

# RabbitMQ

*Introducción a RabbitMQ*



Manuel Vega

2017

# RabbitMQ

## *Introducción a RabbitMQ*

### RabbitMQ In Action

La primera pregunta a resolver: ¿Para qué necesitamos un sistema de encolado de mensajes?, ¿para qué emplearíamos un sistema de encolado de mensajes?

#### **¿Qué es un MQ?**

En sus siglas inglesas MQ hace referencia a message queue (cola de mensajes). Es un patrón de diseño de sistemas que se comenzó a utilizar a principio de los 80. Se compone de un corredor (broker) encargado de acumular los mensajes provenientes de los productores (producers). Estos mensajes se almacenan en una serie de colas. A estas colas están suscritos los consumidores (consumers) que obtienen los mensajes al ritmo que los necesitan.

El inicio de este tipo de sistemas fue en entornos bancarios. La idea es que los productores generen mensajes, operaciones a realizar dentro del sistema. Según la cola donde se almacena el mensaje, llegará a uno o varios consumidores que podrán realizar una acción concreta. La ventaja de este sistema en contraposición al método típico de la función (o procedimiento) es su naturaleza asíncrona, el control de la cantidad de operaciones a realizar y la posibilidad de paralelizar acciones. Esto es gracias a que el mensaje puede llegar a dos consumidores diferentes encargados de realizar acciones independientes.

#### **Problemática a resolver**

A través de los últimos años el uso de Internet viene marcado por siglas como SaaS, PaaS, IaaS, ... todas ellas indicando una misma tendencia: la venta de servicios en lugar de productos. Esto acarrea una posición ventajosa para los desarrolladores de software pudiendo dar a sus clientes siempre la última versión del software mantenido en un único lugar y su pago reducido (en comparación al del producto) a través de una cuota periódica.

Pero también acarrea el problema de tener ejecutar el software en una misma infraestructura servidora para todos los clientes que desean utilizarlo al mismo tiempo.

En sistemas síncronos de cliente-servidor donde las peticiones son enviadas a un mismo punto y el cliente espera puede producirse el efecto avalancha. Recibir en un momento dado demasiadas peticiones para las cuales el sistema no está suficientemente bien dimensionado.

### **Respuestas asíncronas y encolado**

La mejor solución ante este problema es dividir y acumular (o encolar). Ante la avalancha de peticiones el sistema puede atender rápidamente todas las llamadas respondiendo valores temporales. Empleando nuevas llamadas vía AJAX o websocket el servidor puede responder una vez se haya completado la tarea. Esto da la posibilidad al servidor intermediario de presentar otro nivel de información acerca de la espera, o incluso dejar libre al cliente para que pueda realizar más acciones.

El servidor sería en este caso un productor. Cada petición entrante del cliente la convierte en un mensaje obteniendo toda la información posible y la envía al sistema de encolado. En otro nivel y dependiendo de los recursos necesarios para resolver estas tareas, se encuentran los consumidores. Estos se encargan de tomar mensajes del broker y procesarlos.

La diferencia principal es la posibilidad de acumular todos los estados en momentos de alta carga y procesarlos poco a poco. En lugar de obtener una caída por denegación de servicio, se obtendría una demora en el procesamiento de mensajes. Este es el motivo por el que se comenzó empleando en entornos bancarios donde la velocidad no es tan importante como el hecho de realizar la operación de forma fiable.

### **Administración de MQ**

Además de exponer los entornos y su uso, el libro se orienta también para los administradores de sistemas. Muestra como administrar RabbitMQ, configurar Nagios, controlar las colas y su correcto funcionamiento.

Cubre también la configuración de RabbitMQ no solo en un centro de datos (CPD) sino también cuando se quiere o necesita configurar en varias regiones geográficas. Posibles fallos, formas de replicación y funcionamiento.

