Going static

The "why"s for writing one's own static generator, which I want to replace my WordPress with.

> You went full static. Never go full static. - famous webmaster quote

Why?

A long, long while ago, somewhere in 1999, I used to handcraft my HTML. Soon enough I learnt the existence of Microsoft FrontPage[^1] and I was very happy with that for a while. *For those who never used it: FronPage was a static generator, which combined HTMLs as well, so the end result was a bunch of HTML pages.*

Hosted blogging was never really for me; I was barely generating status updates or text in the beginning. Later I moved on to writing entries, which I now refer to as "notes", but I still preferred to do "visuals" instead of text content. So by the end of 2004, I moved on to hilarously dumb PHP files to handle these includes, and from there on, my site was always generated on the fly. Those scripts eventually resulted in a custom CMS, which was in use for a few years. I've never open soured it, for which everyone should be thankful, looking back at the code from today.

Around 2007 I made up my mind and moved on to WordPress[^2]. It was a reasonable choice, WP was much more mature than my little CMS was and I loved it. Somewhere, at a certain point in time, I wrote a full page cache plugin[^3] to solve speed issues with WordPress, not realizing I'm making a temporary static version of it. During the years, I've added numerous plugins to fix broken (in my opinion) or missing things in WordPress.

A few months ago I started looking into things like IPFS[^4] and archiving my virtual things. It made me realize how fragile my system is: that making it always on the fly, requiring a database, a PHP server, a web connection, and so on, is just making it too easy to fall apart. Moving it from one server to another is painful, with all the insfrastructure required behind it. *I could do containers, but I'd still need to build my own container, and that's not much better than writing some ansible setups, and it doesn't really get me anywhere.*

But I still had WordPress, with all of it's features I always took for granted, like search. I decided to try a hybrid approach: keep WordPress, but have text-ish version of my site, which can be archived.

To address the problem, I first moved my posts from HTML to markdown. While this is not a trivial win for many, it is for me: the HTML generated partly by me, party by the WordPress TinyMCE, when I was still using that, was horrible. I first used Parsedown Extra[^5] and moved to Pandoc[^6] later, but since this latter is pretty slow, I had to add intermediate caches, based on the hash of the text.

I've removed a lot of things from my (written from scratch) WordPress theme and moved it to Twig[^7] templating. I wrote yet another WordPress plugin[^8] to export my site into a YAML frontmatter + Markdown format. I also wanted to generate a static HTML version, but that approach failed. Even though everything was already Twig based, I could not get taxonomy generation working from code, only by actually visiting the URL and catching the output into a file before serving it to the user.

I slowly realized this is getting messy, but I still wasn't ready to leave WordPress behind. The turning point was when I wanted to move the generated images into a cache folder to be separated from the original uploads. It all went well on the frontend, using the relevant hooks - but the admin interface completely ignored it. Digging into the code for a few hours told me, that the admin interface really ignores it: it simply doesn't call those hooks. At that point the amount of frustration with WordPress enraged me enough to start looking at static generators[^9] seriously, for the first time ever.

Somewhere around this time, the Brighton IndieWebCamp 2016 happened[^10] and we had an interesting session: Going Static[^11], which further verified my intentions of using a static generator.

Why write yet another one?

During the years I developed some "special" needs. I'm storing tags, content, title, etc. inside my photo post JPG files as EXIF, IPTC and XML. Yes, I had another WordPress plugin to copy this automatically[^12] into a post, and I wanted to keep this "lazy" feature. Unfortunately no existing systems supported this. Extending one or adding a plugin would have taken the same amount of time as writing a new one, and this way I have full control. I'm trying not to reinvent all the wheels by looking at other generators' code though.

Also, writing a static generator is not that hard.

Why Python 3?

Because I wanted to learn it.

I might rewrite the whole thing in another language, sometimes. Maybe.

Approaches

Markdown, Pandoc, YAML Frontmatter

There are arguments surrounding Markdown[^13]; some of them I consider valid criticism, others, I don't. Early Legion of Doom magazin releases[^14] are prime examples how plain text formatting can be really neat and that they can withstand the test of time, but they lack a feature: reflowing. I consider reflow a feature, not a bug. They are also text only, without real images included (I love ASCII art, but those are not images). Altogether Markdown is a compromise and, if used well, it delivers the original promise of easy to read.

Pandoc is a universal converter, which understands nearly all dialects of Markdown, such as fenced code blocks. It had been around for 10 years, and if your format is understood by it, you can convert it to... well, a lot of other formats.

