RabbitMQ Examples – Peer-To-Peer Chat

Why a peer-to-peer chat?

Some of the main reasons of a peer-to-peer chat are related to safety and the fact that everything is decentralized. The benefits of a peer-to-peer network is that no one depends on a server – or a set of servers – to handle the messages being transferred from one place to another. This also means that your data gets directly to the other party, without being encountered by anyone else.

However, there are some disadvantages as well, one being that someone with bad intentions could open a ton of connections to you and maybe deny your access to the application.

One example of such a peer-to-peer chat application is Shazzle, and you can read more about what it is and how it works here: <https://shazzle.com/articles/what-is-peer-to-peer-p2p-messaging/>.

How can RabbitMQ help us build such an application?

Well, we just have to make use of what features it offers to us, and we’ll begin by modelling our application, which is rather simple. The main part of the application stands in its name – the **peer**. A peer is simply a node in the peer-to-peer network, which is, in our case, a person who is willing to chat. Let’s give our person a name so we can easily refer to it later – we’ll call it Patrick. Or Patricia. Your choice.

We also need the bit of data that gets transferred between peers – the **message**.This one is prettystraightforward, as well. It only needs to know from where it got sent – the **sender**, where it should go – the **receiver**, and its **content**.

Now, Patrick needs a place where he can receive his messages, something where other people send him letters, something like a mailbox – this maps to a queue in RabbitMQ. So Patrick will have a queue, identified by its name, where he gets all of his messages from anyone he is connected to. And in a similar fashion, everyone from the peer-to-peer chat has their own queue, to which someone can send a message to.

What we would like now is to have a mechanism to allow communication with only people that Patrick knows – we don’t want Vadim to spam his mailbox day and night. For this, we can implement a simple protocol, which goes like this:

1. If Patrick wants to connect with Patricia, he sends Patricia a “Hello!” message.
2. Patricia sees that Patrick wants to connect with her, and she can choose to accept his request with an “Ack!” message. Or she can choose to ignore Patrick, just like everyone else does.
3. Patrick will either receive Patricia’s message, in which case they can start chatting, or… he does not, and neither of them can send messages to each other.

Fair enough. Everything sounds simple and amazing, but…

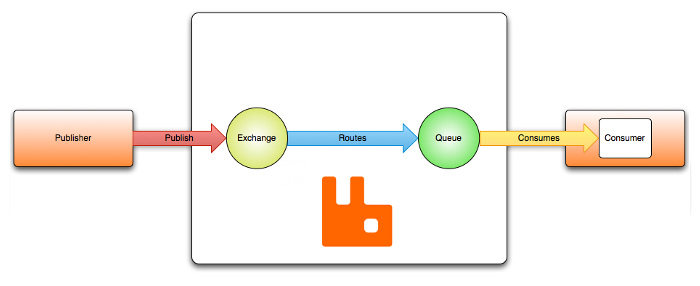
Hold up, is this **really** a peer-to-peer chat?



Conceptually, it is. Maybe if this were implemented directly with TCP connections and not with some kind of framework, it would have been actual peer-to-peer.

Practically, your messages actually go through RabbitMQ – the message broker I talked about earlier – which routes the messages from the publisher to the consumer’s queue.

You can see this in the diagram below.



However, this was a pretty simple example that teaches us about how to use RabbitMQ in such a scenario, and also we get a basic idea of what peer-to-peer is.

Bibliography and Resources:

1. <https://en.wikipedia.org/wiki/Peer-to-peer>
2. <https://en.bitcoinwiki.org/wiki/Peer-to-peer>
3. <https://shazzle.com/articles/what-is-peer-to-peer-p2p-messaging/>
4. [www.rabbitmq.com/tutorials/tutorial-one-java.html](http://www.rabbitmq.com/tutorials/tutorial-one-java.html)