## Gestiona tus propios brokers de mensajería

RabbitMQ es un software de mensajería desarrollado por VMware (SpringSource) y que cuenta con el soporte comercial de Pivotal. Gracias a su fiabilidad, facilidad de uso y disponibilidad para múltiples Sistemas Operativos, RabbitMQ supone una seria alternativa opensource al SW bajo licencias de pago (p.e: MQ series de IBM) que habitualmente representan serios quebraderos de cabeza a la hora de ajustar los presupuestos de un proyecto. Algunas de las principales características de RabbitMQ, cuya última release es la versión 3.4.3, son:

- Soporta múltiples protocolos de mensajería (directamente o mediante plugins programados), tales como AMQP 0-9-1, AMQP 1-0, STOMP, MQTT o HTTP.
- Existe una amplia variedad de clientes y conectores desarrollados para permitir la comunicación en una gran variedad de lenguajes tales como Java/JVM, MacOSX, Amazon Ec2, C/C++, Ruby, Python, Perl, Erlang.
- Existe una UI muy completa que permite (entre otras muchas posibilidades), administrar los brokers generados (crear colas, exchanges, usuarios etc.) o incluso consultar el tráfico de mensajes en el sistema en tiempo real.
- Permite agrupar diferentes nodos en un cluster que actúe como único broker lógico, admitiendo incluso políticas de alta disponibilidad y mirroring.
- Permite establecer distintos brokers en una misma máquina, así como generar diferentes virtual hosts en los que ubicarlos.
- Los brokers generados disponen de un sistema de logs que informan al administrador de las acciones que se están llevando a cabo sobre el mismo, facilitando la depuración de errores.

Los principales elementos que conforman un broker de RabbitMQ, son las colas, los exchanges y los enlaces (bindings) que se establecen entre ambos. Los mensajes, al llegar al sistema, son recibidos por los exchanges, que son entidades AMQP desde donde se enrutan los mensajes hacia las colas. Rabbit tiene 4 tipos de exchange predefinidos:

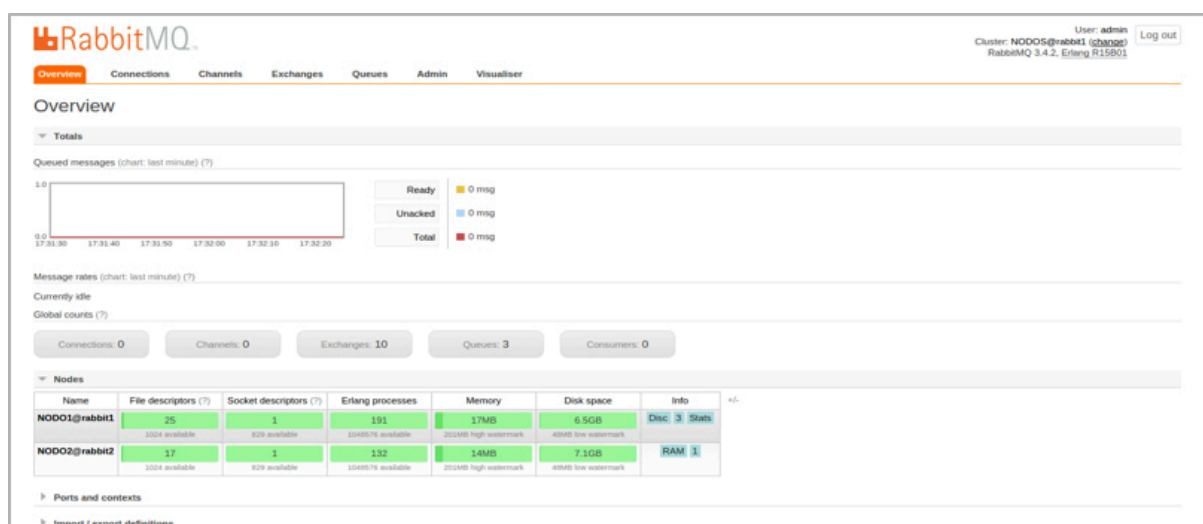
- **DIRECT:** Redirigen los mensajes solo a aquellas colas a las que coincida el routing key de mismo, con el establecido en el binding a la cola.

- **FANOUT:** Redirigen los mensajes recibidos a todas las colas enlazadas al mismo.
- **TOPIC:** Redirige aquellos mensajes cuyo routing key coincida con el patrón de enlace (por ejemplo, la primera o la última palabra del routing key).
- **HEADERS:** Enrutan los mensajes en función de la cabecera del mensaje, la cual **NO** tiene por qué ser un String, y puede representar, por ejemplo, un número entero.


Las colas son las entidades sobre las que el broker almacena los mensajes que recibe de los productores, desde donde los consumidores obtienen dichos mensajes. A la hora de configurarlas, admiten multitud de parámetros (nombre, persistencia, TTL, Exchange de reenvío por error/expiración del mensaje (Conocido como DLX), capacidad de la cola, etc.). Eso sí, una vez generadas no se pueden modificar, por lo que variar sus parámetros implica volver a crearlas.

