STUART GRIMSHAW

I'm British so I know how to queue

Long running RabbitMQ consumers with PHP

I’m British So I Know How to Queue

Writing long running RabbitMQ Consumers In PHP.

Stuart Grimshaw

This book is for sale at http://leanpub.com/im_british_so_i_know_how_to_queue

This version was published on 2014-11-19

This is a Leanpub book. Leanpub empowers authors and publishers with the Lean Publishing process. Lean Publishing is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

Tweet This Book!

Please help Stuart Grimshaw by spreading the word about this book on Twitter!

The suggested hashtag for this book is #phprabbit.

Find out what other people are saying about the book by clicking on this link to search for this hashtag on Twitter:

https://twitter.com/search?q=#phprabbit

For my wife & kids.

Contents

Preface

Who is this book for.

This book is for anyone who uses PHP in their software stack and has any kind of back end processes, so pretty much any PHP developer ever … you should all buy it.

The book tells you when & why to use messages queues and the examples provided are based on real world examples of situations where message queues have been deployed, so it's ideal for people looking at message queues as a potential solution for some upcoming project, or anyone looking to learn about queueing for future projects.

Anyone who works in the development of software will benefit from reading this book. Architects looking to understand what message queues can offer, engineers looking for advice on how to develop their message queues. Quality engineers can learn how queues work in order to better understand how they need to be tested. Finally product managers and other stakeholders can get an insight from the first chapter into why the engineers might be recommending queues as a good solution to the business problem you're trying to solve.

Who is this book NOT for?

Most of this book is not for people who don't know PHP, I know that's obvious but I'm going to say it anyway. The technical parts assume you have a basic understanding of how to use composer, know how to install PHP & extra modules, know your way round an IDE or text editor and are comfortable debugging code. This book **is** for people that don't know how install extra PHP modules etc, but are willing to learn how.

Typographic Conventions.

All the examples in this book are written in PHP, sometimes the syntax highlighting is off, but they are all in PHP.

Any method names mentioned in the text will be in a `monospaced` font and will end with `parentheses()`, the code also follows `PSR-2` coding style guide[1].

[1] https://github.com/php-fig/fig-standards/blob/master/accepted/PSR-2-coding-style-guide.md

Key Benefits

Among the many things you will learn from this book, I'd like to highlight these as the key areas:

- Create long running consumers.
- Create robust queueing systems.
- Gain an insight into areas where message queues may be able to help you, outside of the most common problems they solve.

Please help improve this book.

This book is published via LeanPub[2] which means that authors like me can publish books and the people that buy them can keep on getting updated versions as people report bugs and suggest improvements. I think it's a wonderful idea and I really do encourage you to send me your thoughts on the book.

If there's something you don't understand then there's something other readers don't understand, tell me & I'll update the section of the book and make it better for everyone. There are a few places you can contact me, I'm @stubbs[3] on Twitter, or you can send me something via Google+[4] or leave a comment on the Leanpub feedback page[5] for the book, there's even a form to send me an email on that page too.

I really look forward to hearing from you.

About the author.

Stuart has spent almost 20 years as a Software Engineer and Software Engineering Manager, in industries as diverse and the haulage sector and local government via internet service provision, ecommerce and bog data using languages like Visual Basic, Java, Delphi/Pascal, Perl & PHP as well as being a network engineer & sysadmin in between too.

He is currently co-organiser of ShefPHP[6], the Sheffield PHP User Group in South Yorkshire, where he lives with his wife & 2 kids and is an occasional conference speaker.

[2] http://www.leanpub.com
[3] http://twitter.com/stubbs
[4] https://plus.google.com/u/0/+StuartGrimshaw/
[5] https://leanpub.com/im_british_so_i_know_how_to_queue/feedback
[6] https://twitter.com/shefphp

About the cover

The cover art is from a photograph[7] published under the Creative Commons[8] license on Flickr by Petras Gagilas[9]

[7] https://www.flickr.com/photos/gagilas/2210556480
[8] https://creativecommons.org/licenses/by-sa/2.0/
[9] https://www.flickr.com/photos/gagilas/

Why Bother Queueing?

A short history of message queues.

Message queues provide asynchronous communication between two systems, the message sender (or producer as they are known in the RabbitMQ manual) places a message on a queue and doesn't interact with the system that receives & processes that message (the consumer). The message itself will stay on the queue until a consumer is available to process it.

Message queues serve a wide variety of purposes. Linux has at least 2 different message queues available to it (POSIX[10] & System V[11], with System V Inter-process communication (IPC) being the older implementation.) but they only really provide queues for processes on the same machine to communicate.

This book concentrates on using message queueing software, or rather the producers & consumers, rather than IPC message queues. We will use RabbitMQ as our message service, but there are many others, both commercial (such as IBM's Websphere MQ[12], or Microsoft Messaging Queueing[13]) and open source (JBoss Messaging[14], Apache ActiveMQ[15], RabbitMQ[16] & Beanstalkd[17], amongst others). There are even SaaS implementations out there too like Amazon SQS[18], StormMQ[19] & IronMQ[20]. Appendix A looks at implementing some of the open source & SaaS queuing services but the rest of the book uses RabbitMQ.

It's common practice that enterprise vendors attempt to lock their customers to their products in order to make sure that, in future, they continue to buy their products and services. They commonly do so by using proprietary protocols to stop other manufacturers providing an easy migration route and message queueing software is no different. There are two open source protocols that have become popular though, Advanced Message Queueing Protocol (AMQP), which is what RabbitMQ uses, and Streaming Text Oriented Messaging protocol (STOMP). Despite having a cooler name, adoption of STOMP has been slower.

