SMASHING MAGAZINE

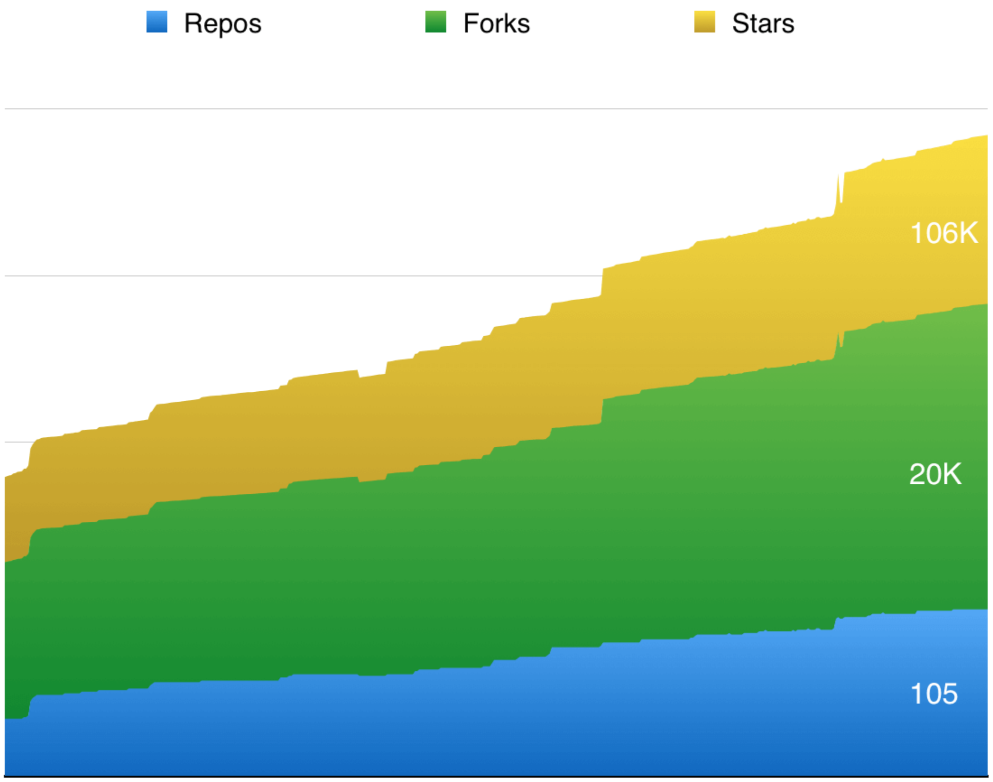
Why Static Website Generators Are The Next Big Thing

By Mathias Biilmann Christensen

Published on November 2nd, 2015 in Tools, Web Development with 37 Comments

At StaticGen[281], our open-source directory of **static website generators**, we've kept track of more than a hundred generators for more than a year now, and we've seen both the volume and popularity of these projects take off incredibly on GitHub during that time, going from just 50 to more than 100 generators and a total of more than 100,000 stars for static website generator repositories.

Influential design-focused companies such as Nest and MailChimp now use static website generators for their primary websites. Vox Media[2] has built a whole publishing system[3] around Middleman. Carrot[4], a large New York agency and part of the Vice empire, builds websites for some of the world's largest brands with its own open-source generator, Roots[5]. And several of Google's properties, such as "A Year In Search[6]" and Web Fundamentals[7], are static.



StaticGen's graph of growth over the last year. (View large version[9])

Static websites are hardly new, going all the way back to the beginning of the web. So, why the sudden explosion in interest? What's up? Why now?

When Static Was It

The first ever website, Tim Berners-Lee's original home page for the World Wide Web[10] , was static. A website back then was a folder of HTML documents that consisted of just 18 tags[11] . Browsers were simple document navigators that would fetch HTML from a server and allow the end user to navigate them by following hyperlinks. The web was fundamentally static.

As browsers evolved, so did HTML, and gradually the limitations of purely static websites started to show.

Initially, websites were just plain unstyled documents, but soon they grew into carefully designed objects, with graphical headers and complex navigation. By that point, managing each page of a website as its own document stopped making sense, and templating languages entered the picture.

It also quickly became evident that reserving HTML for structure and CSS for style was not enough of an abstraction to keep the content of a website (the stories, products, gallery items, etc.) separate from the design.

Around the same time, SQL-based relational databases started going mainstream, and for many online companies, the database became the almost-holy resting place of all of their content, guarded by vigilant, long-bearded database administrators.