Para establecer una comunicación con el broker, es necesario disponer de un usuario que esté registrado en el mismo. RabbitMQ permite establecer limitaciones en los usuarios que genera, bien mediante tags (administrator, policymaker, management o monitoring) o mediante cada virtual host al que este se encuentre asociado.

A continuación, y a modo de ejemplo simple, se muestran un par de capturas sobre la interfaz rabbit de un broker conformado como cluster de 2 nodos.



Broker de RabbitMQ conformado como cluster de 2 nodos



User: admin  
Cluster: NODO1@rabbit1 (change)  
RabbitMQ 3.4.2, Erlang R15B01

[Overview](#)
[Connections](#)
[Channels](#)
[Exchanges](#)
[Queues](#)
[Admin](#)
[Visualiser](#)

## Queues

All queues

Filter:  ☐ Regexp (?)
3 items (show at most 100)

Overview				Messages			Message rates		
Name	Node	Features	State	Ready	Unacked	Total	incoming	deliver / get	ack
COLA1	NODO1@rabbit1	<span>D</span> ha	idle	0	0	0			
COLA2	NODO1@rabbit1	<span>D</span> ha	idle	0	0	0			
TEST	NODO1@rabbit1	<span>D</span> TTL Exp Lim ha	idle	0	0	0			

Add a new queue

Name:

Durability: Durable

Node: NODO1@rabbit1

Auto delete: (7) No

Arguments:  String

Add: Message TTL (?) | Auto expire (?) | Max length (?) | Max length bytes (?)  
Dead letter exchange (?) | Dead letter routing key (?)

Add queue

[HTTP API](#) | [Command Line](#)

Update: every 5 seconds

Last update: 2015-01-22 17:32:07

## Descubriendo RabbitMQ: una solución para colas de mensajería

Dentro de un proyecto en ocasiones hay que integrarse con otros actores, componentes o sistemas internos y externos, surgiendo la necesidad de aportar o recibir información de ellos. En el mayor de los casos, estas comunicaciones tienen que estar permanente disponibles, ser rápidas, seguras, asíncronas y fiables entre otros requisitos.

Las colas de mensajes (MQ) solucionan estas necesidades, actuando de middleware entre emisores y destinatarios, o en un contexto más definido, productores y consumidores de mensajes. Aportan a su vez más beneficios:

- Garantía de entrega y orden: los mensajes se consumen, en el mismo orden que se llegaron a la cola, y son consumidos una única vez.
- Redundancia: Las colas persisten los mensajes hasta que son procesados por completo.
- Desacoplamiento: siendo capas intermedias de comunicación entre procesos, aportan la flexibilidad en la definición de arquitectura de cada uno de ellos de manera separada, siempre que se mantenga una interfaz común.
- Escalabilidad: con más unidades de procesamiento, las colas balancean su respectiva carga.

Existen múltiples soluciones de colas de mensajería, como ActiveMQ, ZeroMQ, OpenAmq, etc.

Nos vamos a centrar en descubrir una de ellas: RabbitMQ, un sistema de mensajería empresarial completo y altamente confiable basado en el estándar AMQP. Está licenciado bajo la licencia de código abierto Mozilla Public License y cuenta con una distribución independiente de plataformas. Debido a esto es de los más extendidos, incluso existe un módulo de puppetlabs que permite instalarlo, configurarlo y administrarlo. Sus características principales son:

- Garantía de entrega
- Enrutamiento flexible
- Clusterización
- Federación
- Alta disponibilidad
- Tolerancia a fallos



En concreto vamos a ver cómo instalar un servidor (broker) federado compuesto por dos nodos con RabbitMQ para disponer de alta disponibilidad, administrar cada uno de ellos, mecanismos de autenticación y autorización, configurándolos de moda segura mediante Two-way SSL Authentication.

## Instalación RabbitMQ

Se instala como servicio de manera sencilla, añadiendo el repositorio siguiente en `/etc/apt/sources.list` e instalándolo.

```
deb http://www.rabbitmq.com/debian/ testing main
sudo apt-get update
sudo apt-get install rabbitmq-server
service rabbitmq-server start
```

Se puede instalar también usando una imagen de docker oficial `rabbitmq:3-management` con RabbitMQ instalado y su plugin de administración activado por defecto.

