

Part 4: RabbitMQ Exchanges, routing keys and bindings

🕒 Last updated: 2019-09-24

What is an exchange? What are routing keys and bindings? How are exchanges and queues associated with each other? When should I use them and how? This article explains the different types of exchanges in RabbitMQ and scenarios for how to use them.

Messages are not published directly to a queue. Instead, the producer sends messages to an exchange. Exchanges are message routing agents, defined by the virtual host within RabbitMQ. An exchange is responsible for routing the messages to different queues with the help of header attributes, bindings, and routing keys.

A **binding** is a "link" that you set up to bind a queue to an exchange.

The **routing key** is a message attribute the exchange looks at when deciding how to route the message to queues (depending on exchange type).

Exchanges, connections, and queues can be configured with parameters such as *durable*, *temporary*, and *auto delete* upon creation. Durable exchanges survive server restarts and last until they are explicitly deleted. Temporary exchanges exist until RabbitMQ is shut down. Auto-deleted exchanges are removed once the last bound object is unbound from the exchange.

In RabbitMQ, there are four different types of exchanges that route the message differently using different parameters and bindings setups. Clients can create their own exchanges or use the predefined default exchanges which are created when the server starts for the first time.



LOVISA JOHANSSON

Developer

FREE EBOOK

"The Optimal
RabbitMQ Guide"

📄 [Download your copy.](#)

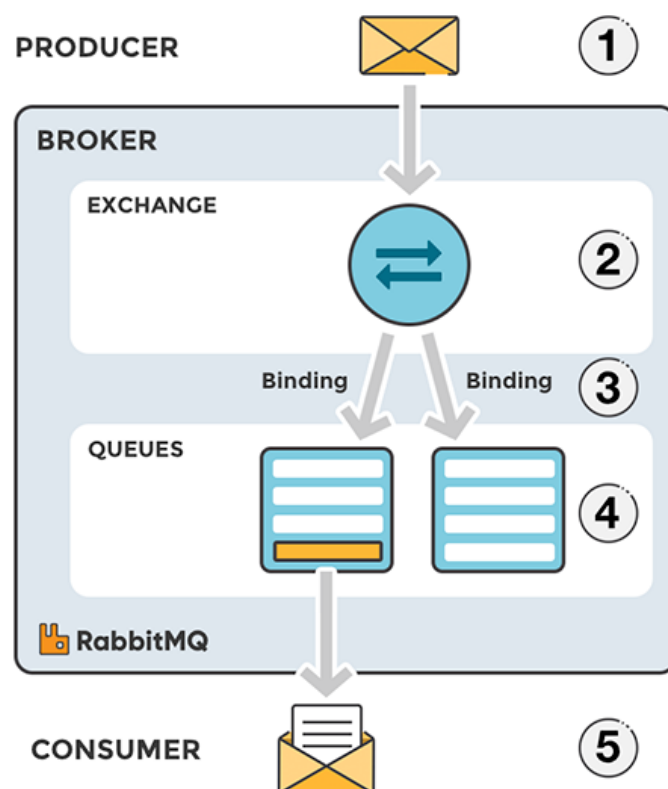
[Tweets by CloudAMQP](#)

RabbitMQ Explained - Exchanges



Standard RabbitMQ message flow

1. The producer publishes a message to the exchange.
2. The exchange receives the message and is now responsible for the routing of the message.
3. Binding must be set up between the queue and the exchange. In this case, we have bindings to two different queues from the exchange. The exchange routes the message into the queues.
4. The messages stay in the queue until they are handled by a consumer.
5. The consumer handles the message.



If you are not familiar with RabbitMQ and message queueing, read [RabbitMQ for beginners - what is RabbitMQ?](#) before reading about exchanges, routing keys, headers, and bindings.

Exchange types

Direct Exchange

A direct exchange delivers messages to queues based on a message routing key. The routing key is a message attribute added to the message header by the producer. Think of the routing key as an "address" that the exchange is using to decide how to route the message. **A message goes to the queue(s) with the binding key that exactly matches the routing key of the message.**

The direct exchange type is useful to distinguish messages published to the same exchange using a simple string identifier.

