

Extracted from:

# Ruby on Rails Background Jobs with Sidekiq

Run Code Later without Complicating Your App

This PDF file contains pages extracted from *Ruby on Rails Background Jobs with Sidekiq*, published by the Pragmatic Bookshelf. For more information or to purchase a paperback or PDF copy, please visit <http://www.pragprog.com>.

Note: This extract contains some colored text (particularly in code listing). This is available only in online versions of the books. The printed versions are black and white. Pagination might vary between the online and printed versions; the content is otherwise identical.

Copyright © 2023 The Pragmatic Programmers, LLC.

All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form, or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior consent of the publisher.

The Pragmatic Bookshelf

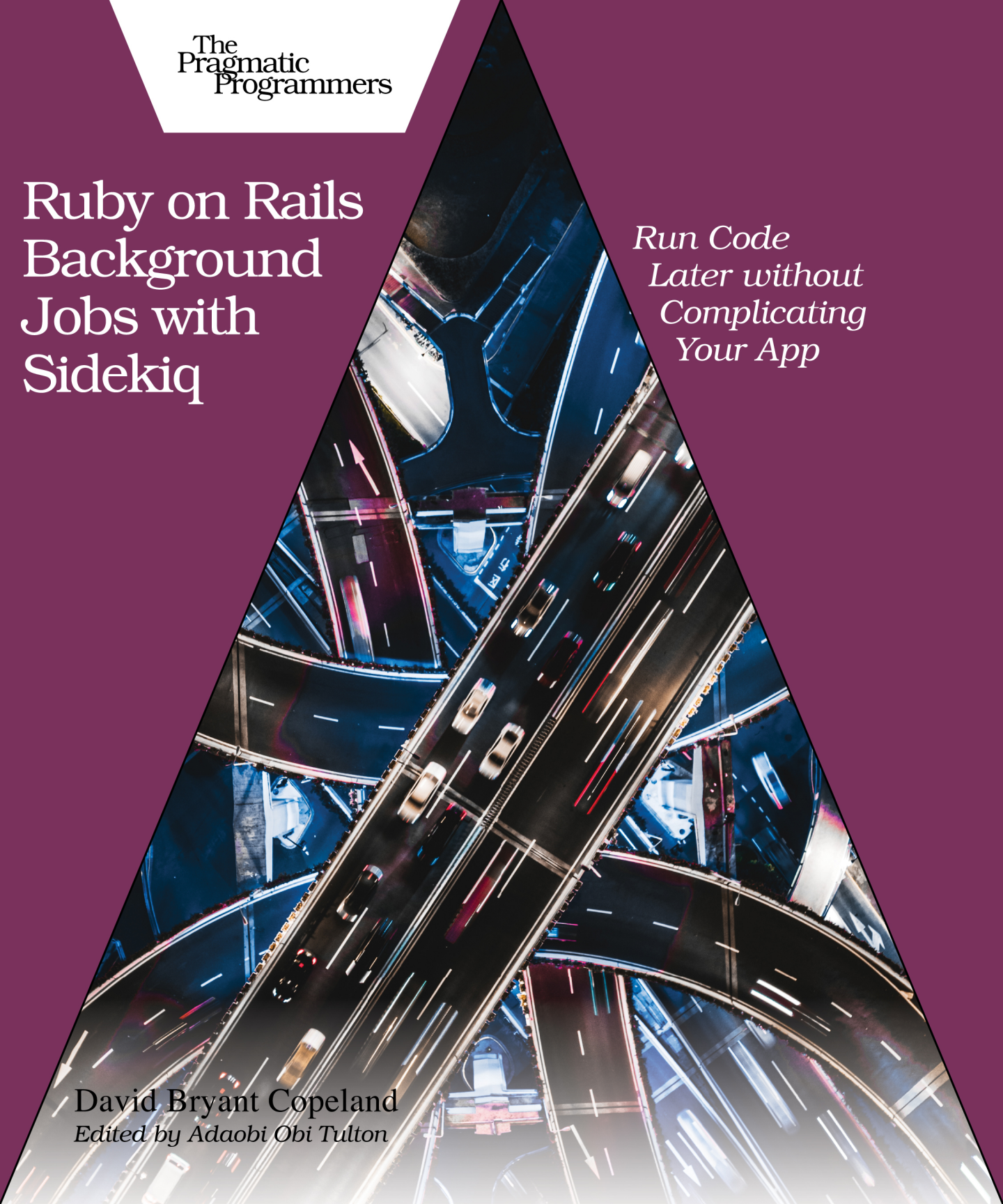
Dallas, Texas

The  
Pragmatic  
Programmers

# Ruby on Rails Background Jobs with Sidekiq

*Run Code  
Later without  
Complicating  
Your App*

David Bryant Copeland  
*Edited by Adaobi Obi Tulton*





# Ruby on Rails Background Jobs with Sidekiq

Run Code Later without Complicating Your App

David Bryant Copeland

The Pragmatic Bookshelf

Dallas, Texas



Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and The Pragmatic Programmers, LLC was aware of a trademark claim, the designations have been printed in initial capital letters or in all capitals. The Pragmatic Starter Kit, The Pragmatic Programmer, Pragmatic Programming, Pragmatic Bookshelf, PragProg and the linking *g* device are trademarks of The Pragmatic Programmers, LLC.

Every precaution was taken in the preparation of this book. However, the publisher assumes no responsibility for errors or omissions, or for damages that may result from the use of information (including program listings) contained herein.

For our complete catalog of hands-on, practical, and Pragmatic content for software developers, please visit <https://pragprog.com>.

The team that produced this book includes:

Publisher: Dave Thomas

COO: Janet Furlow

Managing Editor: Tammy Coron

Development Editor: Adaobi Obi Tulton

For sales, volume licensing, and support, please contact [support@pragprog.com](mailto:support@pragprog.com).