Al iniciarse el servicio de RabbitMQ se puede ver la versión y los plugins iniciados se muestran al final.

```
=INFO REPORT===== 30-Dec-2015::11:33:17 =====
Starting RabbitMQ 3.5.7 on Erlang 18.1
Copyright (C) 2007-2015 Pivotal Software, Inc.
Licensed under the MPL. See http://www.rabbitmq.com/
RabbitMQ 3.5.7. Copyright (C) 2007-2015 Pivotal Software, Inc.
## ## Licensed under the MPL. See http://www.rabbitmq.com/
## ##
##### Logs: tty
##### ## tty
#####
Starting broker...
=INFO REPORT===== 30-Dec-2015::11:33:17 =====
node : rabbit@my-rabbit
home dir : /var/lib/rabbitmq
config file(s) : /etc/rabbitmq/rabbitmq.config
```

```
cookie hash : 5ZLi4SCoYuR5W/C1lYQ0yQ==
log : tty
sasl log : tty
database dir : /var/lib/rabbitmq/mnesia/rabbit@my-rabbit
=INFO REPORT===== 30-Dec-2015::11:33:17 =====
Memory limit set to 3051MB of 7629MB total.
...
=INFO REPORT===== 30-Dec-2015::11:33:18 =====
Statistics database started.
completed with 7 plugins.
=INFO REPORT===== 30-Dec-2015::11:33:18 =====
Server startup complete; 6 plugins started.
* rabbitmq_management
* rabbitmq_web_dispatch
* rabbitmq_management_agent
* webmachine
* mochiweb
* amp_client
```

## **Administración**

El nombre del plugin es `rabbitmq_management` y permite acceder a una web de administración que también tiene su propio api rest. Los usuarios de esta herramienta administrativa o del api, pueden tener distintos roles. Por defecto trae un usuario `guest` con el mismo password. Con este usuario inicial, se pueden crear usuarios administrativos con cada uno de los roles permitidos para acceder a la administración.

Se puede securizar el acceso mediante SSL a esta url administrativa con un certificado cliente. Pero la autorización de usuarios tiene que coexistir.

## **Autenticación**

Los mecanismos de autenticación que ofrece RabbitMq y la comunidad desarrolladora colaboradora son los siguientes:

- SASL PLAIN y SASL AMQPLAIN: por defecto, usuario y password en texto plano, por lo que es recomendable enviarlos a través de SSL.
- SASL EXTERNAL: para indicar si un usuario puede acceder a través de algún mecanismo de autenticación externo, dependiendo de cada uno de ellos habrá que usar distintos plugins:
- SSL: rabbitmq-auth-mechanism-ssl Autentica a los usuarios con certificado de cliente.
- LDAP: rabbitmq\_auth\_backend\_ldap Autentica a los usuarios contra un directorio activo.
- HTTP: rabbitmq-auth-backend-http Realiza consultas a un servidor web que conoce las credenciales de cada usuario y puede permitir o no a ciertos usuarios acceso a los recursos, generando una respuesta en un formato predefinido.

## **Autorización**

- Interna: RabbitMQ tiene su propia base de datos que guarda a qué recursos tienen acceso sus usuarios.
- LDAP: Complejo ya que el directorio activo no conoce los permisos que un usuario necesita en el protocolo AMQP. RabbitMQ añade un mecanismo de consulta para saber si a un usuario se le permite el acceso a un host virtual. Para más información, ver Configuring authorisation en la documentación del plugin de LDAP de RabbitMQ.

## **SSL**

Para securizar un servidor de Rabbit es necesario en primer lugar activar el plugin rabbitmq\_auth\_mechanism\_ssl

```
root@my-rabbit:/# rabbitmq-plugins enable rabbitmq_auth_mechanism_ssl
The following plugins have been enabled:
rabbitmq_auth_mechanism_ssl
```

Disponer de una entidad certificadora, y unos certificados cliente-servidor (o bien generarlos con la ayuda de openssl) y configurar el fichero de configuración

de RabbitMQ bajo /etc/rabbitmq/rabbitmq.config de una manera similar a ésta:

```
[
  {ssl, [{versions, ['tlsv1.2', 'tlsv1.1']}]},
  {rabbit, [{loopback_users, []},
    {auth_mechanisms, ['EXTERNAL']},
    {ssl_cert_login_from, common_name},
    {ssl_listeners, [5673]},
    {ssl_options, [{cacertfile, "/etc/rabbitmq/ssl/cacert.pem"},
      {certfile, "/etc/rabbitmq/ssl/server/rabbit3.cert.pem"},
      {keyfile, "/etc/rabbitmq/ssl/server/rabbit3.key.pem"},
      {verify, verify_peer},
      {fail_if_no_peer_cert, true}
    ]}
  ]},
  {rabbitmq_management, [{listener, [{port, 15672},
    {ssl, true},
    {ssl_opts, [{cacertfile, "/etc/rabbitmq/ssl/cacert.pem"},
      {certfile, "/etc/rabbitmq/ssl/server/rabbit3.cert.pem"},
      {keyfile, "/etc/rabbitmq/ssl/server/rabbit3.key.pem"},
      {verify, verify_peer},
      {fail_if_no_peer_cert, true}
    ]}
  ]}
  ]}
  ]}
  ].
```