The default exchange AMQP brokers must provide for the direct exchange is "amq.direct".

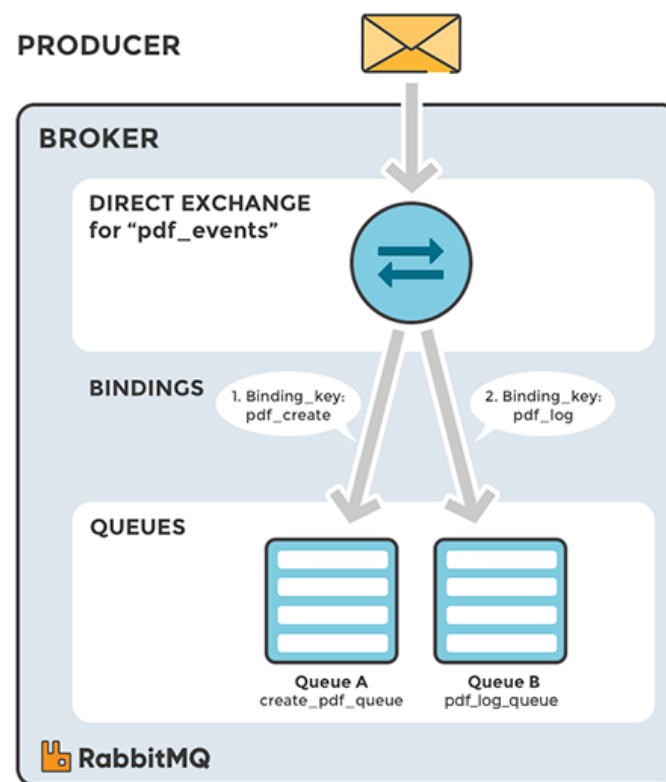
Imagine that queue A (create_pdf_queue) in the image below (Direct Exchange Figure) is bound to a direct exchange (pdf_events) with the binding key *pdf_create*. When a new message with routing key *pdf_create* arrives at the direct exchange, the exchange routes it to the queue where the *binding_key* = *routing_key*, in the case to queue A (create_pdf_queue).

SCENARIO 1

- Exchange: pdf_events
- Queue A: create_pdf_queue
- Binding key between exchange (pdf_events) and Queue A (create_pdf_queue): pdf_create

SCENARIO 2

- Exchange: pdf_events
- Queue B: pdf_log_queue
- Binding key between exchange (pdf_events) and Queue B (pdf_log_queue): pdf_log



EXAMPLE

Example: A message with routing key *pdf_log* is sent to the exchange *pdf_events*. The message is routed to pdf_log_queue because the routing key (pdf_log) matches the binding key (pdf_log).

If the message routing key does not match any binding key, the message is discarded.

Direct Exchange: A message goes to the queues whose binding key exactly matches the routing key of the message.

Default exchange

The default exchange is a pre-declared direct exchange with no name, usually referred to by an empty string. When you use default exchange, your message is delivered to the queue with a name equal to the routing key of the message. Every queue is automatically bound to the default exchange with a routing key which is the same as the queue name.

Topic Exchange

Topic exchanges route messages to queues based on wildcard matches between the routing key and the routing pattern, which is specified by the queue binding. Messages are routed to one or many queues based on a matching between a message routing key and this pattern.

The routing key must be a list of words, delimited by a period (.). Examples are *agreements.us* and *agreements.eu.stockholm* which in this case identifies agreements that are set up for a company with offices in lots of different locations. The routing patterns may contain an asterisk ("*") to match a word in a specific position of the routing key (e.g., a routing pattern of "agreements.*.b.*" only match routing keys where the first word is "agreements" and the fourth word is "b"). A pound symbol ("#") indicates a match of zero or more words (e.g., a routing pattern of "agreements.eu.berlin.#" matches any routing keys beginning with "agreements.eu.berlin").

