Migrating Ariadne from Drupal to a Static Site

Jason Cooper describes how the Ariadne journal has recently been moved from a Drupal based site, to a static site managed by Hugo and git.

Introduction

At the start of 2019 Ariadne[1] moved from being a Drupal[2] site to being a static site. This move realised a number of benefits for the journal including an improvement in the site performance and a reduction in the ongoing effort required for site maintenance.

Why move away from Drupal?

When the technical running of Ariadne moved from the University of Bath to Loughborough University in 2015 it was decided to upgrade Drupal to the latest version[3]. This turned out to be far more intensive task than was originally envisaged. This trend continued with each security update for Drupal requiring significant effort to apply to the site.

Within a few weeks of going live, it became obvious that Drupal also required far more resources than expected. A number of unfriendly bots had to be blocked from accessing the site due to their aggressive harvesting practice, which would quickly bring the server load up to unacceptable levels.

Another mitigation implemented to reduce the tendency for the Drupal setup to have high server loads was to place it behind a Varnish cache[4]. As published journal articles don't change after being published the pages can be cached for a long period. This caching did resulted in a significant reduction in the number of requests that needed to be served by the Drupal setup itself.

The downside to the addition of the Varnish cache in front of Drupal was that it was an additional application that needed to be maintained by the technical team - and the learning curve for new members of the technical team to pick up the system was already very steep with just the intricacies of Drupal and its configuration.

As well as the technical effort required, Drupal was also impacting on the editorial process. The process of laying out an article was at times more of a black art than a science.

In summary the decision to move away from Drupal to an alternative process for publishing Ariadne can be summed up with the following sentence: It took up too many resources, both server-wise and person-wise when it should have been a lot easier and quicker for everyone.

Requirements for the new solution

Given the issues with the Drupal setup, the technical team drew up the following list of requirements for a new solution:

1. **Low maintenance effort** - Any updates or security patching process should be kept to a minimum.
2. **Low resource impact on servers** - The solution should be able to serve a large number of pages quickly without significantly impacting on the host server.
3. **Simple editorial process** - The process to add a new article and issue should be simple and easy to document.
4. **Reduced learning curve** - Where possible the solution should use technologies in common use by the technical team.

WordPress

The technical team has quite a bit of experience of configuring and using WordPress and so spent some time experimenting with trying to get a WordPress instance that would meet the identified requirements.

The technical team found that it was certainly lower maintenance effort for them and they had a lot of trust in WordPress's automated updates. The impact on the server was also less than Drupal's. However, the editorial process was still more complex than required, especially adding new author profiles.

After a while it became obvious that despite being closer to meeting the requirements when compared to the existing Drupal setup, it still wasn't as good a solution as the technical team would like.

The ideal solution

While experimenting with using WordPress the technical team commented on a number of occasions that the ideal solution from a server resource perspective would be to simply produce a static site, served by a standard web server. Their main reasoning being that once published the content of an article should never change. In fact the only time that parts of the site should change are:

- when a new article is published
- when a new issue is published
- when a journal policy is updated
- when the site's theme is updated

A suitable solution presented itself when one of the technical team was learning about Hugo[5] and realised that it would be ideally suited for generating a static site for Ariadne.

How Hugo works?

Hugo lets you separate a site's content from its style. When you generate the site, Hugo goes through each piece of content and applies the suitable templates for the content type. Each piece of content may have multiple templates applied to it, for example an issue will have the table of contents template applied to produce the HTML table of contents, but it could also have an RSS feed template applied to it as well to generate the RSS feed for that issue.

Structure of the Hugo project

Ariadne's Hugo project uses the following directories:

- archetypes
- content
- public
- static
- themes

archetypes

Archetypes are empty content types. When you create a new piece of content with Hugo it will use an archetype as the base for the new content file, if one is available. This avoids people having to remember what each piece of front matter is required for each content type.

Ariadne has four archetypes defined:

- **article** - the archetype for all new Ariadne articles
- **issue** - the archetype for all new Ariadne issues
- **author** - the archetype for all new Ariadne author profiles
- **page** - the archetype used for any pages that aren't articles, issues or author profiles

content
Hugo content consists of two sections, the first section is the front matter which is structured metadata about the content. The second section is the content itself.

