Part 1: RabbitMQ for beginners - What is RabbitMQ?

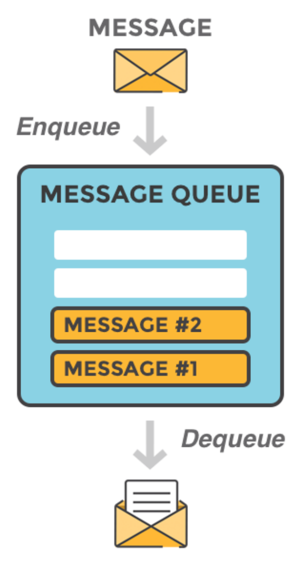
Last updated: 2019-09-23

The first part of **RabbitMQ for beginners** explains what RabbitMQ and message queueing is - the guide also gives a brief understanding of message queueing and defines important concepts. The guide goes on to explain the steps to set up a connection and the basics of publishing/consuming messages from a queue.

LOVISA JOHANSSON
Developer

THIS BLOG SERIES IS A LIVING DOCUMENT THAT IS CONTINUALLY UPDATED. LAST UPDATED JANUARY 2022.

RabbitMQ is a message-queueing software also known as a *message broker* or *queue manager.* Simply said; it is software where queues are defined, to which applications connect in order to transfer a message or messages.



A message can include any kind of information. It could, for example, have information about a process or task that

TABLE OF CONTENTS

should start on another application (which could even be on another server), or it could be just a simple text message. The queue-manager software stores the messages until a receiving application connects and takes a message off the queue. The receiving application then processes the message.

All articles from **Getting Started with RabbitMQ** can be downloaded as a free ebook here.

An **online training tool for RabbitMQ** can be found at: https://training.cloudamqp.com/

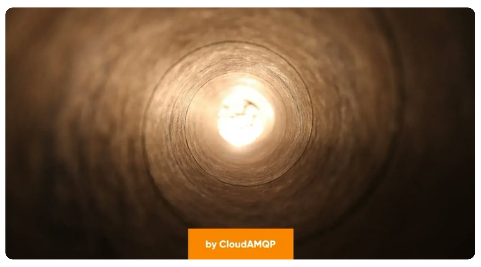
are associated with each other

Download the ebook Getting started with RabbitMQ for free

Tweets by @CloudAMQP

CloudAMQP
@CloudAMQP

The #RabbitMQ delayed exchange plugin implements a wait time between when a message reaches the exchange and when it is delivered to a queue. This article will help you understand how it works to avoid potential pitfalls. #CloudAMQP #SaaS #messagequeueingbuff.ly/38iQhlu



May 12, 2022

CloudAMQP

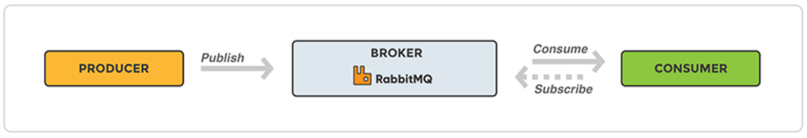
Embed View on Twitter

RabbitMQ Example

A message broker acts as a middleman for various services (e.g. a web application, as in this example). They can be used to reduce loads and delivery times of web application servers by delegating tasks that would normally take up a lot of time or resources to a third party that has no other job.

In this guide, we follow a scenario where a web application allows users to upload information to a website. The site will handle this information, generate a PDF, and email it back to the user. Handling the information, generating the PDF, and sending the email will, in this example case, take several seconds. That is one of the reasons why a message queue will be used to perform the task.

When the user has entered user information into the web interface, the web application will create a "PDF processing" message that includes all of the important information the user needs into a message and place it onto a queue defined in RabbitMQ.

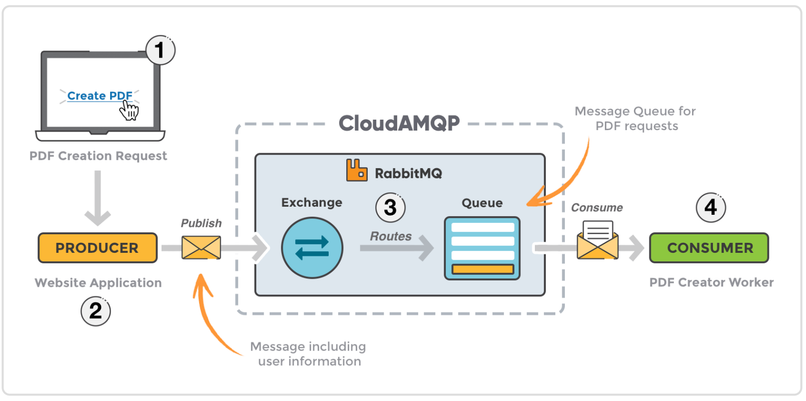


The basic architecture of a message queue is simple - there are client applications called producers that create messages and deliver them to the broker (the message queue). Other applications, called consumers, connect to the queue and subscribe to the messages to be processed. Software may act as a producer, or consumer, or both a consumer and a producer of messages. Messages placed onto the queue are stored until the consumer retrieves them.