I've settled with the following rules, apart from the common Markdown:

" `language` " type code blocks

Pandoc has built-in syntax highlighting, so if the language is present, I don't need anything else to make code look pretty, apart from a bit of CSS.

footnotes instead of links

The regular way of linking is broken for prints. One of the most common tricks to bypass this is to force CSS to show the link URL right next to the linked text. Since URLs can get long, this completely breaks readability. Due to this, I've moved on to use footnotes, with the link in the footnote.

definition lists, subscript, superscript, strikethrough

Definition lists are one of the oldest features of HTML[^15], yet the original Markdown completely ignores it. I don't. Same goes for the others.

Originally I wanted to use raw Markdown; no front meta, no alien format. Meta blocks with headers, like "Author", followed by lines of name, email, web, etc. This turned out to be a little tedious to maintain, since each block would needed their own rigid regexes to parse. Eventually I've settled with the YAML block before the Markdown content, which seems to be the de facto standard for statig generators.

Dropping font based icons and using inline SVG instead

I've first used @font-face when only IE6 supported it, for little while, somewhere in, maybe, 2001; since then, times changed, and I came to use icon fonts. They are flexible, easy to intergrate, can be used in CSS as `content`, but they come with a downside: you need at least 4 external files (choose freely from woff, woff2, ttf, eot, or svg).

There are services, like Icomoon[^16], with which you can create your own set instead of loading everything but even like that they might end up eating a significant amount of bandwidth[^17]. And though they might be small, they will still be external references.

At a certain point I tried to switch to SVG icons, but my approach was wrong; these are not drop-in replacements. I ended up with some css hacks and in-HTML SVG icons, but I finally got rid of those external fonts.

```html
<svg class="hide" version="1.1" xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink">
    <symbol id="icon-gitlab" width="16" height="16" viewBox="0 0 16 16">
        <path d="M0.929 6.304l7.071 9.063-7.75-5.625c-0.214-0.161-0.304-0.438-0.223-0.688l0.902-2.75zM5.054 6.304h5.893l-2.946 9.063zM3.286 0.839l1.768 5.464h-4.125l1.768-5.464c0.098-0.277 0.491-0.277 0.589 0zM15.071 6.304l0.902 2.75c0.080 0.25-0.009 0.527-0.223 0.688l-7.75 5.625 7.071-9.063zM15.071 6.304h-4.125l1.768-5.464c0.098-0.277 0.491-0.277 0.589 0z"></path>
    </symbol>
</svg>

<p>
    <a href="https://gitlab.com">
        <svg class="icon"><use xlink:href="#icon-note" /></svg>
        GitLab.com
    </a>
</p>
```

```css
svg {
    -ms-transform: rotate(0.01deg);
    -webkit-transform: rotate(0.01deg);
    transform: rotate(0.01deg);
}

.icon {
    width: 1rem;
    height: 16px;
    display: inline-block;
    fill: currentColor;
    vertical-align: text-top;
    overflow: visible;
}
```

Dropping LESS[^18] and inlining CSS

In 2014 I moved my CSS to LESS. I wanted to do it for a long time: less code, better structure, and only a minor downside: I need to convert. I used a PHP converter, lessphp[^19], which worked stable and sound. When I started my move to Python all the converter libraries failed to parse my less, mostly due to whitespace handling.

Eventually I got angry enough to drop the whole LESS thing and rewrite the styling from scratch, copying as little as possible from the previous implementation.

Once this was done I decided to inline the CSS as well, just like the SVG. My blog doesn't have enough traffic for the maybe cached CSS file to exist, and, to be honest, this is an essential part of the site, so it makes sense to be a part of it.

Putting (minor) logic in templates

WordPress has a lot of templating-like functionality in functions, for example, pagination. `<?php echo paginate_links( $args ); ?>`. Simple. Not really portable though. So deep dived into the internets and I made a Jinja2 template to handle pagination, based on other founds[^20].

Proper print support

I have no idea how computer storage will be in ten, fifty, or a hundred years. The current, in theory, most reliable thing you can get, is Blu-ray MDisc[^21]. *This is a little*

too expensive for me, so I've settled with regular, single layer, Panasonic Blu-ray for archives and disaster backups. However, there is another way of archiving text and photos: printing them.

Archival prints are different beast; they require special paper, usually special tint, but in order to print, I need a formatting that can be printed well. I should be doing this, though the physical space it may require is problematic, but to be able to do this, I first had to set up print support in my CSS. With ideas from here and there[^22], I've made my photos not to break midway on a page and not to occupy complete pages. The footnote URLs were also part of print optimization.