The front matter can be structured using either YAML[6], TOML[7] or JSON[8] and the content itself can be either Markdown[9] or HTML.

Unless explicitly overridden the directory structure under the content directory maps directly through to the final static site's directory structure.

public
Hugo renders the final static site in the public directory.

static
Any static files required by the site's content are stored in the static directory. The underlying directory structure and content is copied into the public directory when building the final site.

themes
The themes directory stores the themes that can be used when rendering the final site. Ariadne's theme directory contains the one ariadne theme.

Each Hugo theme directory contains all the resources specific to that theme, e.g. layout templates, CSS style sheets, theme images.

Developing the core Ariadne theme
To get a starting point to develop the Ariadne theme from, a couple of issues along with their articles and related author profiles, were manually migrated into site content files and then the static site was generated using Hugo's default templates.

One by one new templates for each content type were added to the theme. New front matter fields were added to the articles and issues as the need arose. Once the core theme templates were complete (articles, issues, author profiles, generic pages and homepage - see figure 1) and the required front matter fields were known, the archetypes for each content type was created.

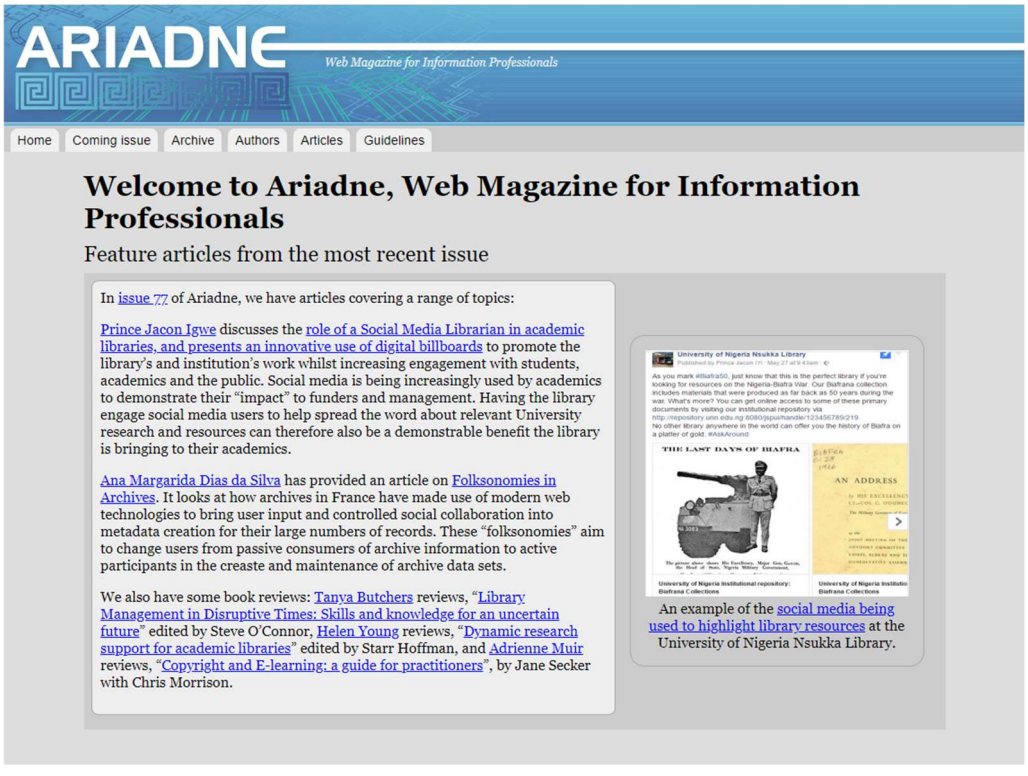

figure 1: Ariadne Homepage

Extracting existing content from Drupal

With the core archetypes, content directory structure and theme started, the next step was to extract the existing content from Drupal and populate content files for each one in the appropriate location in the Hugo project's content directory.

A few of the top level pages were migrated manually (Homepage, Guidelines, Privacy Statement and Access Terms and Copyright), but for Issues, Articles and Author Profiles it was decided that it would be easier to write a Perl script to extract the data directly from Drupal's database and create the related content files for the Hugo project.

Developing the export script took about 3 days of effort overall, which was not only significantly faster than a manual migration would have been, but also resulted in a consistent migration.