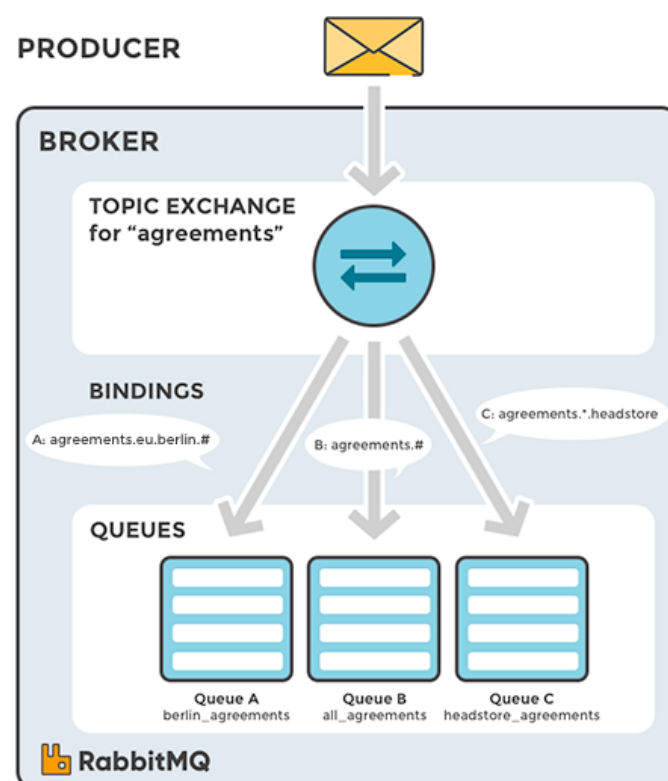
The consumers indicate which topics they are interested in (like subscribing to a feed for an individual tag). The consumer creates a queue and sets up a binding with a given routing pattern to the exchange. All messages with a routing key that match the routing pattern are routed to the queue and stay there until the consumer consumes the message.

The default exchange AMQP brokers must provide for the topic exchange is "amq.topic".

SCENARIO 1

The image to the right shows an example where consumer A is interested in all the agreements in Berlin.

- Exchange: agreements
- Queue A: berlin_agreements
- Routing pattern between exchange (agreements) and Queue A (berlin_agreements):
agreements.eu.berlin.#
- Example of message routing key that matches:
agreements.eu.berlin and
agreements.eu.berlin.headstore



Topic Exchange: Messages are routed to one or many queues based on a match between a message routing key and the routing pattern.

SCENARIO 2

Consumer B is interested in all the agreements.

- Exchange: agreements
- Queue B: all_agreements
- Routing pattern between exchange (agreements) and Queue B (all_agreements):
agreements.#
- Example of message routing key that matches:
agreements.eu.berlin and
agreements.us

SCENARIO 3

Consumer C is interested in all agreements for European head stores.

- Exchange: agreements
- Queue C: headstore_agreements
- Routing pattern between exchange (agreements) and Queue C (headstore_agreements): agreements.eu.*.headstore
- Example of message routing keys that will match: agreements.eu.berlin.headstore and agreements.eu.stockholm.headstore

EXAMPLE

A message with routing key *agreements.eu.berlin* is sent to the exchange *agreements*. The messages are routed to the queue *berlin_agreements* because the routing pattern of "agreements.eu.berlin.#" matches the routing keys beginning with "agreements.eu.berlin". The message is also routed to the queue *all_agreements* because the routing key (agreements.eu.berlin) matches the routing pattern (agreements.#).

