Gatsby Brings Speed and Simplicity to Mediacurrent.com

Mark Casias
December 10th, 2019

Sr. Software Engineer for Mediacurrent. Bass player.

Follow Mark Casias on Twitter

Tagged with case-studies, drupal View all Tags

Back in 2016 we were planning the latest Mediacurrent.com site redesign and were intrigued by the decoupled approach which was gaining popularity in the Drupal community. We liked the idea of breaking apart the front-end and back-end so that we could have more flexibility in how we executed redesigns (i.e. redesigns would not be tied to the Drupal upgrade lifecycle).

We started this journey with Jekyll, but quickly switched to Gatsby. We're excited to have found the organizational agility we were looking for and now believe that Gatsby is a superior solution that can work for many organizations looking to leverage a fast, secure React-based front-end for their website.

Finding a Front End Framework

While we were investigating the best decoupled approach for Mediacurrent.com, our team researched various options including Node.js frameworks like Express.js. We ran into a common problem — the level of complexity required to integrate the various framework systems. We asked ourselves, "Was all of this necessary for a pretty typical corporate site?"

Take One: A Static Site Build With Drupal and Jekyll

Along the way, we reviewed static generators that could solve some of the complexity problems and we liked the performance and security benefits Jekyll offered. At the time of our review, we expected to see more Jekyll/Drupal integrations in the wild than we found. Drupal could *simply* publish content as markdown files that Jekyll could then consume. We did several POC's and liked the overall approach. At first, Jekyll looked great, but it wasn't long before Jekyll turned into "Mr. Hyde."

For small blogging sites where content structure is simple and doesn't require a myriad of 3rd party integrations, Jekyll shines. But when used as a static front-end for our Drupal 8 website, it soon became apparent that Jekyll wasn't the best fit. We typically make daily content updates through our Drupal CMS which required Jekyll's markdown files to constantly be updated with new content via source control. This added to the operational load of maintaining the site because there were often Git merge conflicts that had to be resolved.

Why Gatsby

Luckily, our team had been researching and testing Gatsby as a potential solution for our site's front-end woes. We realized Gatsby was an ideal fit for our front-end due to its ability to compile quickly and its component display flexibility. Other features of Gatsby that appealed to us included:

- A robust plugin system that easily integrates with Drupal and other systems
- React and GraphQL-based
- Highly optimized for performance

Additionally, Gatsby's streamlined development workflow allowed us to make front-end design changes more quickly compared to working with a CSS-based templating system or Jekyll's markdown files.

Migrating to Gatsby

To replace Jekyll, we embarked on migrating our site's front-end to Gatsby. We scoped the effort required which included converting Jekyll markdown files to Gatsby React components. The conversion process took less time than we estimated due to Gatsby's clean, well-thought architecture. The back-end CMS configuration was minimal — enable and configure Drupal's JSON:API module. We had 7 content types on our site whose content would publish to Gatsby. Within each content type, we previously implemented Drupal's Paragraphs module to allow structured content to be published to different devices. We found that in order to properly publish Paragraphs-based content, we needed to integrate it with GraphQL fragments.

During the migration project, we ran into a series of unknowns that we hadn't considered before starting development:

- Build times
  - The amount of content and assets were more than a usual Gatsby build. This caused our builds to time out in Netlify, our host.
  - To resolve this, we started using a Jenkins build process which does the build, then pushes the build to Netlify. Since caching is still available on Jenkins, the build times have improved.
  - More recently, we are working on leveraging Gatsby Preview which allows us to see changes *immediately* on a preview site as well as the Gatsby Builds feature which cuts down our build times dramatically.
- Image processing
  - Processing images was one of the build processes that increased the build time. Consequently, many published images caused the build to take longer than expected. We worked on reducing and unpublishing older content and redirecting it to new, fresh content.
- Body field processing.
  - Images in the body field are not part of the regular processing and Drupal will render relative paths. Since these paths are not part of Gatsby, we needed to add logic that updates an image's path to point to our Drupal instance URL.
  - While we can access the images through the entity id, there was a second problem where the file would not transliterate the names and add the `%20` HTML character code where spaces are expected. At this time we are working on a patch for the gatsby-source-drupal, which changes the name of the file to replace `%20` with a dash to make it more browser friendly. This will be contributed back to the community. In the meantime, we are redirecting any images inside a body field to our publishing system.
- We found short codes for files from our Drupal 7 instance that were processed in the Jekyll build process.
  - We band-aided this fix with our Jekyll module. We had to go back and re-publish this information correctly.
- Audio file information.
  - This was patched through our Jekyll module, but we added a JSON:API field enhancer which added the necessary data to show audio files.
- External JS calls. (Acquia Lift, Pardot)
  - Since Gatsby is ultimately a React application, the way to insert tracking cookies, and other 3rd Party scripts required workarounds. Fortunately, the gatsby-ssr API made this easy. We used the onRenderBody lifecycle to confirm which environment we were on, and inserted the required script tags accordingly.

Final Thoughts

Ultimately, the decision to convert Mediacurrent.com to a Gatsby front-end was a wise move. Considering the issues we inherited by using a static-site generator that wasn't intentionally built as a CMS front-end, Gatsby has been a welcome improvement. The benefits we're experiencing with Gatsby are:

- No wait time to apply Drupal module updates. The Gatsby front-end and Drupal back-end are completely decoupled.
- Simplified decoupled architecture with no middle-men.
- Front-end design updates can be deployed without concern for impact on back-end logic.
- Gatsby Preview allows us to view content updates in real-time prior to publishing to production.
- Build times are now much improved using Gatsby's new Build feature (in beta). We are seeing builds come in at 4-8 minutes on average compared to Jekyll which took up to 15 minutes to compile. We also know that incremental builds are coming soon and will provide substantial improvements.