Luckily the images for each article were a lot easier to migrate, as they were stored under a single images directory in the Drupal setup - which could simply be copied over to the static project directory. In addition this also removed any requirement to update the image URLs in migrated content as the actual URL of the image would stay the same.

Revisiting the theme

Once all the content files from Drupal were ready the theme could be revisited. This time concentrating on the larger browsing lists (Archives, Authors and Articles) and taxonomies (e.g. Authors, Domains, Buzz - a.k.a article keywords), all of which couldn't be easily developed without having a large number of articles migrated.

Once the theme was complete we could generate a static site version of Ariadne using Hugo and, while we could manually generate the site and upload it when publishing a new article, it was decided to investigate automating the publishing process.

Designing the continuous deployment pipeline

The manual process to publish new content (article, issue, author profile, etc.) was:

1. Create the new content with a draft status
2. Edit the new content
3. Generate the site and push it to the staging environment
4. Proof reading by the authors (if anything needs correcting then return to step 2)
5. Remove the draft status from the new content
6. Generate the site and push it to the staging environment
7. Final proof read by the Editor
8. Push the site to the production environment

As the Hugo project is under git[10] version control and using GitLab[11] as the master repository, a GitLab Continuous Integration (CI) / Continuous Deployment (CD) pipeline was created. The CI / CD pipeline was very simple with just one deploy stage consisting of two tasks. The first task runs on each push into the GitLab repository and generates the static site, including draft content, which it then pushes out to the staging environment via scp.

The second task needs to be manually triggered via the GitLab web interface (see figure 2) and generates the static site, excluding draft content, and then pushes it out to the production environment, also via scp.