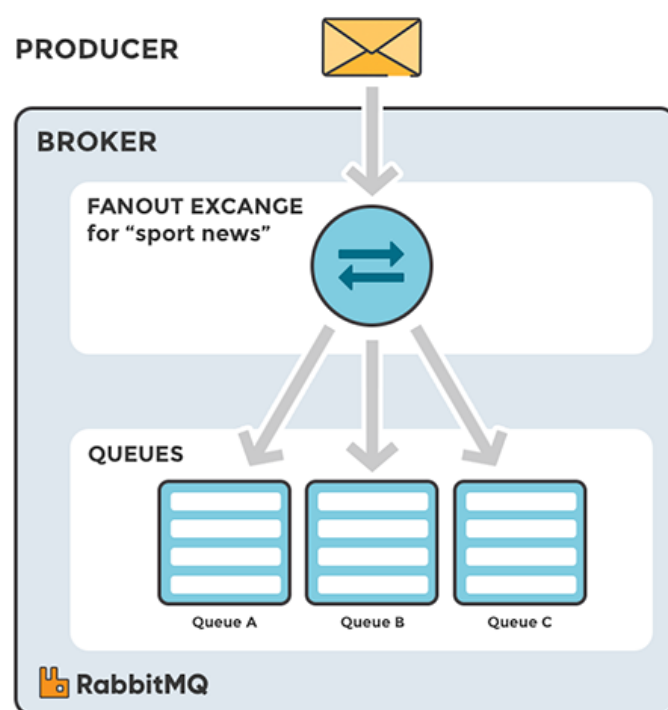
Fanout Exchange

A fanout exchange copies and routes a received message to all queues that are bound to it regardless of routing keys or pattern matching as with direct and topic exchanges. The keys provided will simply be ignored.

Fanout exchanges can be useful when the same message needs to be sent to one or more queues with consumers who may process the same message in different ways.

The image to the right (Fanout Exchange) shows an example where a message received by the exchange is copied and routed to all three queues bound to the exchange. It could be sport or weather updates that should be sent out to each connected mobile device when something happens, for instance.

The default exchange AMQP brokers must provide for the topic exchange is "amq.fanout".



Fanout Exchange: The received message is routed to all queues that are bound to the exchange.

SCENARIO 1

- Exchange: sport_news
- Queue A: Mobile client queue A
- Binding: Binding between the exchange (sport_news) and Queue A (Mobile client queue A)

EXAMPLE

A message is sent to the exchange *sport_news*. The message is routed to all queues (Queue A, Queue B, Queue C) because all queues are bound to the exchange. Provided routing keys are ignored.

Headers Exchange

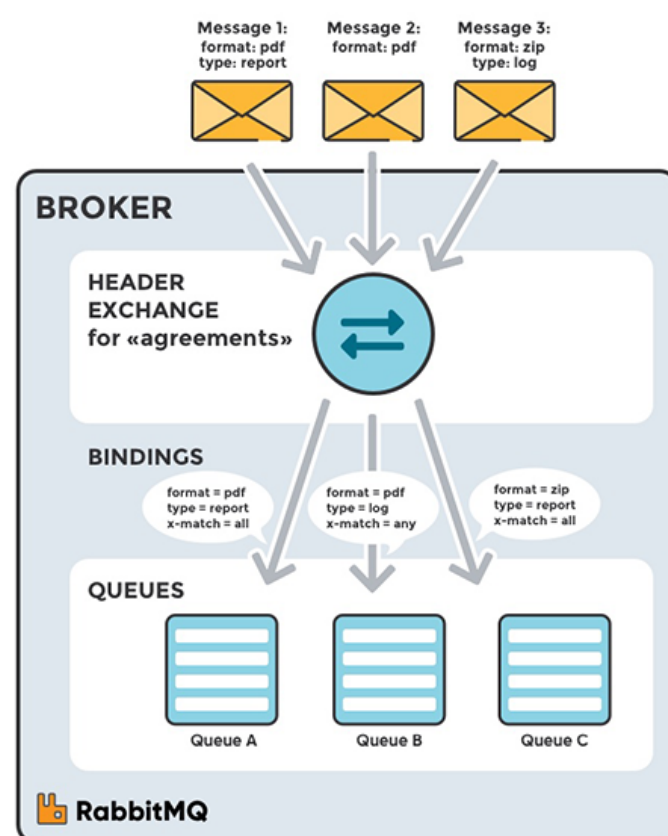
A headers exchange routes messages based on arguments containing headers and optional values. Headers exchanges are very similar to topic exchanges, but route messages based on header values instead of routing keys. A message matches if the value of the header equals the value specified upon binding.