Desktop applications such as Dreamweaver and FrontPage offered solutions for building content-driven websites through WYSIWYG editors, where pages could be separated into reusable parts, such as navigation, headers and footers, and where content to some degree could be put in a database. In some ways, fatally flawed as they were, these were the **original static website generators**: building websites from templates, partials, media libraries and sometimes even SQL databases, and publishing them via FTP as static files. As late as 2004, I had the unique experience of working on a major content-driven website, with tens of thousands of pages spread across different editorial groups, all managed via Dreamweaver!

Even if Dreamweaver could, to some degree, integrate with a database, it had no **content model**, offering no sense of content being separate from design, each half being editable independently with the appropriate tools.

The most mainstream answer to these problems was the LAMP stack and CMS' such as WordPress, Drupal and Joomla. All of these played an incredibly important role in moving the web forward, enabling the Web 2.0 phenomenon, in which user-generated content became a driving factor for a lot of websites. Users went from following hyperlinks to ordering products, participating in communities and creating content.

Dynamic Problems

When I built my first dynamic website more than 15 years ago, I was following the original LAMP-stack tutorials from the MySQL documentation. When I realized that all of this stuff was going on every time someone visited a website that was built like this, it blew my mind!

A web server would load my code into a PHP interpreter, on the fly, and then open connections to a database, sending queries back and forth, using the data in templates and stitching together strings of text into an HTML document, tailor-made for the visitor at that moment. Amazing!

It was, admittedly, a bit less amazing when I visited the website a few years later and found the whole web page replaced with a message from a hacker who pointed out the security flaws in the configuration and was at least generous enough just to deface the website, rather than use it as a vehicle to spread malware.

This dynamic website architecture moved the web forward, but it also opened a can of worms. By a conservative estimate, more than 70% of today's WordPress installations are **vulnerable to known exploits** (and WordPress powers more than 23% of the web). Just a few months ago, ~~1.2 million Drupal installations got infected with malware in the span of a few days~~ 12 million Drupal sites needed emergency patching[12] , and any not patched within 7 hours of the exploits' announcement should be considered infected with malware.. Not a week goes by when I don't follow a link from social media to a website that shows a "Database connection error." Scaling a dynamic website can be very expensive, and agencies that launch a campaign website or the like often have to overprovision drastically in order to guard against the website blowing up if it manages to go viral — or else they have to desperately scale it while trying to get it back online (something that never seems to happen during office hours).

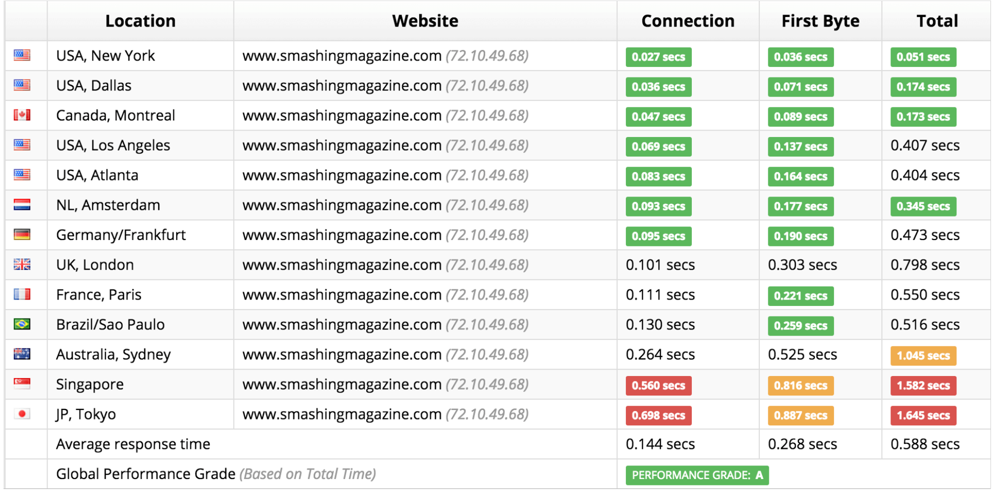
We pay a huge price for the underlying complexity of dynamic code running on a server for every request — a price we could avoid paying entirely when this kind of complexity is not needed.

Dynamic Websites And Caching

To some degree, we tend to work around this by caching. No high-profile WordPress website would be capable of running without a plugin such as WP Super Cache. Large websites no doubt rely on proxy caches such as Varnish, Nginx and Apache Traffic Server in front of their websites.

Caching is notoriously difficult to get right, however, and even the most optimized dynamic website will normally be many times slower than a static solution.