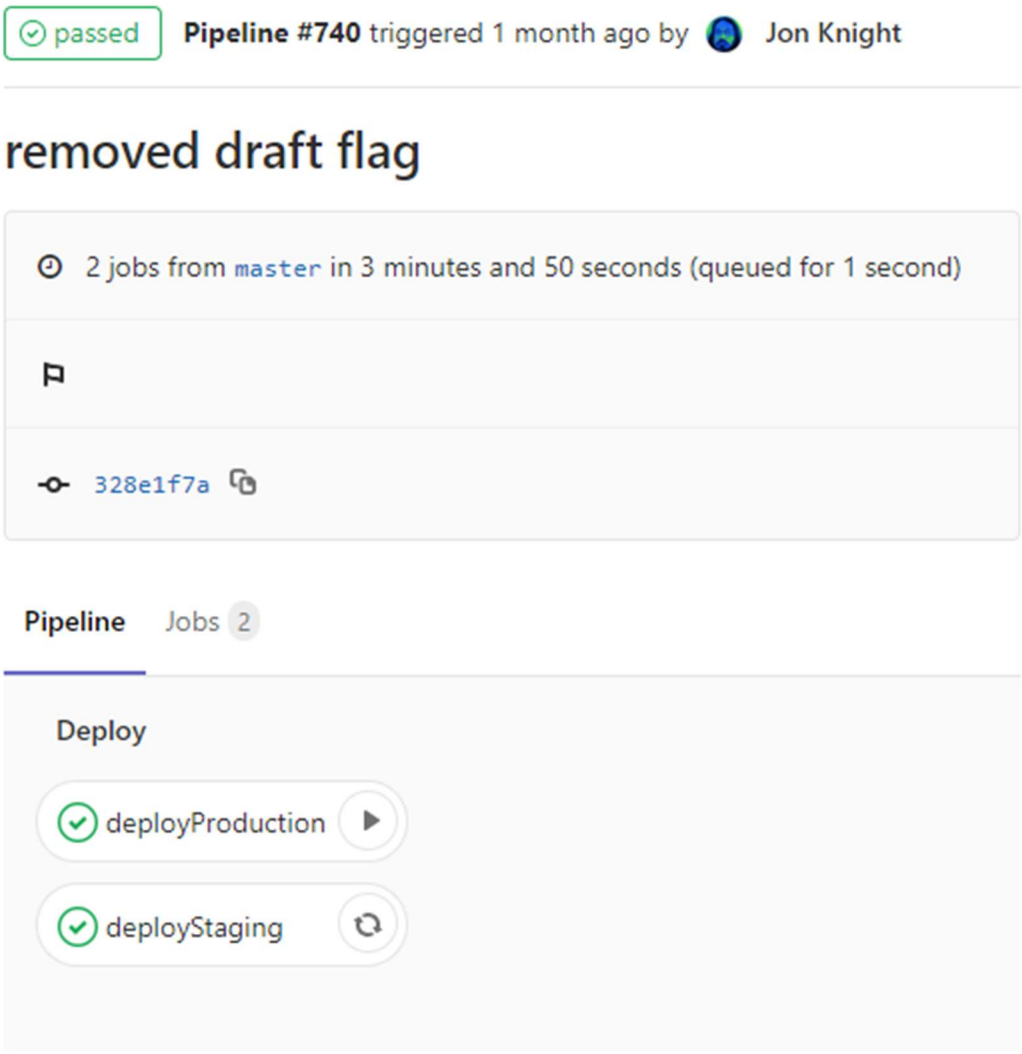

Figure 2: GitLab CI/CD pipeline interface

Citations

In the Drupal version of Ariadne there was a section that listed a citation for each article. As part of the migration it was decided not to recreate this section, as how a citation should be formatted is entirely dependent on where the citation is being published. Instead it was decided to provide a BibTeX[12] and RIS[13] file for each article to enable people to easily add the article to their citation manager software.

To get Hugo to generate the BibTeX and RIS files required the definition of two new media types to be added to the project's configuration (application/x-bibtex and application/x-research-info-systems) then define two new output formats (BIBTEX and RIS) and, finally, set the output types for pages to include these new output formats. At this point Hugo would try to render the BibTeX and RIS versions of the pages, but fail as it didn't yet have a template for them.

Adding the appropriate templates involved creating two new page layouts in the Ariadne theme, one for BibTeX and one for RIS. Hugo would then use these to generate the .bibtex and .ris files to accompany the HTML version of the file.

Once the BibTeX and RIS files were being generated the articles template in the theme was updated to include links to them in the article's aside box.

Going live

The process for going live with the new static site version of Ariadne consisted of removing the Varnish cache and Drupal from the server and pointing the Apache[14] virtual host for www.ariadne.ac.uk at the static site files on the web server.

The most noticeable difference for the new site was the responsiveness. Previously the response time between pages had widely varied depending on whether the page was present in the Varnish cache or not. With the new static site the performance was not only fast, but also consistent between pages.

Conclusion

Moving to a static site generated via Hugo involved quite a bit of effort, but the benefits realised for both the editorial and technical aspects has significantly reduce the ongoing effort required, both by the technical team and the editorial team. Of course the biggest benefit is that readers of Ariadne now have a smoother online experience than previously.

References

1. (2019). Ariadne [online] Available at http://www.ariadne.ac.uk [7th March 2019]
2. (2019). Drupal - Open Source CMS [online] Available at https://www.drupal.org [7th March 2019]
3. KNIGHT, J. 2015, Editorial: Ariadne: the neverending story. http://www.ariadne.ac.uk/issue/74/editorial/
4. (2019). Varnish HTTP Cache [online] Available at https://varnish-cache.org [7th March 2019]
5. (2019). Hugo - The world's fastest framework for building websites [online] Available at https://gohugo.io [7th March 2019]
6. (2019). The Official YAML Web Site [online] Available at https://yaml.org [7th March 2019]
7. (2019). GitHub - toml-lang/toml: Tom's Obvious, Minimal Language [online] Available at https://github.com/toml-lang/toml [7th March 2019]
8. (2019). JSON [online] Available at https://www.json.org [7th March 2019]
9. (2019). Markdown - Wikipedia [online] Available at https://en.wikipedia.org/wiki/Markdown [7th March 2019]
10. (2019). Git [online] Available at https://git-scm.com [7th March 2019]
11. (2019). GitLab [online] Available at https://about.gitlab.com [7th March 2019]
12. (2019). BibTeX [online] Available at http://www.bibtex.org [7th March 2019]
13. (2019). RIS (file format) - Wikipedia Available at https://en.wikipedia.org/wiki/RIS_(file_format)
14. (2019). Welcome! - The Apache HTTP Server Project [online] Available at http://httpd.apache.org [7th March 2019]