A special argument named "x-match", added in the binding between exchange and queue, specifies if all headers must match or just one. Either any common header between the message and the binding count as a match, or all the headers referenced in the binding need to be present in the message for it to match. The "x-match" property can have two different values: "any" or "all", where "all" is the default value. A value of "all" means all header pairs (key, value) must match, while value of "any" means at least one of the header pairs must match. Headers can be constructed using a wider range of data types, integer or hash for example, instead of a string. The headers exchange type (used with the binding argument "any") is useful for directing messages which contain a subset of known (unordered) criteria.

The default exchange AMQP brokers must provide for the topic exchange is "amq.headers".

EXAMPLE

- Exchange: Binding to Queue A with arguments (key = value): format = pdf, type = report, x-match = all
- Exchange: Binding to Queue B with arguments (key = value): format = pdf, type = log, x-match = any
- Exchange: Binding to Queue C with arguments (key = value): format = zip, type = report, x-match = all



Example of Headers Exchange. Routes messages to queues that are bound using arguments (key and value) in the amq.headers attribute.

SCENARIO 1

Message 1 is published to the exchange with header arguments (key = value): "format = pdf", "type = report".

Message 1 is delivered to Queue A because all key/value pairs match, and Queue B since "format = pdf" is a match (binding rule set to "x-match = any").

SCENARIO 2

Message 2 is published to the exchange with header arguments of (key = value): "format = pdf".

Message 2 is only delivered to Queue B. Because the binding of Queue A requires both "format = pdf" and "type = report" while Queue B is configured to match any key-value pair (x-match = any) as long as either "format = pdf" or "type = log" is present.

SCENARIO 3

Message 3 is published to the exchange with header arguments of (key = value): "format = zip", "type = log".

Message 3 is delivered to Queue B since its binding indicates that it accepts messages with the key-value pair "type = log", it doesn't mind that "format = zip" since "x-match = any".

Queue C doesn't receive any of the messages since its binding is configured to match all of the headers ("x-match = all") with "format = zip", "type = pdf". No message in this example lives up to these criterias.

It's worth noting that in a header exchange, the actual order of the key-value pairs in the message is irrelevant.

Dead Letter Exchange

If no matching queue can be found for the message, the message is silently dropped. RabbitMQ provides an AMQP extension known as the "Dead Letter Exchange", which provides the functionality to capture messages that are not deliverable.

Please email us at contact@cloudamqp.com if you have any suggestions about missing content or other feedback.

GUIDE - RABBITMQ FOR BEGINNERS

[CloudAMQP - industry-leading RabbitMQ as a Service](#)

Sign Up

[GO BACK TO PART 3](#)

[The management interface](#)

Enjoy this article? Don't forget to share it with others. 😊



What do you think?
57 Responses

👍 Upvote

😂 Funny

❤️ Love

😮 Surprised

😡 Angry

😞 Sad

Comments Community Privacy Policy 1 Login ▾

Recommend 2 Tweet Share Sort by Best ▾

Join the discussion...

LOG IN WITH OR SIGN UP WITH DISQUS

Name



Arnaud Dumas • a year ago • edited
Hi

Nice article, simple and detailed.

I have a question please.
Having defined a routing-key on an exchange through a dynamic variable that I store in a header. In a case where the variable is null;
is there a way to define a default behavior so that messages are routed automatically based on existing topic exchange ? Thanks

2

^

|

▼


•

Reply

•

Share

›



LovisaJohansson

CloudAMQP

➔

Arnaud Dumas

• a year ago

Hi, thanks!

You can maybe set up an alternate exchanges via policies. Whenever an exchange with a configured alternate exchanges cannot route a message to any queue, it publishes the message to the specified alternate exchanges instead. (If that alternate exchanges does not exist then a warning is logged.) Here is a link:<https://www.rabbitmq.com/ae...>

^

|

▼


•

Reply

•

Share

›



Arnaud Dumas

➔

LovisaJohansson

• 10 months ago

Yes, it could be a good solution. Thanks

^

|

▼


•

Reply

•

Share

›



97vaqasazeem

• a year ago

Excellent article . Thanks for make it really simple

1

^

|

▼


•

Reply

•

Share

›



LovisaJohansson

CloudAMQP

➔

97vaqasazeem

• 8 months ago

Thank you for your feedback!