For international rights, please contact [rights@pragprog.com](mailto:rights@pragprog.com).

Copyright © 2023 The Pragmatic Programmers, LLC.

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form, or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior consent of the publisher.

ISBN-13: 979-8-88865-036-3

Encoded using the finest acid-free high-entropy binary digits.

Book version: P1.0—October 2023

## Handling Permanent Failures via Monitoring

If a job has failed and will never succeed, it stands to reason that the only way to fix the underlying problem is first to become aware of it. An extremely common way to do this is to configure an *error catching service* (or *error catcher*), like Honeybadger<sup>1</sup> or Bugsnag.<sup>2</sup>

An error catcher is a service that integrates with your app to receive notifications of any unhandled exception, including those from your Sidekiq jobs. The error catcher then notifies you about the exception via email or an alerting system like Pager Duty. The idea is that you don't have to proactively check for failed jobs; instead, you allow the error catcher to notify you if a job fails. The example app you set up includes a mock error catcher so you can see how it works without having to sign up for a real service and manage that integration.

To connect Sidekiq to an error catcher, you'll need to configure an *error handler*. Sidekiq's configuration provides the attribute `error_handlers`, an array of Proc objects that are called when an unhandled error is caught. Our mock error catcher has a client library class included in the example app `ErrorCatcherServiceWrapper`, which has a method named `notify` that will send the exception to the service.

To set this up, you'll need to create an initializer for Sidekiq in `config/initializers/sidekiq.rb`. It will look like the following code, with the Proc configured for the error catcher:

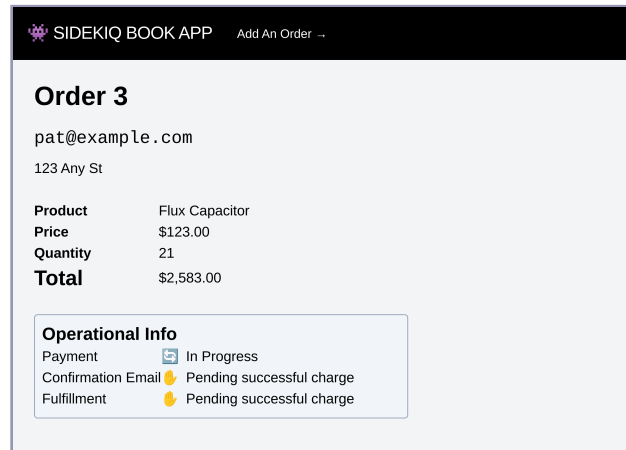
```
snapshots/2-1/sidekiq-book/config/initializers/sidekiq.rb
Sidekiq.configure_server do |config|
  config.error_handlers << ->(exception,context_hash) {
    ErrorCatcherServiceWrapper.new.notify(exception)
  }
end
```