---

[10] http://linux.die.net/man/7/mq_overview
[11] http://man7.org/linux/man-pages/man7/svipc.7.html
[12] http://www-03.ibm.com/software/products/en/websphere-mq/
[13] http://msdn.microsoft.com/en-us/library/ms834460.aspx
[14] http://jbossmessaging.jboss.org/
[15] https://activemq.apache.org/
[16] http://www.rabbitmq.com/
[17] https://kr.github.io/beanstalkd/
[18] https://aws.amazon.com/sqs/
[19] http://stormmq.com/
[20] http://www.iron.io/mq

What can a message queue do?

Message queues provide solutions for a wide range of problems when you're building any new system, that's not to say they're a solution for every problem, but here are a few:

Decouple systems.

The part of your API that listens for incoming requests can be separated from the part that deals with the requests and does all the processing, or separate the process that reads information from a datasource from that which processes it. It can also be used as a global event system so the part of your app that takes signups can be decoupled from the bit that sends the welcome email, creates accounts and takes any initial payment.

Easy to scale.

Message queues are not limited to a single consumer, so a quick way to increase throughput is to add more consumers. Consumers can be on a remote machine too, so if you've somehow managed to max out the capacity on 1 box, you can spin up another and deploy your consumers to it in parallel to the existing ones.

Redundancy.

In a scenario with no message queue, if something goes wrong while you're processing data, then the chances are it's lost. Messages are left on the queue until the queue software receives an OK from the consumer to say that the message has been processed, so any failure of the consumer does not lose the message. This guarantees that as long as there is at least 1 consumer working the queue, a message will be processed eventually.

Resilience.

Every single reader of this book will have been in a situation where you've had to take an entire site or system offline while you deploy some updates to another part of the system. Changes to a payment provider means you have to stop signups, you can't receive API calls because the email process is down for an upgrade. Using a message queue between different areas of your system means that you only need to have downtime on the area that's being updated.

Some real-world examples.

User sign up.

When a user signs up for your service, you will usually perform numerous actions, traditionally in sequence, one after the other. You may send them an email to confirm or take a payment of some

sort, create them an account on a 3rd party support app and record their details in your database. None of these things really need to happen sequentially, there's no piece of data from one of these actions that is required by another. You might want to create an entry in a support agent's diary to check up on the customer after a few days and, log the fact that you have a new signup and finally prompt an update of the company dashboard.

Using message queues you can achieve all of this with one message. RabbitMQ lets you specify what's called a "fan out" configuration where a single incoming message will end up on several queues and be processed by different consumers. If one of the different items fails it's much easier to correct the problem and put the message back on the right queue and complete a single action than to kick off the user creation process midway through.

We'll look at 'fanning out' messages later on when we look at different ways to configure RabbitMQ.

"Internet of Things"

It's a long running complaint from many home automation enthusiasts that while there are many many awesome and sweet devices to plug your home into, they're all data silos and nothing they collect or do is easily shared between other devices. You could solve this problem by having each device post event data to a message queue and using the fan out configuration again have other devices react via consumers.

Imagine you have a Nest thermostat[21] downstairs and you tell it to set the room temperature to 20C. Nest produces a message[22] with that information on it, and your queue config fans that message out to a logging queue, a speech queue that announces the change, a queue that closes your windows because you don't want to waste the heat & a consumer delivers the message to your Apple Home Kit[23] enabled app to bridge the gap in between two silos.

How about you want to go to bed, so you issue a voice command via your RaspberryPI running lightweight voice recognition[24], it produces a message that contains the command and it's picked up by a consumer that switches on the upstairs lights, this produces a message with a TTL of 10 minutes so that when the message hits your consumer you're tucked up in bed and the lights go off & the downstairs alarm zone is activated.

Billing system queues billable events.

If you're a phone company that charges people by the minute for usage of your service, then you need to be sure that records of calls are recorded accurately. Using the redundancy of message queues you can be sure that the call record will be consumed and the amount deducted from the clients account balance. You can easily scale the consumption of message by adding more consumers, and fan out

[21] https://nest.com/

[22] https://nest.com/works-with-nest/

[23] https://developer.apple.com/homekit/

[24] http://www.raspberrypi.org/meet-jasper-open-source-voice-computing/

the initial message to be consumed by systems that generate itemised bills, watch for fraudulent patterns on accounts, log system events for support and send warning emails when the account balance drops below a certain amount.

Nokia HERE (Traffic application)

Nokia uses a message queue to receive data points from various sources all around the globe. These messages originate from fixed sensors, toll tags, trucking logistics systems and manually entered traffic incident reports (crashes!) amongst many others. All this data is processed in real time so that the traffic reports they give are accurate, and not for what the traffic was like an hour ago. The system processes 800,000 messages per minute, or just over 13,000 messages per second.

HERE uses consumers to model & transform the incoming data before delivering on to the various products that ultimately consume it, the same message may end up in several products. These consumers are very small & focussed on doing their single job, that's one of the ways that the system is able to consume so many messages.

You can read a full account of Nokia's HERE architecture here[25]

[25] http://blog.gopivotal.com/pivotal/case-studies-2/800000-messagesminute-how-nokias-here-uses-rabbitmq-to-make-real-time-traffic-maps