Webmentioning outgoing links

On build, each and every link I have in an articles gets pinged and goes into a "database" - a rather big JSON file - to avoid repeating this on every build.

Using brid.gy[^23] to POSSE[^24]

I've been trying to minimize my use of social networks, or at least to think about posts there as ephemeral, one-to-many messages instead of posts. *This also means I manually delete what I can, but knowing about Facebook's Blu-Ray system[^25] makes me wonder why are they allowed to use the word "delete" at all.* I still, sometimes post things, and for that, I need to thank brid.gy: a simple, formatted link and a webmention, and the post is present remotely as well.

Short URLs

There was, again, a WordPress plugin[^26] I had to generate base32 (0-9a-z) short slugs for every post, based on their publish date. In case of missing titles, these also acted as the main and only slug.

This plugin was built to utilize the routing built into WordPress, so I had to solutions to do this in the static manner: either generate an nginx config and reload nginx on build, or generate HTML files with refresh meta tags to redirect. I went with the second - I want static, so let's keep it away from dependencies.

The non-static parts

Search

There is a functionality on my site which, for now, I don't want to outsource: search. WordPress comes with tolerable built-in search, so I had to put together my own to replace it.

It turned out to be easier than I anticipated. The web endpoint is handled by Sanic[^27] (yet another python web library, but this is embracing the built-in asyncio) the indexing and searching by Whoosh[^28]. The index is populated on build and used by the service after that and the results are rendered on the fly with a very simple template in Jinja2.

Receiving webmentions

I wanted to keep receiving webmentions as my own service, just like search. Especially since by default, I forward webmentions as emails, which I had to write for myself.

They are handled by the same Sanic service as a different routing.

Scheduled posting

WordPress managed it scheduling by WP Cron, which is a monster of it's own. To mitigate it, I've been triggering WP Cron every minute from the system cron. This is how scheduled posts and scheduled tasks were executed.

In the *nix land, there is a utility, called at[^29]. It's a simple little utility, which accepts text and number based "when" and a list of commands to do at the time. It makes it easy to trigger non-recurring builds an schedules at certain times.

For everything else, there's cron[^30].

What's still missing

Obviously, I don't have an admin interface any more; it's all command line at the moment. I wrote a little wrapper which accepts arguments, so it's scriptable, but most probably I'll need to make a Micropub[^31] endpoint once. I also want to make it possible to post by email, to write a little fetcher, which pulls in mails from an account and converts them to posts. I have email on every device and it's still simple enough to use; it should make an obvious interface for posting.

Right now the whole process of generating the site is not sophisticated enough; it will need a lot of optimization. It's also not too fast.

There are the unknown unknowns: the ones I took for granted from WordPress and yet to figure out, they are missing. And of course, bugs will need fixing.

Show me the code!

I'm not truly proud of it yet. My Python knowledge is fresh, therefore it's terrible. Once I feel like it's somewhat ready, it'll be on Github.

Links

1. https://en.wikipedia.org/wiki/Microsoft_FrontPage
2. https://wordpress.org
3. https://github.com/petermolnar/wp-ffpc
4. https://ipfs.io/
5. http://parsedown.org/
6. http://pandoc.org/
7. http://twig.sensiolabs.org/
8. https://github.com/petermolnar/wp-flatexport
9. http://www.staticgen.com/
10. http://indieweb.org/2016/Brighton
11. http://indieweb.org/2016/Brighton/goingstatic
12. https://github.com/petermolnar/wp-photo2content
13. http://indieweb.org/markdown#Criticism
14. http://www.textfiles.com/magazines/LOD/lod-1
15. http://info.cern.ch/hypertext/WWW/TheProject.html
16. https://icomoon.io/app/
17. https://meowni.ca/posts/web-fonts/
18. http://lesscss.org/
19. http://leafo.net/lessphp/
20. http://dev.dbl-a.com/symfony-2-0/symfony2-and-twig-pagination/
21. http://www.mdisc.com/
22. https://adactio.com/journal/1121/
23. https://brid.gy
24. http://indieweb.org/posse
25. http://datacenterfrontier.com/inside-facebooks-blu-ray-cold-storage-data-center/
26. https://github.com/petermolnar/wp-shortslug
27. https://github.com/channelcat/sanic/
28. https://pypi.python.org/pypi/Whoosh
29. https://linux.die.net/man/1/at
30. https://linux.die.net/man/1/crontab
31. http://indieweb.org/micropub