This website, Smashing Magazine, is obviously run by one of the most performance-focused teams out there and is, in general, very heavily optimized for performance. So, I ran a small experiment for this article. Using HTTrack[13], I grabbed a copy of this website one level deep and then deployed the static version to Netlify[14], a static-hosting platform based on a content delivery network (CDN). I didn't do anything to improve performance of the static version apart from simply deploying to a host with deep CDN integration.

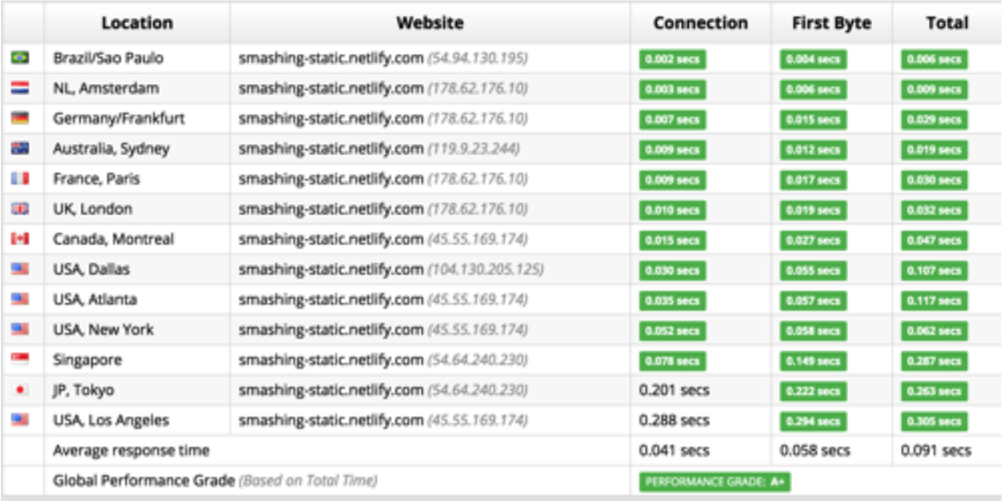


	Location	Website	Connection	First Byte	Total
	USA, New York	www.smashingmagazine.com *(72.10.49.68)*	0.027 secs	0.036 secs	0.051 secs
	USA, Dallas	www.smashingmagazine.com *(72.10.49.68)*	0.036 secs	0.071 secs	0.174 secs
	Canada, Montreal	www.smashingmagazine.com *(72.10.49.68)*	0.047 secs	0.089 secs	0.173 secs
	USA, Los Angeles	www.smashingmagazine.com *(72.10.49.68)*	0.069 secs	0.137 secs	0.407 secs
	USA, Atlanta	www.smashingmagazine.com *(72.10.49.68)*	0.083 secs	0.164 secs	0.404 secs
	NL, Amsterdam	www.smashingmagazine.com *(72.10.49.68)*	0.093 secs	0.177 secs	0.345 secs
	Germany/Frankfurt	www.smashingmagazine.com *(72.10.49.68)*	0.095 secs	0.190 secs	0.473 secs
	UK, London	www.smashingmagazine.com *(72.10.49.68)*	0.101 secs	0.303 secs	0.798 secs
	France, Paris	www.smashingmagazine.com *(72.10.49.68)*	0.111 secs	0.221 secs	0.550 secs
	Brazil/Sao Paulo	www.smashingmagazine.com *(72.10.49.68)*	0.130 secs	0.259 secs	0.516 secs
	Australia, Sydney	www.smashingmagazine.com *(72.10.49.68)*	0.264 secs	0.525 secs	1.045 secs
	Singapore	www.smashingmagazine.com *(72.10.49.68)*	0.560 secs	0.816 secs	1.582 secs
	JP, Tokyo	www.smashingmagazine.com *(72.10.49.68)*	0.698 secs	0.887 secs	1.645 secs
	Average response time		0.144 secs	0.268 secs	0.588 secs
	Global Performance Grade *(Based on Total Time)*		PERFORMANCE GRADE: A		

Smashing Magazine is faster than most websites, but it serves all requests from a single data center. (View large version[16])

I then ran some tests to see how this affected the time to first byte and the complete download time of the main `index.html` page. Here's what Sucuri's super-useful performance tool[17] showed.

Even with a highly optimized dynamic website, the static version is **more than six times as fast on average**! Granted, not every static host will make this kind of difference, but leveraging this level of CDN-based caching simply wouldn't be possible without any manual configuration of a dynamic website, at least not without introducing really weird caching artifacts.