When and why should you use RabbitMQ?

Message queueing allows web servers to respond to requests quickly instead of being forced to perform resource-heavy procedures on the spot that may delay response time. Message queueing is also good when you want to distribute a message to multiple consumers or to balance loads between workers.

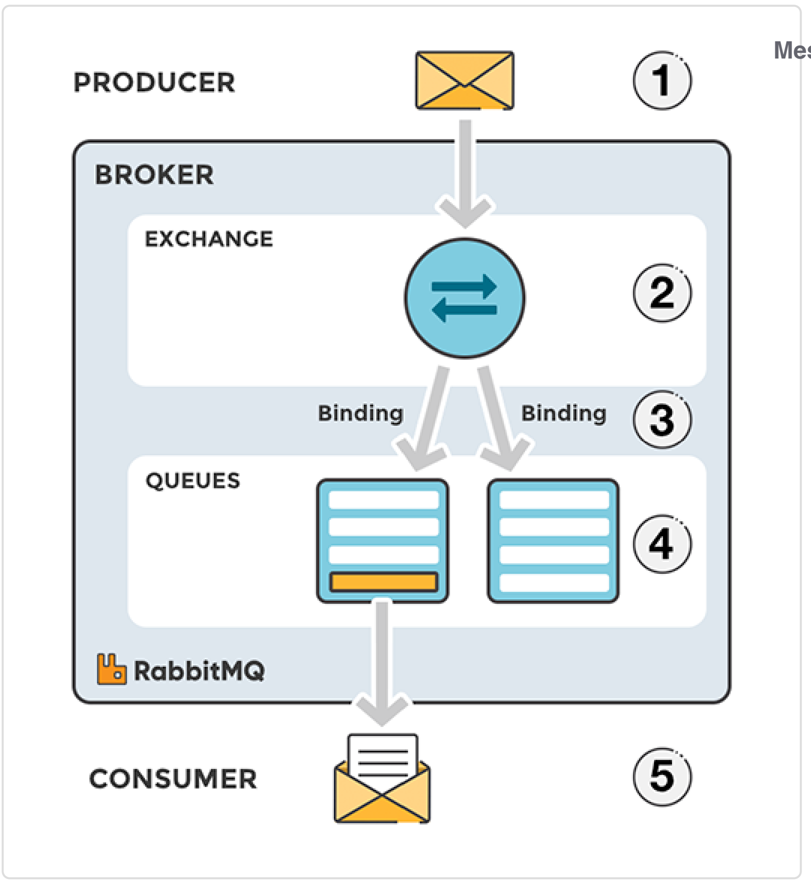
The consumer takes a message off the queue and starts processing the PDF. At the same time, the producer is queueing up new messages. The consumer can be on a totally different server than the producer or they can be located on the same server. The request can be created in one programming language and handled in another programming language. The point is, the two applications will only communicate through the messages they are sending to each other, which means the sender and receiver have low coupling.



1. The user sends a PDF creation request to the web application.
2. The web application (the producer) sends a message to RabbitMQ that includes data from the request such as name and email.
3. An exchange accepts the messages from the producer and routes them to correct message queues for PDF creation.
4. The PDF processing worker (the consumer) receives the task message and starts processing the PDF.

EXCHANGES

Messages are not published directly to a queue; instead, the producer sends messages to an exchange. An exchange is responsible for routing the messages to different queues with the help of bindings and routing keys. A binding is a link between a queue and an exchange.

