Why I moved to Next.js

Saying goodbye to Gatsby

Written by Seth Corker for Benevolent Bytes blog

Banner uses photo by Kadarius Seegars on Unsplash

I've been using Gatsby on my blog for a while now. A framework that uses React, offers a familiar GraphQL API and generates performant static websites. It's a solid choice which has allowed me to get up and running quickly but, over the past couple of years, cracks have started to appear. Over the holiday period, I finally decided that I'd had enough. I've now moved to Next.js.

Let's take a step back and walk through why I chose Gatsby in the beginning and why I made the decision to move away from it in 2022.

Gatsby in the beginning

I started out with Ghost and was using a pretty standard template. This suited my needs for a while but updating the template became tedious, whenever I wanted to add new features, I'd wade my way though handlebars templating and adding

interactive elements proved challenging. I needed something a bit more flexible, which is where Gatsby came in.

Why did I pick Gatsby after Ghost?

The main reason was ease of use. Familiarity and the developer experience of React allowed me to create my own blog with the features I wanted much more quickly than I could in a template/theme. The blog starters featured on Gatsby were great, and I could add features as I required them or just as easily take them out.

A side effect of moving to Gatsby was my content moved from a database to markdown files in the same repo as the website. The experience of editing files in Markdown was nice and allowed me to experiment. My structure could be defined in front matter and is rendered out to static pages. I could even write new posts, create PRs and use Git as my publishing flow. The final thing that won me over is the plugin ecosystem that Gatsby touts.

Plugins

The power of Gatsby *is* the plugin system. Essentially, anything you need to do is a plugin. In my Gatsby blog I used plugins to read markdown, setup syntax highlighting for code blocks, automatic embeds, generating a table of contents per post and much more. The majority of my use cases were once handled by plugins within the ecosystem and just required a bit of configuration. This lead to being able to add features rapidly and create a much richer experience than I could if I had to create everything myself.

Performance

One of my favourite features is static site generation (SSG). It means I can create interactive components where needed, and the resulting pages are output as static files that can be hosted anywhere. With Ghost, I had a server hosted on Digital Ocean, but now I just needed a GitHub repo and Netlify. The resulting static website is very fast and images are compressed and optimised to make the overall experience snappy. So if everything was good, why did I migrate to Next.js?

Cracks appearing

I stayed with Gatsby for a long time. Even though issues came up frequently, I'd usually find workarounds and fixes. Over time, the problems got to the point where I'd had enough, so I spiked an experiment to see if I could migrate to Next.js. These are some of the issues that made me stop using Gatsby on my main blog.

Slow builds

Build times weren't really on my radar when I started my blog. I thought that a few posts wouldn't pose much of a problem, and a few minutes was acceptable. Nowhere near the speed of Hugo, but for my small blog – I didn't mind.

As I added more plugins, more images and more complex queries, the build times started to become more unwieldy. I remember spending a few days just on optimising the queries and seeing if there were tweaks I could make or features I could remove to speed up builds. This isn't something I wanted to dedicate time to when I could put that time to better use, like writing more posts.

The last Gatsby build, before moving to Next.js, was 12 minutes. When using a platform like Netlify on the free plan, a few posts a month would use 10% of the build minutes, but any minor bug fixes would also start to add up. When testing a migration between version 2 and 3. I kept running into issues and having to deploy fixes. Pretty soon, I'd exhausted my allowance. This happened a few times when releasing larger features, or one time when I introduced a bug that would cause a build failure after 15 minutes.

Difficult migrations

Updating Gatsby became a bit of a chore. The migration from version 2 to 3 was quite painful. I wanted the added benefits of faster build times, as discussed above, but there was a lot of work that needed to be done. The hardest part was getting the plugins to work. Some underlying APIs had been deprecated, which meant many plugin authors had to update their plugins to work with version 3. This took a lot of testing and in some cases swapping the plugin out with something else or removing it entirely.

Another more frequent problem was updating plugins themselves. I had issues upgrading some plugins between patch and minor versions. The plugin I used for syntax highlighting would break and cause the whole page to break. I could never figure out the issue because the error messages were incredibly cryptic. Other

times, the build would pass, and I'd rejoice, only to find that the highlighting had failed for a particular language.

Overall, Gatsby and the plugins I used had saved a lot of time setting my blog up, but maintaining them became increasingly time-consuming.