	Location	Website	Connection	First Byte	Total
	Brazil/Sao Paulo	smashing-static.netlify.com *(54.94.130.195)*	0.002 secs	0.004 secs	0.006 secs
	NL, Amsterdam	smashing-static.netlify.com *(178.62.176.10)*	0.003 secs	0.006 secs	0.009 secs
	Germany/Frankfurt	smashing-static.netlify.com *(178.62.176.10)*	0.007 secs	0.015 secs	0.029 secs
	Australia, Sydney	smashing-static.netlify.com *(119.9.23.244)*	0.009 secs	0.012 secs	0.019 secs
	France, Paris	smashing-static.netlify.com *(178.62.176.10)*	0.009 secs	0.017 secs	0.030 secs
	UK, London	smashing-static.netlify.com *(178.62.176.10)*	0.010 secs	0.019 secs	0.032 secs
	Canada, Montreal	smashing-static.netlify.com *(45.55.169.174)*	0.015 secs	0.027 secs	0.047 secs
	USA, Dallas	smashing-static.netlify.com *(104.130.205.125)*	0.030 secs	0.055 secs	0.107 secs
	USA, Atlanta	smashing-static.netlify.com *(45.55.169.174)*	0.035 secs	0.057 secs	0.117 secs
	USA, New York	smashing-static.netlify.com *(45.55.169.174)*	0.052 secs	0.058 secs	0.062 secs
	Singapore	smashing-static.netlify.com *(54.64.240.230)*	0.078 secs	0.149 secs	0.287 secs
	JP, Tokyo	smashing-static.netlify.com *(54.64.240.230)*	0.201 secs	0.222 secs	0.263 secs
	USA, Los Angeles	smashing-static.netlify.com *(45.55.169.174)*	0.288 secs	0.294 secs	0.305 secs
	Average response time		0.041 secs	0.058 secs	0.091 secs
	Global Performance Grade *(Based on Total Time)*		PERFORMANCE GRADE: A+		

The exact same HTML served from a high-performance static host. (View large version[19])

Caching and, more specifically, **cache invalidation** is extremely hard to get right with a dynamic website, especially the kind of distributed caching required to take full advantage of a CDN. With a WordPress website, there's no guarantee that the same URL won't return different HTML depending on whether the user is logged in, query parameters, ongoing A/B tests and so on. Keeping track of when a page needs to be invalidated in the cache is a complex task: Any change to a comment, global website setting, tag, category or other content in the database could lead to changes in the lists of related posts, index pages, archive, comment counters, etc.

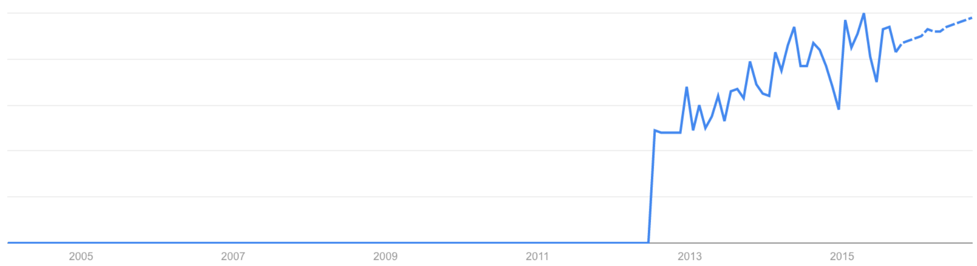
Static websites are fundamentally different in this regard. They stick to a really simple caching contract: Any URL will return the same HTML to any visitor until the specific file that corresponds with that URL is explicitly updated.

Working with this caching contract does impose constraints during development, but if a website can be built under these constraint, then the difference in performance, uptime and cost can be enormous.

The Modern Static Website Generator

In recent years, this alternative to the traditional dynamic infrastructure has gained ground. The idea of a static website generator is nothing new. Even WordPress' largest competitor back in the day, Movable Type, had the option of working as a static website generator.

20



Google Trends for "static website generator". (View large version[21])

Since then, a lot of the constraints that made static websites lose out have fallen away, and today's generators are modern, competitive publishing engines with a strong appeal to front-end developers.

More static website generators are released every week, and keeping up with developments can be hard. In general, though, the most popular static generators share the following traits.

TEMPLATING

Allowing a website to be split into layouts and includes to get rid of repetition is one of the basics of static website generators. There are myriad of template engines to chose from, each with its own tradeoffs — some being logic-less, some inviting a mixture of template and code, all allowing you to get rid of duplicate headers, footers and navigations.

