Hakyll on Netlify

Justus Perlwitz

2019-09-01

Contents

1 Why I never write any posts

If you are like me, you are more busy switching between static site generators than actually writing posts for your blog. If you take a closer look at the commit history for this website, you will notice that I have switched site generators several times (including to and away from my own site generator) But who can resist the appeal of yet another static site generator that promises some combination of

- easy content editing
- ridiculously simple deployment
- blazing fast build times
- a vibrant plugin ecosystem that will support every flavor of Markdown
- easy to extend
- sane configuration that works for everyone

This time I have chosen a fabulous site generator toolkit called hakyll. Hakyll is a static site generator implemented in Haskell. It uses a DSL for configuration and allows its user to build their site in almost any way they like. Completely custom URL schema? No problem. Compile the same file using several different compilers, including Pandoc? No problem. Compile to one easy to deploy static executable, including all your configuration? No problem.

Now recalling the advantages that every shiny static site generator out there promises to have, we quickly notice that Hakyll has

- easy content editing – only when set up correctly,
- ridiculously simple deployment – once you manage to correctly install and master Haskell and Stack
- blazing fast build times – if you ignore that a clean build of the generator itself will take 15 minutes
- no vibrant plugin ecosystem – but a vibrant ecosystem of tutorials and snippets in the lower 100's range,
- easy extensibility – if you are willing to put in the extra effort of learning what Monads are, since they're used everywhere, and a
- sane default configuration – which only does 20 % of what Jekyll can do out of the box (like pagination, liquid templates, data files).

As you can see, there are a lot of places where Hakyll can only shine once you spend a considerable amount of time configuring everything. But what might sound like a big drawback is actually Hakyll's biggest advantage. Not many site generators give you the ability to make something tailored exactly to what you are doing. The next level of course would be to make your own site generator.

And, this is what I've done previously. My blog once was a Flask site turned into a bundle of static HTML using Frozen Flask. So, if Jekyll is on *one end* of the site generator spectrum – thanks to its batteries-included approach, then building your own static site generator in Python is clearly on the *other end* of the spectrum, even

with the advantage of using powerful Jinja 2 templates inside Flask. You are forced to do everything yourself. That includes the boring stuff such as watching your Markdown files and other assets and performing live rebuilds for local testing.

Of course, creating your own site generator is a valuable experience. It can teach you a lot about how to design a good build system. At the same time, we often need some more complicated features that just take a lot of time to implement from scratch by yourself. For example, one of those advanced features is tight integration with external tools and using those tools to provide several build versions of the same blog post that is written in Markdown. Exactly this is what ultimately led me to choose Hakyll for my personal website.

My ideal workflow for blog posts and other pages on this site would look as follows

1. Write a rough draft of what I want to say in Markdown.
2. Start a live build version and check how my initial draft *feels* on screen.
3. Create illustrations and charts as needed to illustrate my point, while
4. continuing to edit and write my post.
5. Publish by doing a `git commit` followed by `git push`.

With Jekyll or any other popular static site generator especially the last step is simple. Netlify support Jekyll quite well and can be set up within 10 minutes. Here, I am not willing to compromise and I would like to keep things simple and reliable. Where I always felt Jekyll was lacking, was in external tooling support. I am a big fan of Pandoc, Graphviz and mscgen. Pandoc allows me to use the best of Latex, while Pandoc filters allow me to easily add external tools, including Graphviz and mscgen.

For a while I used jekyll-graphviz, and it worked fine. Furthermore, there is a second plugin for Jekyll, that allows you to use Pandoc. But it doesn't go so well with jekyll-graphviz, and I really want to avoid using any non-Pandoc-syntax. And Jekyll plugins mostly work by creating custom liquid tags, so that I would need to type

```
{% graph some graph title %}
a -- b
b -- c
c -- a
{% endgraph %}
```

every time. The preferred way for Pandoc filters would be more like this:

```
~~~
digraph G {Hello->World}
~~~
```

(Source)

2 Enter HASKELL and HAKYLL

At some point when you go down the customization road, you will discover that you need something that was made with *extreme* customizability in mind. That means, no implicit configuration, a better configuration language, and ideally something that blends configuring the site with writing your own customizations.

Haskell's approach to this has always been using Domain Specific Languages (DSLs). On the one hand, you have a library of words that can do a few things really well, but since on the other hand you are writing plain Haskell, you can drop into your own custom code anytime you want. Jekyll, which is based on Ruby, would have you drop your plugin files in certain folders, and that would work fine, but the degree of *implicitness* was always nagging me.

Then, we should also consider that Pandoc just *happens* to be written in Haskell, and there is this wonderful static site generation toolkit called Hakyll. So after some careful pondering, I decided to venture into static site generator customization land for the fifth time or so.

And here we are, roughly 2000 lines (delta) of fresh code later. I started off using the sample Hakyll configuration and I've added a few things:

- Automatic table of content generation using Pandoc, thanks to Gwern. My implementation can be found here.
- Teaser extraction, thanks to this tutorial. My implementation can be found here here.

- Sitemap generation, thanks to this tutorial. My implementation can be found here.
- A Pandoc graphviz filter, thanks to this Pandoc filter. My modifications can be found here.
- A Pandoc mscgen filter, inspired by the above filter, to be found here.
- A Netlify deploy using CircleCI, as implemented here

What does all of this buy me? Extremely fast site building speeds. The static site generator code for Hakyll and Pandoc are compiled into one executable, including all the filters that I've defined. Whereas in Jekyll, using Pandoc with filters would have meant Jekyll spawning a Pandoc process per document, and each Pandoc process spawning a filter configured via the `--filter` option. Here, the main cost is not forking itself, but the startup time for each subprocess.

Then, in the Hakyll version I've just created one big switch statement that checks what to do when it encounters a "‘ block:

```
renderAll :: Block -> IO Block
renderAll cblock@(CodeBlock (id, classes, attrs) content)
  | "msc" `elem` classes =
    let dest = fileName4Code "mscgen" (T.pack content) (Just "msc")
     in do ensureFile dest >> writeFile dest content
           % Do mscgen things here
  | "graphviz" `elem` classes =
    let dest = fileName4Code "graphviz" (T.pack content) (Just "dot")
     in do ensureFile dest >> writeFile dest content
           % Do Graphviz things here
  | otherwise = return cblock
  where
    image img =
      Para
        % Insert the image here
renderAll x = return x
```

The full code can be found here. If I ever want to add any new diagramming tools, I can just extend the `renderAll` function.

Then, the question remains: Why bother with all of this?

3 Writing in Freedom

Every few years or so, a blogging platform announces itself and promises to be the one place where writers can gather and exchange stories, while unburdened with paywalls, dubious content licensing, sudden changes in content policies, and so on.

So far, every single blogging platform has disappointed in some way. Tumblr has suddenly decided to strictly forbid all adult content. Medium started adding paywalls and pesters users even when reading free articles.

When I write text and publish it, then I express my opinions and views. I take some time off my day to sit down and share my thoughts with others, hoping that I will be able to find someone out there with whom I can engage in a deep conversation. I am convinced that engaging in a meaningful conversation requires freedom. Freedom from any platform agenda, freedom from the pressures of capital, and the freedom of knowing that your opinions and that what you express can reach others exactly as you intend.

If I can influence how my content is rendered, what the impression people have when they reach my site, and ultimately what ideas will stick with my audience, then I feel I have freedom. This freedom of course also comes with the downside that if I'm clumsy at designing my site, not optimizing for usability or accessibility, if my site goes down, then my ideas will have been expressed in vain. The upside though, makes it worth it. If I can make my site work and reach someone, I have reached my goal.