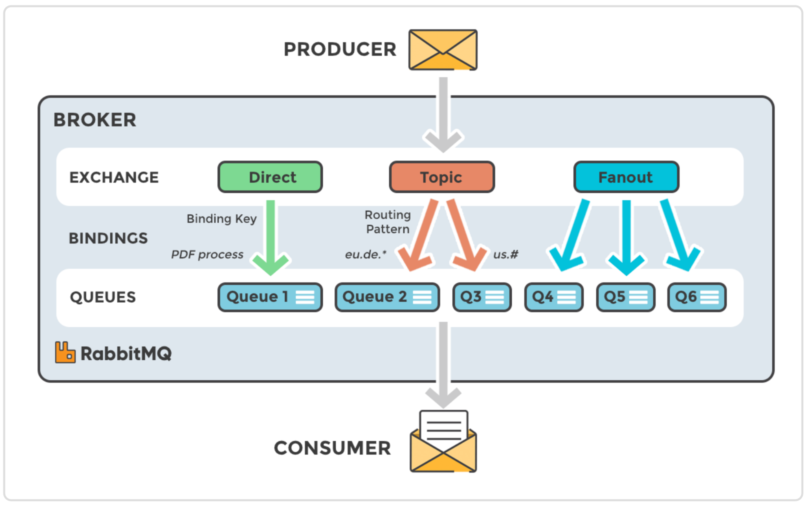


Message flow in RabbitMQ

1. The producer publishes a message to an exchange. When creating an exchange, the type must be specified. This topic will be covered later on.
2. The exchange receives the message and is now responsible for routing the message. The exchange takes different message attributes into account, such as the routing key, depending on the exchange type.
3. Bindings must be created from the exchange to queues. In this case, there are two bindings to two different queues from the exchange. The exchange routes the message into the queues depending on message attributes.
4. The messages stay in the queue until they are handled by a consumer
5. The consumer handles the message.

TYPES OF EXCHANGES

Part 2 of the tutorial uses direct exchanges. A deeper understanding of the different exchange types, binding keys, routing keys and how or when you should use them can be found in Part 4: RabbitMQ for beginners - Exchanges, routing keys and bindings.



- **Direct:** The message is routed to the queues whose binding key exactly matches the routing key of the message. For example, if the queue is bound to the exchange with the binding key *pdfprocess, a message published to* the exchange with a routing key *pdfprocess is routed to that queue.*
- **Fanout:** A fanout exchange routes messages to all of the queues bound to it.
- **Topic:** The topic exchange does a wildcard match between the routing key and the routing pattern specified in the binding.
- **Headers:** Headers exchanges use the message header attributes for routing.

RABBITMQ AND SERVER CONCEPTS

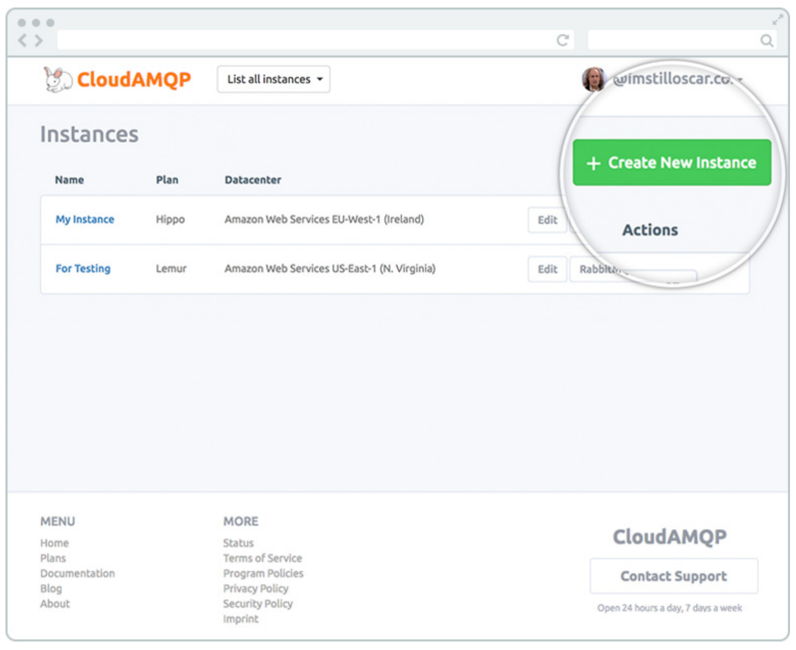
Some important concepts need to be described before we dig deeper into RabbitMQ. The default virtual host, the default user, and the default permissions are used in the examples, so let's go over the elements and concepts:

- **Producer:** Application that sends the messages.
- **Consumer:** Application that receives the messages.
- **Queue:** Buffer that stores messages.
- **Message:** Information that is sent from the producer to a consumer through RabbitMQ.
- **Connection:** A TCP connection between your application and the RabbitMQ broker.
- **Channel:** A virtual connection inside a connection. When publishing or consuming messages from a queue - it's all done over a channel.
- **Exchange:** Receives messages from producers and pushes them to queues depending on rules defined by the exchange type. To receive messages, a queue needs to be bound to at least one exchange.
- **Binding:** A binding is a link between a queue and an exchange.
- **Routing key:** A key that the exchange looks at to decide how to route the message to queues. Think of the routing key like an *address for the message*.
- **AMQP:** Advanced Message Queuing Protocol is the protocol used by RabbitMQ for messaging.
- **Users:** It is possible to connect to RabbitMQ with a given username and password. Every user can be assigned permissions such as rights to read, write and configure privileges within the instance. Users can also be assigned permissions for specific virtual hosts.
- **Vhost, virtual host:** Provides a way to segregate applications using the same RabbitMQ instance. Different users can have different permissions to different vhost and queues and exchanges can be created, so they only exist in one vhost.

At the beginning of this article series, we had one producer (the website application) and one consumer (the PDF processing application). If the PDF processing application crashes, or if many PDF requests are coming at the same time, messages would continue to stack up in the queue until the consumer starts again. It would then process all the messages, one by one.

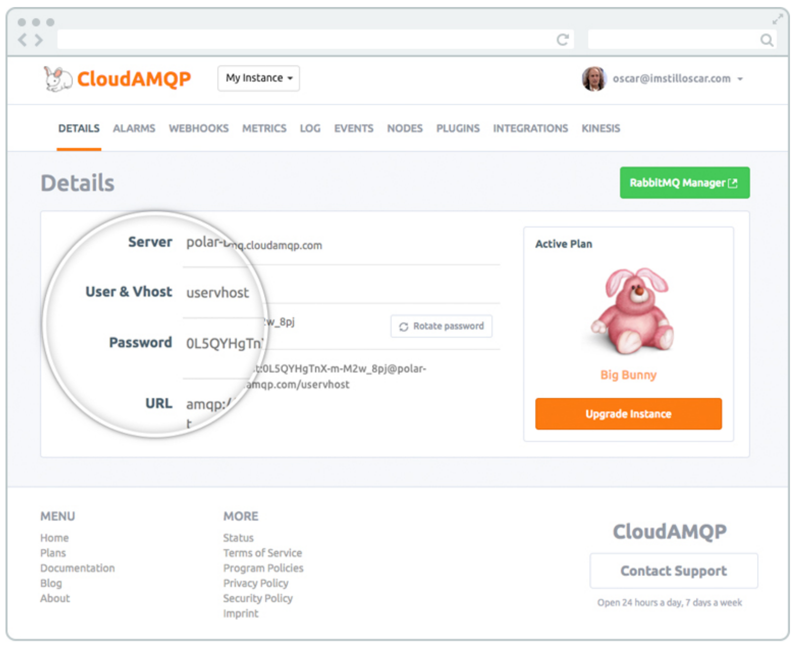
Set up a RabbitMQ instance

To be able to follow this guide you need to set up a CloudAMQP instance or download and install RabbitMQ. CloudAMQP is a hosted RabbitMQ solution, meaning that all you need to do is sign up for an account and create an instance. You do not need to set up and install RabbitMQ or care about cluster handling, CloudAMQP will do that for you. CloudAMQP can be used for free with the plan little lemur. Go to the plan page and sign up for any plan and create an instance.



Name	Plan	Datacenter	
My Instance	Hippo	Amazon Web Services EU-West-1 (Ireland)	Edit
For Testing	Lemur	Amazon Web Services US-East-1 (N. Virginia)	Edit

When your instance is created, click on details for your instance to find your username, password, and connection URL for your cloud-hosted RabbitMQ instance.



Server	polar-.nq.cloudamqp.com
User & Vhost	uservhost
Password	0L5QYHgTn w_8pj
URL	amqp:/ t:0L5QYHgTnX-m-M2w_8pj@polar- amqp.com/uservhost