^

|

▼

•

Reply

•

Share

›



esprit immortel

• a year ago

great work and nice article and just wanna know that do you have youtube videos for the same.

1

^

|

▼


•

Reply

•

Share

›



Lovisa Johansson

➔

esprit immortel

• a year ago

We are working on a video about exchanges, but it's not yet finished :(

^

|

▼


•

Reply

•

Share

›



esprit immortel

➔

Lovisa Johansson

• a year ago

Hey I am bit confused on routing key and binding so can you help me please.

^

|

▼


•

Reply

•

Share

›



84codes

CloudAMQP

➔

esprit immortel

• a year ago

yes of course, can you specify your issue further?

^

|

▼


•

Reply

•

Share

›



esprit immortel

➔

84codes

• a year ago

Do you have any plan for live online session for Q & A?

Please **read my comments below and correct me If I am wrong:**

In my opinion, **Broker is an abstract design where all component resides like exchange queues**
Routing key is used to find the correct Queue and send the message from exchange to queue.
and Binding a path from exchange to queue.

But then what is binding key?
What is connection? and we are not defining anywhere in java code.
What is channel? as you said its

what is channel ? as you said its virtual connection so can we define through code?
Actually I am confused in assuming all these in one bucket and not able to imagine it that how the things are working properly.
^ | v • Reply • Share ›



Daniel Marklund CloudAMQP →
esprit immortel • a year ago

Correct. RabbitMQ is an example of a broker (and everything it entails: queues, exchanges etc).

I can recommend our blog post about [the relationship between Channels and Connections](#) to clear things up. And you can see in our [Java example docs](#) how to create a connection and channel.

Yes you are right. "Binding key" vs "routing key" can be confusing. The [Python tutorial from the RabbitMQ team](#) attempts to explain it in the following words:

A binding is a relationship between an exchange and a queue. This can be simply read as: the queue is interested in messages from this exchange. Bindings can take an extra routing_key parameter. To avoid the confusion with a basic_publish parameter we're going to call it a binding key... The meaning of a binding key depends on the exchange type.

I hope that helps. Let us know if you have any further questions. A live Q & A sounds like a great idea! We will look into that.

1 ^ | v • Reply • Share ›



esprit immortel → Daniel Marklund
• a year ago

Hi Deniel,
Thanks for sharing and really helpful for me but now have some more questions:

A connection (TCP) is a link between the client and the broker

So what about connections between Queue and receiver?

Many applications needs to have multiple connections to the broker,

Means the clients who wants to send the messages?

[see more](#)

^ | v • Reply • Share ›



LovisaJohansson CloudAMQP →
esprit immortel • a year ago

Hi,

A connection is a TCP connection, while a channel is an AMQP connection. You are first of all opening up a TCP connection, connecting to the server (handshake process etc), you are then opening a channel in that connection - meaning that you don't need to redo the connection process.

Yes, you can reuse the connection and instead create more channels. Channels is the "connection" you are doing operations on, like `queue.create` and `exchange.create` on.

1 ^ | v • Reply • Share ›



Gaurav Kumar ➔ esprit immortel
• a year ago • edited

Client means a producer or a receiver.

There are basically 3 components. Producer, Broker and Receiver.

Broker: It contain exchanges. Each exchange in a way is connected to queues.

Producer: You create a connection between a producer and broker. When you publish a message you also specify an exchange and a parameter that specify which queue to push this message to.

Receiver: You create a connection between broker and receiver. You also specify which queue in the broker to get message from. So you connect only to broker. Rest of the

[see more](#)

1 ^ | v • Reply • Share ›



Maciej Tyrcha • 3 months ago

Really nice explanation of exchange mechanism. From what I understand both Producer and Consumer are not the users who sends / recieves those messages but functionalities in our application. So the flow is something like that 1. User sends a message with rest API to the server 2 The message goes into Producer 3 The