MARKDOWN SUPPORT

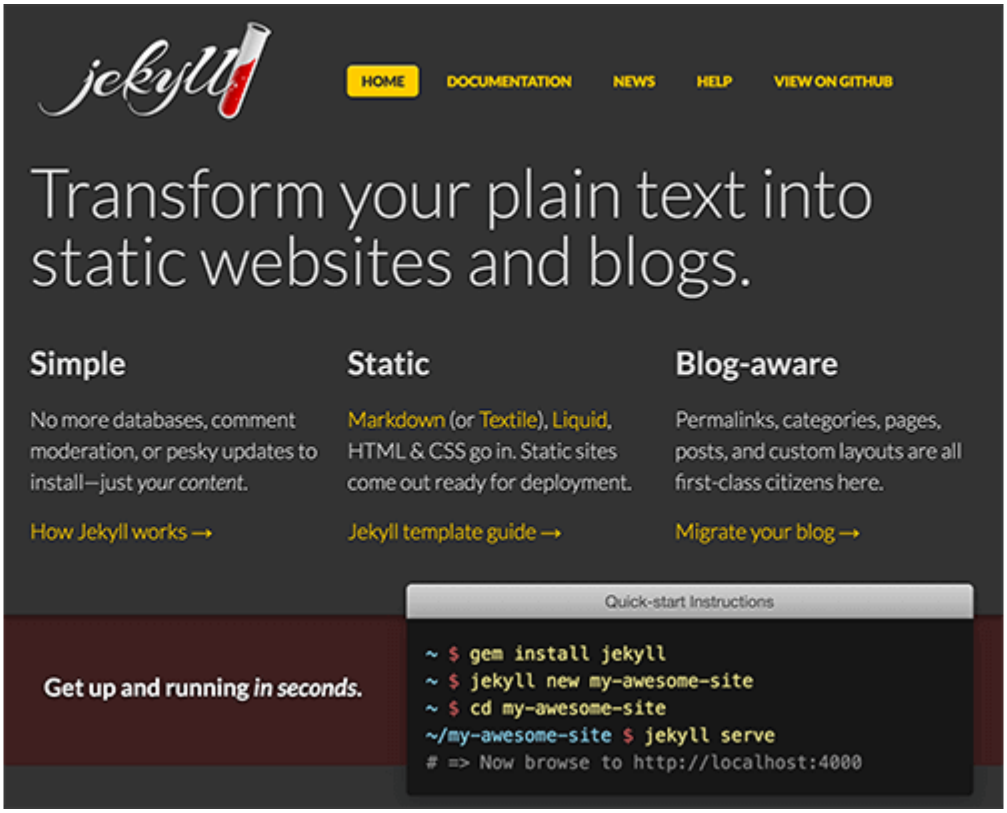
The rise of Markdown is likely one of the primary reasons why static website generators have become so popular. Few people would dream of writing all of their content in BBCode[22] or pure HTML, whereas Markdown is very pleasant to work with, and Markdown editors for serious writing, note-taking and blogging seem to be exploding in popularity.

All of the major static generators support Markdown. Some swear by reStructuredText[23] or an alternative markup format. In general, they all allow content developers to work by writing plain-text documents in a structured format.

This approach keep content and design separate, while keeping all files as plain text. As developers, we've grown accustomed to an amazing suite of tools for working with plain text, so this is a huge step up from having all content dumped into a database as binary blobs.

META DATA

Content rarely stands completely on its own. Readers will often want to know the author of a blog post, the date of the post, the categories it belongs to and so on.



Transform your plain text into static websites and blogs.

Simple

No more databases, comment moderation, or pesky updates to install—just *your content*.

How Jekyll works →

Static

Markdown (or Textile), Liquid, HTML & CSS go in. Static sites come out ready for deployment.

Jekyll template guide →

Blog-aware

Permalinks, categories, pages, posts, and custom layouts are all first-class citizens here.

Migrate your blog →

Get up and running *in seconds.*

```
~ $ gem install jekyll
~ $ jekyll new my-awesome-site
~ $ cd my-awesome-site
~/my-awesome-site $ jekyll serve
# => Now browse to http://localhost:4000
```

Jekyll[2625] *has pushed the idea behind static site generators forward: now it could be powered by Markdown templates.*

When GitHub's own Tom Preston Werner wrote Jekyll[2625] to power his blog, he came up with a really interesting solution for representing meta data when working primarily with Markdown documents and templates: front matter.

Front matter is the bit of meta data, typically in YAML format, at the very top of a document:

```
title: Title of the document
categories:
- Category A
- Category B
---

# Actual content

This is the document
```

This makes annotation of single-file documents with meta data straightforward, and it lends a simple human-readable text format to all of the data that would normally be stored in a much more opaque format in a database.

ASSET PIPELINE

Front-end development today almost always involves several **build tools and compilers**. We want our assets to be minified and bundled. CSS preprocessors have gone from oddities to mainstream tools. And CoffeeScript and ECMAScript 6 transpiling have made compilers an integrated part of programming for the browser.