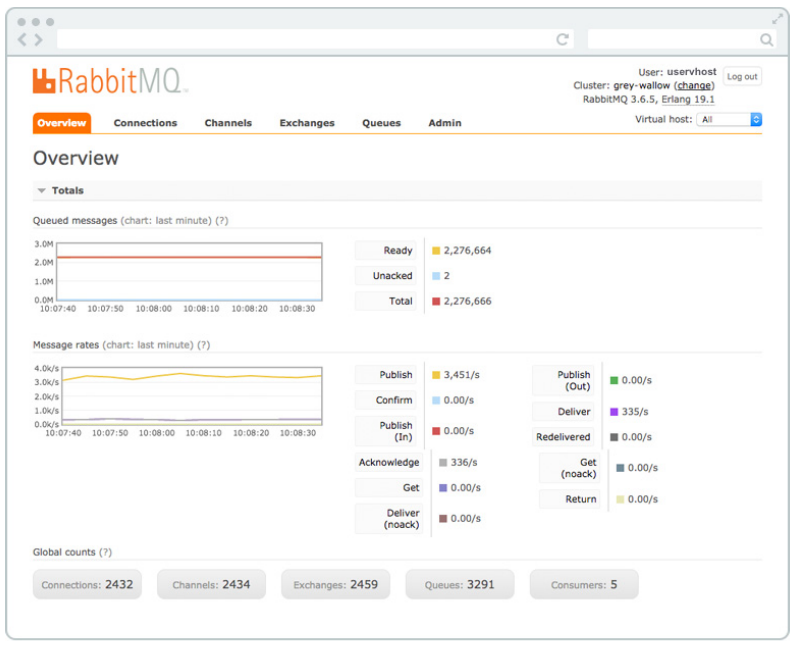
Getting started with RabbitMQ

Immediately after a RabbitMQ instance has been created it is possible to send a message across languages, platforms, and OS. This way of handling messages decouple your processes and creates a highly scalable system. You can now start by opening the management interface to get an overview of your RabbitMQ server.

THE MANAGEMENT INTERFACE - MANAGEMENT AND MONITORING

RabbitMQ provides a web UI for the management and monitoring of your RabbitMQ server. The RabbitMQ management interface is enabled by default in CloudAMQP and a link can be found on the details page for your CloudAMQP instance.

From the management interface, it is possible to handle, create, delete and list queues. It is also possible to monitor queue length, check message rate, or change and add users permissions and much more.



User: uservhost
Cluster: grey-wallow (change)
RabbitMQ 3.6.5, Erlang 19.1
Virtual host: All

Overview

Queued messages (chart: last minute) (?)

Ready	2,276,664
Unacked	2
Total	2,276,666

Message rates (chart: last minute) (?)

Publish	3,451/s	Publish (Out)	0.00/s
Confirm	0.00/s	Deliver	335/s
Publish (In)	0.00/s	Redelivered	0.00/s
Acknowledge	336/s	Get (noack)	0.00/s
Get	0.00/s	Return	0.00/s
Deliver (noack)	0.00/s		

Global counts (?)

Connections: 2432 Channels: 2434 Exchanges: 2459 Queues: 3291 Consumers: 5

More information about the management interface can be found in Part 3 - The management interface.

PUBLISH AND SUBSCRIBE MESSAGES

RabbitMQ uses a protocol called AMQP by default. To be able to communicate with RabbitMQ you need a library that understands the same protocol as RabbitMQ. Download the client library for the programming language that you intend to use for your applications. A client library is an application programming interface (API) for use in writing client applications. A client library has several methods; in this case, to communicate with RabbitMQ. The methods should be used when you connect to the RabbitMQ broker (using the given parameters, hostname, port number, etc.), for example, or when you declare a queue or an exchange. There is a choice of libraries for almost every programming language.

Steps to follow when setting up a connection and publishing a message/consuming a message:

1. Set up/create a connection object. The username, password, connection URL, port, etc., will need to be specified. A TCP connection will be set up between the application and RabbitMQ when the *start* method is called.
2. Create a channel in the TCP connection, then the connection interface can be used to open a channel through which to send and receive messages.
3. Declare/create a queue. Declaring a queue will cause it to be created if it does not already exist. All queues need to be declared before they can be used.
4. Set up exchanges and bind a queue to an exchange in subscriber/consumer. All exchanges must be declared before they can be used. An exchange accepts messages from a producer application and routes them to message queues. For messages to be routed to queues, queues must be bound to an exchange.
5. In publisher: Publish a message to an exchange
   In subscriber/consumer: Consume a message from a queue.
6. Close the channel and the connection.

SAMPLE CODE

Sample code will be given in part 2, starting with Part 2.1 - Ruby, followed by Part 2.2 - Node.js, and Part 2.3 Python, Having different programming languages on different parts of the system is possible - for example, the publisher could be written in node.js and the subscriber in Python.

