# 2 Lessons Learned, And 3 Resources For For Learning RabbitMQ On NodeJS

March 26, 2014 By derickbailey — 4 Comments

I've been a huge fan of messaging systems and distributed application design through messages for a good number of years now. I've written several articles on this general area of development, and my MarionetteJS framework took a lot of influence from messaging based architectures. It's been a part of how I think for a long time now. But until last year, I had never used RabbitMQ. What's even worse is that until very recently (like, within the last 2 weeks), my use of RabbitMQ was mostly hopes and prayers, constantly wondering if my code was going to crash again.

But I've started to correct that mistake – the mistake of not properly learning RabbitMQ, and assuming that it worked in a manner similar to what I was used to. Turns out it doesn't… big surprise… but it can be used in a manner that I'm more accustomed to, given the right libraries on top of it. So I wanted to offer a couple of quick lessons learned from my experience in trying to learn more about RabbitMQ and improve my use of it. I also have a few resources you may want to look at, and a recommendation for NodeJS libraries.

## Lesson #1: One Connection Per Client Process. Many Channels Per Connection

This lesson alone was my biggest break-through in understanding RabbitMQ – and it all stems from how RabbitMQ manages connections to the server vs how you actually interact with the server.

In RabbitMQ, you have a connection and a channel. A connection is what it sounds like – it's the connection between a RabbitMQ client and a RabbitMQ server. This connection travels over the TCP/IP sockets or whatever wire protocol you're using. The thing about connections that I didn't understand, was that they are very expensive to create and destroy. A single RabbitMQ connection is a single TCP/IP connection. You want to avoid having too many of these open. In fact, I would go so far as to say you want to limit your RabbitMQ connections to one per client process. That is, if you have ApplicationA, a single RabbitMQ connection should be opened by and maintained by that application instance.

If Connections are TCP/IP (and they are, really), then Channels are the next protocol layer on top of Connections. Think of it like this… when you get on the internet, you have an open connection to some server somewhere. You can then choose to use HTTP, FTP, WebSockets, XMPP and other instant messaging protocols, and more. These protocols on top of TCP/IP are the communication channels that your applications use while connected to the internet. A channel in RabbitMQ is similar. It's the thing that your application uses to communicate with the RabbitMQ server.

Here's the best part about channels, though: you can (and should!) have a lot of open channels on top of a single connection. You can create and destroy channels very quickly, and very cheaply. They allow you to have a single connection to the RabbiMQ server, but have sandboxed communication for various parts of your application. Channels are how your application communicates with the RabbitMQ server.

So, keep one connection per client process (instance) and many channels within that process (instance).

## Lesson #2: Learn The Channel-Oriented Protocol / API Before Learning An Abstraction

One of the main reasons that I had a hard time learning RabbitMQ initially, and why my code was so terrible for so long, was my lack of understanding in how RabbitMQ actually works. When I started using it, I jumped right to a library that provided some abstractions on top of the channel-oriented nature of the protocol and I didn't understand the abstractions. My lack of understanding the protocol itself was to blame. I couldn't understand why the commands I was issuing were happening through the channel object all the time, instead of using Exchange and Queue objects like I expected.

It turns out the protocol itself is very channel-oriented. Understanding this opened my eyes as to why the library I was using was set up the way it is. I think there are possibilities for improving the API that we interact with, on top of the channel-oriented API set… and I've found a library that I'm using on top of it, that I like.

The point is, before you jump off the deep end and put yourself in a bad situation, like I did, take the time to learn the AMQP protocol (which is what RabbitMQ runs) and the RabbitMQ extensions to it. Having this foundational knowledge will make it easier for you to see which library you will want to use, and understand the options and API within that library. If you don't learn the protocol, you'll likely end up confused like I was.

## Some Resources For Learning RabbitMQ

I found it incredibly easy to get started with RabbitMQ, but had a little more difficulty getting anything more than a "hello world" message going. It wasn't until I started reading additional resources, other than what is listed at the RabbitMQ homepage, that I really started seeing how to build things and why. Here are some of the resources that I've been using:

- The RabbitMQ Docs – the official docs. Be sure to check out the tutorials, as well
- **RabbitMQ In Action – the best book that I've found on the subject**, and the book the finally taught me what I was doing wrong with the API / protocol. I HIGHLY recommend picking up this before and reading the intro / first two chapters before you start coding
- Alex Robson's Notes On RabbitMQ – I asked Alex a few questions a while back, and he posted this amazing gist of info. This is something I still go back to on a regular basis, to verify the direction that I'm heading against the things that Alex has said. It's not something that can be consumed / understood in one sitting, by a n00b, though. Keep it around as reference material, like I do.