Most modern static website generators include an asset pipeline that handles asset compilation, transpiling, minification and bundling. Some are based on build tools, such as Grunt, Gulp and Broccoli, and let you hook into whole ecosystems of tasks and build steps. Others are more focused on streamlining a particular process or making sure a certain set of tools work well together without any complex configuration. Live browser refreshing when a file is saved has also become standard for many generators.

PUTTING IT ALL TOGETHER

A static website generator typically comes with a command-line UI for building a website or running a local server with your website for development.

Jekyll, for example, comes with a `jekyll build` command that you run from within a folder containing the source files for a Jekyll project, and it then outputs a completely static website in a `_site` subfolder.

Here's what a simple source folder looks like:

```
  _posts/
     2015-03-01-first-post.md
     2015-03-11-amazing-post.md
  _layouts/
     default.html
  _includes/
     main-nav.html
     index.html
     about.md
  js/
     app.coffee
  css/
     style.scss
     _config.yml
```

This folder is a completely self-contained static website that can be uploaded to any static host or served from any normal web server.

Why Now?

If all of this sounds pretty awesome, that's because it is. But why is static website technology taking off now, and why did the early generators fail to make a dent in WordPress' dominance? What's changed? And how far can we take this?

Today's generators play into a totally different ecosystem than their predecessors. Many of the constraints that made dynamic websites the best option for creating anything but the most basic online brochure have fallen away, although some remain.

THE BROWSER IS GROWING UP

When Tim Berners-Lee launched the first website of the World Wide Web, a browser was a simple document viewer that could display hypertext, links and little else.

Today, we're finally in the process of burying the last browser that has been holding the web back (RIP Internet Explorer 8). The modern browser is an operating system in its own right, no longer merely displaying documents downloaded from the web, but

capable of running full-fledged web applications, making external calls to any CORS-compatible API, storing data locally, opening WebSockets to streaming servers, and even handling peer-to-peer connections to other browsers via WebRTC.

With the maturation of browsers, many features that used to require dynamic code running on a server can be moved entirely to the client. Want comments on your website? Add Disqus, Isso or Facebook comments. Want social integration? Add Twitter or Facebook's JavaScript widget to your website. Want real-time data updating live on your website? Add a squirt of Firebase. Want search? Add Swiftype. Want to add live chat support? Olark is there. Heck, you can even add an entire store to a static website with Snipcart.

The list goes on and on, as a whole ecosystem of purely browser-based add-ons to websites is emerging. Apart from that, modern web apps built with Ember.js, AngularJS or React are often deployed entirely as static websites and served directly from a CDN with a pure API back end that's shared between the website's UI and the mobile client.

THE CDN IS GOING MAINSTREAM

When Akamai launched the first content delivery network in 1999, only the largest web properties in the world could afford to deliver their web assets from CDN edge nodes distributed all over the world. It wasn't that long ago that CDNs were used only by companies at the scale of CNN and Facebook, rather than mere mortals.

While Akamai still has enterprise-level pricing, today anyone can sign up for Amazon AWS and put CloudFront on top of their website. Also, companies like Fastly, MaxCDN and CloudFlare offer CDN services at prices that even a small business can afford.

You could use a CDN with a dynamic website, but cache invalidation is one of those notoriously tricky problems in computer science. Getting the right balance between caching on edge nodes and running a dynamic system on a back end that is potentially doing ad-hoc computations on each request is tricky, to say the least.

A static website, on the other hand, can be readily deployed directly to a CDN and served straight from local caches near end users. Fiddling with the configuration still takes some time, and cache invalidation can be tricky, but it’s doable and can be completely automated with services such as Netlify.

PERFORMANCE IS A MUST

The explosion of mobile devices has changed the face of the web in many ways. More and more visitors are coming to the web from a mobile device, sometimes on a 3G connection. Never has performance been as important as it is now.

We all know this data: 57% of online visitors will abandon a page if it takes longer than 3 seconds to load. People used to be willing to wait up to 10 seconds, but expectations are way higher today. And on mobile, where there’s no multi-tasking and little else to do, waiting for a website to load is so frustrating that more than 4% of people report that they’ve physically thrown their phone while using a slow mobile website!

No matter how much you optimize a dynamic website for performance or how many thousand of dollars you throw at it, it will never give you the same basic performance guarantee as a well-tuned static website hosted right on a CDN for a few bucks a month. With performance constantly growing in importance, it’s no wonder that developers are looking for ways to pre-generate their HTML, instead of letting the server spend time and resources on generating a page for every single HTTP request.

Static website generation also eliminates a lot of performance concerns during the development process.