We hope this article helped you gain an understanding of RabbitMQ!

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

GUIDE - RABBITMQ FOR BEGINNERS

CONTINUE WITH PART 2

Get started with RabbitMQ - Sample code

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

What do you think?

135 Responses

Upvote Funny

Love Surprised

Angry Sad

19 Comments **CloudAMQP** **Disqus' Privacy Policy** 1 **Login**

♡ **Favorite** 6 Tweet Share **Sort by Best**

Join the discussion…

LOG IN WITH OR SIGN UP WITH DISQUS ?

Name

Brad Gibson • a year ago

Thank you for the clear and concise introduction to RabbitMQ and the problem it addresses!

Can you share what tool(s) you used for your diagrams in the article, they are excellent as well!

2 ^ | ˅ • Reply • Share ›

Daniel Marklund CloudAMQP ➦ Brad Gibson • a year ago

Very glad you liked it.
The diagrams are custom made. We have reusable graphical components inside Figma which makes it easy for anyone in the team to create illustrations with a consistent look.

2 ^ | ˅ • Reply • Share ›

Krishnareddy Thamatam • 8 months ago

Thanks for the explanation..

1 ^ | ˅ • Reply • Share ›

animesh • 9 months ago
Thanks for the clear and precise explanation.
1 ^ | ˅ • Reply • Share ›

Максим • 9 months ago
Great solution! Thx a lot! I will try to learn more!!
1 ^ | ˅ • Reply • Share ›

Lovisa Johansson CloudAMQP → Максим • 8 months ago
Thanks! Have a look at our training tool: https://training.cloudamqp.... is you want to (still in beta thou).
^ | ˅ • Reply • Share ›

smanaqvi83 • a year ago
Excellent article, guys in the part 2 can you also cover Java with Spring Boot?
1 ^ | ˅ • Reply • Share ›

Lovisa Johansson CloudAMQP → smanaqvi83 • a year ago
Hi, thank you!

Good suggestion, we will add that to our pipeline of things to write about.
^ | ˅ • Reply • Share ›

Augustin Akpagni • 2 months ago
I have a question? Does the Consumers never deconnect to rabbitMq?
^ | ˅ • Reply • Share ›

Mohammad Elsheimy • 4 months ago
Thanks for the explanation, easy as a pie :)
^ | ˅ • Reply • Share ›

Sankhadeep Das • a year ago
Under 'PUBLISH AND SUBSCRIBE MESSAGES' section at 4th pointer it says "Set up exchanges and bind a queue to an exchange in subscriber/consumer". But exchanges & queues are created on the message broker. Are we considering the message broker as a consumer too?
^ | ˅ • Reply • Share ›

Lovisa Johansson CloudAMQP → Sankhadeep Das • a year ago
No, we are not considering the message broker a consumer. You can set up the exchange and bind the queue to the exchange in the consumer/publisher (via code, like in this example (PHP): https://www.cloudamqp.com/d...).

```
$exchange = 'amq.direct';
$queue = 'basic_get_queue';
$ch->queue_declare($queue, false, true, false, false);
$ch->exchange_declare($exchange, 'direct', true, true, false);
$ch->queue_bind($queue, $exchange);
```
^ | ˅ • Reply • Share ›

Jack Diamond • a year ago
What about the scenario where there are multiple producers requesting 'PDF creating' jobs. How do the producers 'map' which pdf belongs to which http request once the consumer create the pdf's for all the producers? Is there a message tag?
^ | ˅ • Reply • Share ›

Lovisa Johansson CloudAMQP → Jack Diamond • a year ago
Multiple producers should be able to request 'PDF creating' jobs, even at the same time. You can always add a reference/id into the message, along with the message body. The user information in part 2 can, for example, include a json with pdf id, name, file info etc etc.
^ | ˅ • Reply • Share ›

Thomas Gotwig • a year ago
I think the Website Application can't send something directly to RabbitMQ, first of all it should go through a REST-Endpoint or something like that? 🤔
^ | ˅ • Reply • Share ›

whereismycoffee → Thomas Gotwig • a year ago

Sure it could. No different than connecting to a DB to run a query. Now if you're attempting it from an SPA, you would perform the call to RabbitMQ at the API endpoint, not the client.

1 ^ | ⌄ • Reply • Share ›

Lovisa Johansson CloudAMQP ➦ Thomas Gotwig • a year ago

Hi, you can read/listen how we built our system here: https://www.cloudamqp.com/b... or here: https://www.cloudamqp.com/b...