Let's see this in action by introducing a permanent failure. Add a line of code to raise an exception at the start of `OrderCreator`'s `complete_order` method, which is what `CompleteOrderJob` calls:

```
snapshots/2-1/sidekiq-book/app/services/order_creator.rb
def complete_order(order)
  ➤ raise 'forced error to demonstrate error-catching'
  payments_response = charge(order)
  if payments_response.success?
```

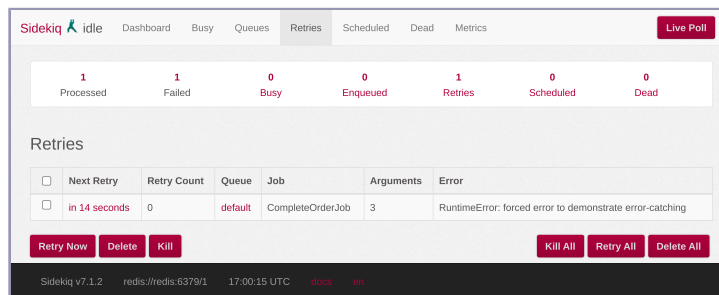
1. <https://www.honeybadger.io>
2. <https://www.bugsnag.com>

Now, create a new order using the example app. After doing so, you should be redirected to the new order's show page, but even after reloading that page, the order should still indicate that payment, email, and order fulfillment are in progress, as shown in the next screenshot.



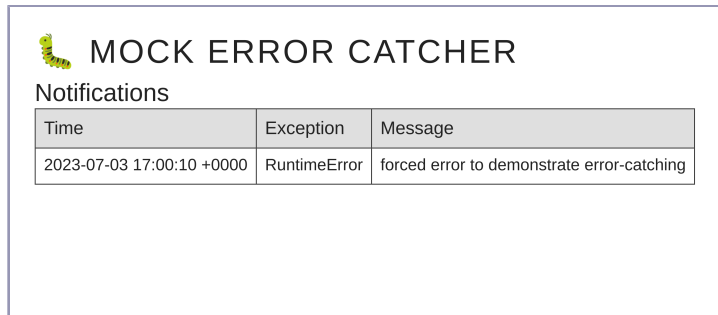
A screenshot of the order show page that indicates that the payment, notification email, and order fulfillment request are all pending.

Now, Go to the Sidekiq Web UI at <http://localhost:3000/sidekiq> and click on “Retries.” You should see that the CompleteOrderJob is there, along with the error message you added to complete\_order. Sidekiq will also show you when it's planning on retrying the job, as shown in this screenshot:



A screenshot of the Sidekiq Web UI's Retries page. It shows a single job's class and error message, along with an indicator of when the job will be retried.

The mock error catcher has a Web UI you can examine to view the notifications it has received. You can see it at <http://localhost:3001>, and it should look like the shot below. If you see a notification here, a real error catcher would've notified you.



The screenshot shows a web interface titled "MOCK ERROR CATCHER" with a small green caterpillar icon. Below the title is a section labeled "Notifications" containing a table with the following data:

Time	Exception	Message
2023-07-03 17:00:10 +0000	RuntimeError	forced error to demonstrate error-catching

A screenshot of the mock error catcher that shows a table with entries for all the retry attempts. Each entry shows a timestamp, the exception class name, and the exception’s message.

Now, undo the syntax error:

```
snapshots/2-2/sidekiq-book/app/services/order_creator.rb
```

```
def complete_order(order)
  # removed forced error
  payments_response = charge(order)
  if payments_response.success?
```

Go back to the Retries section of the Sidekiq Web UI. If the job is still there waiting to be retried, click it, then click “Retry.” You should be returned to the Retries section and not see anything there—the job succeeded! If you refresh the order show page, you should see that all the operational stuff has completed properly.

What’s important to take away here is that if you had not been notified about the job failure, you wouldn’t have known to fix the underlying problem and the job would never have succeeded. This mechanism of being notified of failure works extremely well...until you start getting transient failures, which are far more common.