If you’re building a dynamic database-driven website, the efficiency of the database queries you’re making is extremely important because they’ll need to be fast enough to run once for every single HTTP request. Even if a solid caching layer lies on top of your website, there's often a risk that some requests will effectively work as cache-busters, triggering worst-case queries in the back end and causing the whole system to grind to a halt.

With a static-generated website, it doesn't matter much if pulling content into a template takes a few seconds more or less: That only happens when you publish, and there will never be a performance penalty for end users.

BUILD TOOLS ARE EVERYWHERE

Compilers and build tools used to be something that C and Java programmers worried about, not something you would wield while building a website. Now, for better or worse, that has completely changed.

Today, front-end developers have adopted build tools, package managers and various kinds of compilers and transpilers wholesale. Grunt was the first front-end build tool to go mainstream, and now most new projects will have build steps.

With this prevalence of build tools, static website generators feel like a much more natural part of the front-end toolkit, whereas the traditional PHP-based tools for dynamic websites are starting to feel strangely alien to the modern front-end workflow.

What's Missing?

All of these forces have come together to create something like a perfect storm for static website generators, and it's no wonder that more and more websites are being built statically.

It's not all roses, though. Before static websites go fully mainstream, a few areas need to evolve.

Picking a static website generator and starting a project can still be a **surprisingly rough experience the first time around**. There are a lot of ins and outs and a lot of room for improvement in the tools, documentation and resources available.

While the ecosystem around static website generators is growing, it's still a **far cry from the mature theme marketplaces** and support services of traditional dynamic platforms.

The biggest missing piece of the puzzle, however, is **content editing**. While working directly in Markdown in a text editor and pushing to GitHub is close to the ideal workflow for a front-end developer, it's not something you'd get normal, non-technical end user to participate in.

Because of this, many websites built with static website generators currently end up being migrated to a dynamic CMS. There's a huge need to bridge the gap between content editors and static website generation. Before that happens, static website generation will be reserved for a relatively small subset of today's websites.

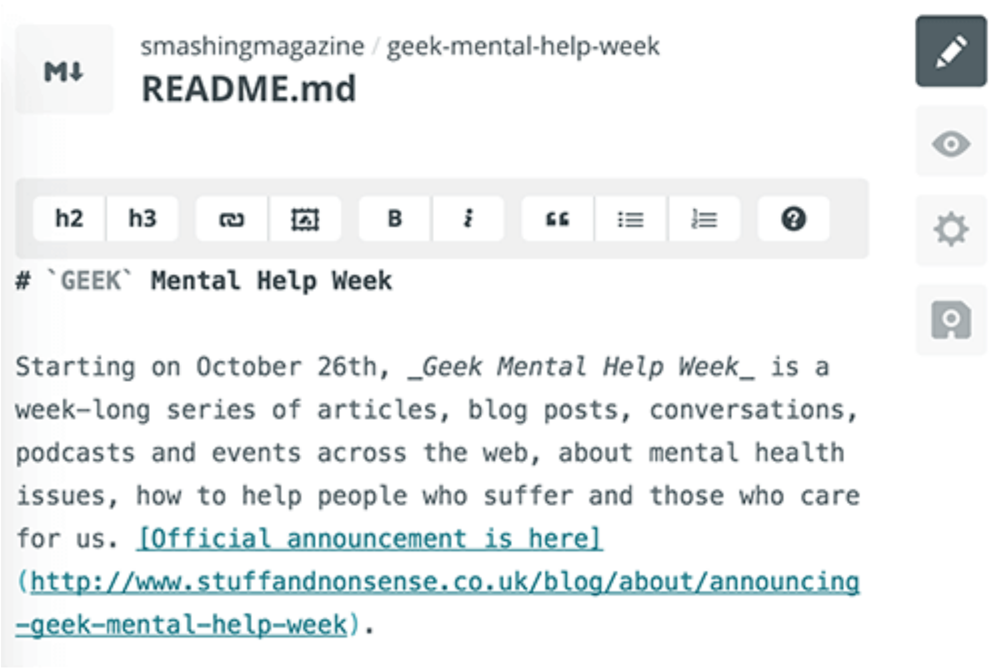
Some interesting "no-CMS" solutions are out there. The Verge has been using Google Sheets as a content layer for Middleman[27]; StaticGen[281] uses Gist and the GitHub API as a kind of database; and Carrot use Contentful as a static CMS[29] to let non-techies produce content for its statically generated websites.

Others are working on tackling the general problem of how to best mix static website generation and content editing, and the coming years will no doubt bring exciting new ways of working with content and publishing.

