permiten garantizar la entrega de los mensajes. Entre ellas, proporciona almacenamiento cuando no hay consumidores disponibles para recibir el mensaje, **brinda la posibilidad de que el consumidor acepte la entrega del mensaje para asegurarse de que lo procesó correctamente** y, en caso de que haya fallado su procesamiento, permite que el mensaje se pueda reencolar para ser consumido por una instancia diferente del consumidor o que  sea procesado de nuevo por el mismo consumidor que inicialmente falló, cuando este se recupere.

RabbitMQ también garantiza el orden de entrega de los mensajes, es decir,  estos se van consumiendo en el mismo orden en que han estado llegando a las colas de RabbitMQ.

#### 2. Creación de clusters

Si bien RabbitMQ proporciona gran rendimiento procesando miles de mensajes por segundo, en ocasiones debe ser capaz de procesar una mayor cantidad de mensajes sin impactar el rendimiento de las a aplicaciones. Para esto RabbitMQ **permite la creación de clústeres para escalar horizontalmente la solución**, lo cual es transparente tanto para los productores como para los consumidores.

#### 3. Colas altamente disponibles

En RabbitMQ las colas pueden ser replicadas en diversos nodos de un cluster, proporcionando la seguridad de que en caso de una falla o indisposición de un nodo, el broker puede seguir recibiendo mensajes de los productores y entregandolos a los consumidores adecuados.

#### 4. Permite la escalabilidad de las aplicaciones

Cuando un consumidor se subscribe a una cola, y existen mensajes para este consumidor, RabbitMQ le va entregando mensajes para su procesamiento . Si la velocidad de producción de mensajes es mayor a la capacidad que el consumidor puede procesar dichos mensajes, se pueden crear nuevas instancia de ese consumidor para hacer frente al mayor flujo de mensajes.

Cuando RabbitMQ identifica que existen varias instancias de un mismo consumidor subscritas a una misma cola, **balancea la entrega de mensajes a cada una de las instancias**. Esta característica no solo nos posibilita la distribución de carga en las aplicaciones consumidoras, también nos permite incrementar la disponibilidad de las mismas.

#### 5. Enrutamiento flexible

En RabbitMQ se pueden definir reglas de enrutamiento flexible, incluso que cumplan un determinado patrón, para enrutar los mensajes entre los exchanges y las colas, a través de los bindings.

#### 6. Soporte a múltiples protocolos

Aparte de soportar el protocolo AMQP, RabbitMQ también soporta STOMP, MQTT y HTTP a través de plugins.

#### 7. Mecanismos de autenticación

Incorpora mecanismos de autenticación y control de acceso a cada uno de los componentes del broker.

#### 8. Soporte de lenguajes

RabbitMQ soporta una gran cantidad de lenguajes de programación con los que es posible construir productores y consumidores de mensajes. Entre estos tenemos Java, Scala, PHP, Python, Ruby, entre otros.

Los brokers de mensajería han sido tradicionalmente una pieza importante dentro de las arquitecturas de las organizaciones, pero con el incremento de usuarios que ingresan por diferentes canales a los sistemas de las empresas, se hace necesarios productos que permitan ser escalados horizontalmente a un bajo costo, permitiendo el procesamiento de una gran cantidad de mensajes por segundo.

Es acá donde una nueva generación de brokers, como RabbitMQ, están tomando mayor relevancia en las aplicaciones modernas, que **buscan ofrecer altos niveles de disponibilidad, confiabilidad, interoperabilidad, y rendimiento a nuestros clientes.**

RabbitMQ es uno de esos productos que las organizaciones deberían evaluar a la hora de tomar la decisión sobre el software que muy posiblemente va a soportar la estrategia de modernización digital en la que varias organizaciones se han venido embarcando en los últimos tiempos.

En el sitio de [RabbitMQ](https://www.rabbitmq.com/" \t "_blank) podrás conocer mucho más sobre este broker de mensajería.