## **Exchanges, queues y bindings**

La forma de enrutar mensajes de RabbitMQ es a través de intercambios: exchanges, dentro de un exchange pueden estar definidas colas (queues) y se puede publicar en cada una de ellas, según una routing key que se defina. Para declarar esta configuración, siguiendo la web de referencia, hay que descargarse el script de la versión de rabbit correspondiente desde la web administrativa /cli/, rabbitmqadmin

Situarlo en el directorio /usr/bin con los permisos correspondientes de ejecución y a continuación se podrían ir declarando exchanges, colas y bindings como en el ejemplo.

```
rabbitmqadmin declare exchange name=myexchange type=direct
rabbitmqadmin declare queue name=myqueue durable=true
rabbitmqadmin declare binding source="myexchange"
destination_type="queue" destination="myqueue" routing_key="mykey"
```

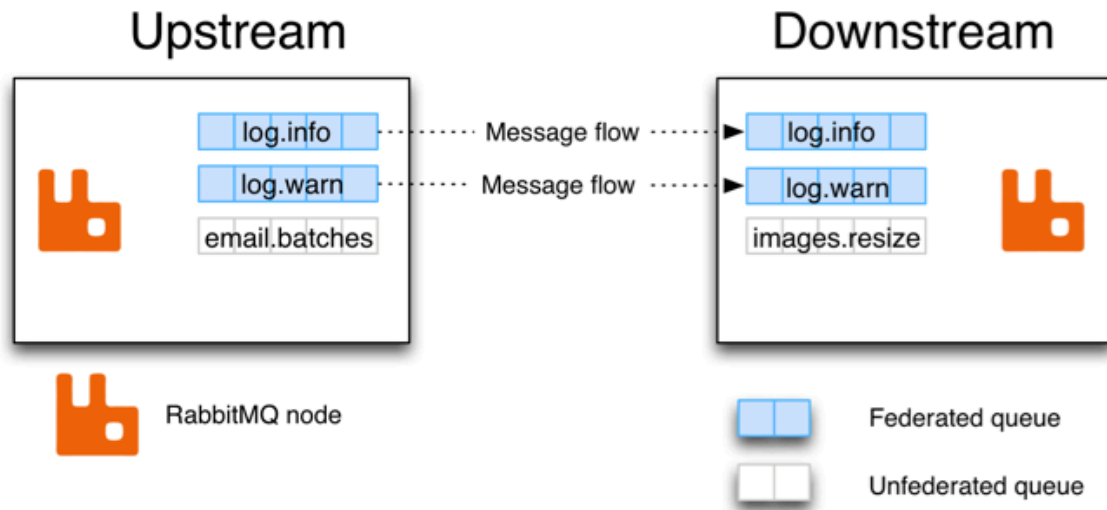
## **Federación**

Para poder tener una instalación con alta disponibilidad con al menos 2 brokers de RabbitMQ, estando disponibles para los consumidores en cada uno de ellos los mensajes mandados a ambos, necesitamos realizar configuraciones adicionales a las mostradas en los pasos anteriores.

La solución que se aplica es: Rabbit Federation ya que tolera conexiones intermitentes como las que pueden ocurrir en una WAN entre otras ventajas.

Además de poder ser escalable aunque no de manera automática. Los pasos que habría que seguir serían:

- Activación de Plugins Federation
- rabbitmq\_federation
- rabbitmq\_federation\_management
- Instalación Certificados Cliente y Servidor en los brokers
- Alta de Usuarios Federated Upstreams
- Declarar Upstreams y Política de intercambio entre Brokers Federados



## Más información

Si queréis profundizar sobre estos aspectos de RabbitMQ, además de la web que es bastante completa, he dejado un proyecto en Github, en el que existe un ejemplo de un productor-consumidor desarrollado bajo spring-amqp y una imagen de docker para construir un Rabbit federado compuesto de dos nodos y securizado mediante SSL.