Systems such as Contentful[30], Prismic.io[31] and GatherContent[32] decouple the CMS layer from the actual website builder. This makes them really interesting tools for multi-channel content management, where you're writing content not just for a particular website, but also for a mobile app, a Facebook page or a white paper. Publishing new content triggers a webhook in a build system; then, a static website generator runs the build and fetches data from the content API; and the result is pushed straight to a CDN.

Another option for content editing is to **work directly on the underlying repository**.

[33]



smashingmagazine / geek-mental-help-week

README.md

```
# `GEEK` Mental Help Week

Starting on October 26th, _Geek Mental Help Week_ is a
week-long series of articles, blog posts, conversations,
podcasts and events across the web, about mental health
issues, how to help people who suffer and those who care
for us. [Official announcement is here]
(http://www.stuffandnonsense.co.uk/blog/about/announcing
-geek-mental-help-week).
```

Markdown editor in a static generator Prose.io[3534] which seamlessly integrates with GitHub's API.

Prose.io[3534] has been around for a while now, integrating with GitHub's API to give content editors a somewhat more gentle UI in which to edit Markdown files in GitHub.

At Netlify, we're working on an open-source CMS[36], without any lock-in to a particular static website generator, Git host or hosting platform. The goal is to make it work with almost all current static website generators, and we think it will be a great way to push the limit of what kind of website you can build within the constraints of modern static website technology.

Obviously, there will always be websites that are simply not a good fit for static generation — especially ones whose core content is a constantly updating feed or ones with an extremely high volume of content that relies heavily on search and filtering.

That being said, static website generators will continue to grow in capability and popularity. The infrastructure and ecosystem will keep maturing. And as the tools improve, we'll see developers push the limit of what can be done with static websites.

At Netlify, we're already starting to see large content-driven websites, with real-time search, multi-language regions and content, and private sections being built with static website generators and content APIs. With awareness of the importance of performance and security increasing, you can expect to see much more of this.

(al, ml, jb)

FOOTNOTES

[1] https://www.staticgen.com

[2] http://www.voxmedia.com/

[3] http://product.voxmedia.com/2015/7/8/8907841/introducing-autotune

[4] http://carrot.is

[5] http://roots.cx/

[6] https://www.google.com/trends/2014/

[7] https://developers.google.com/web/fundamentals/

[8] http://www.smashingmagazine.com/wp-content/uploads/2015/10/01-staticgen-stats-opt-preview.png

[9] http://www.smashingmagazine.com/wp-content/uploads/2015/10/01-staticgen-stats-opt.png

10 http://info.cern.ch/hypertext/WWW/TheProject.html

11 http://www.w3.org/History/19921103-hypertext/hypertext/WWW/MarkUp/Tags.html

12 https://nakedsecurity.sophos.com/2014/10/30/millions-of-drupal-websites-at-risk-from-failure-to-patch/

13 https://www.httrack.com/

14 https://www.netlify.com/

15 https://performance.sucuri.net/domain/www.smashingmagazine.com

16 http://www.smashingmagazine.com/wp-content/uploads/2015/10/02-smashing-dynamic-load-times-opt.png

17 https://performance.sucuri.net/

18 https://performance.sucuri.net/domain/smashing-static.netlify.com

19 http://www.smashingmagazine.com/wp-content/uploads/2015/10/03-smashing-static-load-times-opt.png

20 http://www.smashingmagazine.com/wp-content/uploads/2015/10/04-static-site-generator-trends-opt-preview.png

21 http://www.smashingmagazine.com/wp-content/uploads/2015/10/04-static-site-generator-trends-opt.png

22 http://en.wikipedia.org/wiki/BBCode

23 http://docutils.sourceforge.net/rst.html

24 http://jekyllrb.com/

25 http://jekyllrb.com/

26 http://jekyllrb.com/

27 http://product.voxmedia.com/2014/7/29/5863004/take-a-peek-at-the-code-that-powered-the-verge-50

28 https://www.staticgen.com

29 http://carrot.is/coding/static_cms

30 https://www.contentful.com/

31 https://prismic.io/

32 https://gathercontent.com/

33 http://prose.io/

34 http://prose.io/

35 http://prose.io/

36 https://github.com/netlify/netlify-cms

Mathias Biilmann Christensen

Matt Biilmann has been building developer tools, content management systems and web infrastructure for more than a decade. He is co-founder and CEO of Netlify, the premium static hosting platform. In his spare time he drinks Jazz and listens to Beer, while helping to organize the SF Static Web-Tech Meetup.

With a commitment to quality content for the design community. Founded by Vitaly Friedman and Sven Lennartz. 2006-2015. Made in Germany. http://www.smashingmagazine.com