There are countless other tutorials and blog posts around, but not that much from which I've learned much. I find myself continuing to go back to these resources for the info I need.

## My Choice Of NodeJS Libraries

When I first started trying to really learn RabbitMQ (after having used it for a while), I found myself with a dilemma: which of the many NodeJS libraries do I go with? There are three major players at this point:

- AMQPLib
- Node.AMQP
- BRAMQP

**Node.AMQP** might seem great off-hand, but from what I've read it is odd in that it hides exchanges or channels or something like that. I've only read enough about it to know that I don't want to use it. I haven't actually tried it, but I doubt that I will.

**BRAMQP** is a very low level API on top of RabbitMQ. It positions itself as letting you do ANYTHING with AMQP, because it's a very low level driver like API. But the problem is that you have to do everything yourself. This is a great choice if you're building a solid abstraction on top if a mountain of RabbitMQ knowledge and experience. I'm not there yet, so I'm not using this one yet.

****EDIT****

I was originally recommending the use of AMQPLib, as shown below. However, I ran in to a number of situations where using AMQPLib was too low-level. Having spent time with Rabbit.js, also mentioned below, I found it to be too limiting in certain features and scenarios. Fortunately, I have found a proper abstraction on top of AMQPLib that I am now recommending:

**LeanKit's "Wascally" library, for NodeJS**

I have found Wascally to be tremendously helpful in working with RabbitMQ, and am in the process of updating my current applications to use it.

---

That leaves **AMQPLib** (AKA "amqp.node") – **my current choice of NodeJS library / drivers.** This is the channel-oriented API that I mentioned previously, and was confused about at first. Having learned through the API and the way RabbitMQ works, though, I find it fairly easy to understand and work with.

But I wasn't super happy with using a somewhat low level API library in my code directly. I wanted to build domain specific objects for my application to use, and I wanted them to be based on the Enterprise Integration Patterns that I cut my teeth on, in the messaging world. So I started building my own wrappers to give me pub/sub, point-to-point and other semantics that I wanted. Well it wasn't long until I realized that the author of AMQPLib had already solved most of this for me, with his Rabbit.JS library.

**Rabbit.JS provides some of the core Enterprise Integration Patterns in a library on top of AMQPLib.** If you're coming from an EIP background when trying to learn RabbitMQ, I still suggest starting with the core and fundamentals of RabbitMQ. But once you get the basics down and you understand what an exchange is and how it can be used properly, then you should look at Rabbit.js. I'm finding it to be quite nice to work with and build my domain specific objects on top of.

## Other Resources?

I'm quickly growing fond of RabbitMQ and my choice of libraries for working with it in NodeJS. I've found some good resources, as I've noted above, but I'm sure there are other resources around, and other opinions and advice, too. I'd love to hear what resources you would suggest for someone learning RabbitMQ – especially on NodeJS, but any language would be good, really. Drop a note in the comments below, and let me know.

**P.S.** If you'd like to hear more about RabbitMQ, messaging patterns and other architecture and JavaScript related topics, be sure to join my mailing list. I offer tips and advice on my list that you won't hear anywhere else.

---

Filed Under: JavaScript, New Category, NodeJS, RabbitMQ

**About derickbailey**

Derick Bailey is an entrepreneur, developer, screencaster, writer, blogger, speaker and technology leader in central Texas (north of Austin). He runs a podcast hosting service at SignalLeaf.com, and throws down the screencasts at WatchMeCode.net. He has been a professional software developer since the late 90's, and has been writing code since the late 80's. In his spare time he builds ridiculous electronic toys with Arduino (and a nearby first aid kit), is 1/3rd of a podcast on being a developer/entrepreneur, gets called a spamming marketer by people on Twitter, and blurts out all of the stupid / funny things he's ever done in his career on his email newsletter. Follow @derickbailey, and keep up with the latest bloggerings and writing here at derickbailey.com.

## DERICK BAILEY AROUND THE WEB

Twitter: @derickbailey

Google+: DerickBailey

Screencasts: WatchMeCode.net

eBook: Building Backbone Plugins