Plugin purgatory

Plugins are Gatsby's greatest strength, but they are also the biggest challenge.

Combining plugins in a way that works as you'd expect can be challenging, especially when dealing with transforming input like Markdown or, in my case, MDX. I would encounter odd scenarios where plugins would conflict with each other, resulting in errors and invalid markup. I often had issues with combining multiple plugins to transform MDX, only for a cache reset to resolve the problem.

Plugins are as powerful as the author makes them. In some scenarios, I wanted to use a plugin but with an exception, like when embedding content in posts. If the author didn't allow for this in the plugin options, you are forced to find creative workarounds or fork the plugin.

These small issues evolved into blockers which mounted to the point where I had enough.

My experience with Next.js

I was able to get the same functionality that I had in Gatsby in a couple of weeks with Next.js. Although I haven't released any new features, making changes is significantly easier, and I'm working on migrating to TypeScript to add safety and improve developer experience.

The migration

The migration went relatively smoothly, but the biggest hurdle was data fetching. In Gatsby, I simply pointed a source plugin at the MDX folder and said which page template to use. Gatsby would take care of the rest and create a nice GraphQL API I could query and experiment with.

In Next.js, things are a little more manual.

Loading MDX turned into little functions like:

```
function getDataFromPath(path) {
  const fullPath = join(getPostsDirectory(), path,
"index.mdx")
```

```
  const fileContents = fs.readFileSync(fullPath, "utf8")
  const { data } = matter(fileContents)
  return data
}
```

I used the `fs` node module to list the paths of every post which can be used to determine slugs and other metadata useful for building out the directory pages.

```
function getRawPostPaths() {
  return fs.readdirSync(getPostsDirectory()).filter(f =>
!f.startsWith("."))
}
```

As for the `getStaticPaths` function for my blog posts:

```
export const getStaticPaths = async () => {
  const pages = getPageSlugs().map(slug => {
    return {
      params: { slug },
    }
  })

  return {
    paths: posts,
    fallback: false,
  }
}
```

Then the `getStaticProps` function exported from the blog post page would use that slug to get the actual post contents, transform it from MDX and run any remak/rehype plugins. Although it's more manual, I actually think this is a much more sustainable approach because I'm no longer so dependent on plugins. The plugins for remark and rehype can be used directly, I'm not waiting for a plugin author to add support.

Everything else outside of data fetching pretty much worked. Some pages had to be moved around to make use of filesystem-based routing which was straight-

forward. I now handle syntax highlighting with `rehype-highlight` which replaces the `gatsby-transformer-remark`, the plugin that caused me the most problems. I can now easily add additional languages which helped when I needed to add support for Elixir and Reason. This was as easy as importing the language and adding it to the list of languages.

```
import reason from "highlight.js/lib/languages/reasonml"
import elixir from "highlight.js/lib/languages/elixir"
import bash from "highlight.js/lib/languages/bash"
import { definer as graphql } from "highlightjs-graphql"

const getMdxOptions = () => ({
  rehypePlugins: [
    [
      rehypeHightlightPlugin,
      {
        aliases: { reason: "res" },
        languages: { reason, elixir, graphql, bash },
      },
    ],
})
```

Room for improvement

One of the things I wanted to address was the build times and developer experience. For development, the time to get the server up and interact with a page was insanely fast. Gatsby always had issues starting up even with a cache. The build times, however, aren't quick at approximately 5 minutes, but they're much better than what I saw with Gatsby.

With a bit of optimization, I hope I can get that down to 3 minutes.

Flexibility

The biggest reason for jumping ship from Gatsby is flexibility. I want the option to create different experiences for my blog and although Gatsby has started making this easier, it's not as explicit as Next.js. I use Next.js at work and am familiar with the data fetching capabilities that it offers.

Gatsby was starting to feel a little cramped, I was outgrowing it for what I want to do. Next.js allows me to hack things together more quickly and look to add new experiences with React rather than relying on custom plugins.

Conclusion

Gatsby is still a great tool and I rely on it for my personal website and game development blog. It works without any maintenance as they are simple sites I don't really change regularly. My blog isn't like those other sites, I've been thinking of experiences I want to add to my blog, but I've never done them, or I've scaled them back because it's been too much effort to ensure nothing breaks. Moving to Next.js gives me the flexibility I need and the tools to add those experiences more safely.