

CHAPTER 1

Hugo

IN ACTION

Static sites and dynamic JAMstack apps

Atishay Jain



MANNING



Save 50% on this book – eBook, pBook, and MEAP. Enter **mehia50** in the Promotional Code box when you checkout. Only at manning.com.

Hugo IN ACTION

Static sites and dynamic JAMstack apps

Atishay Jain

 MANNING



Hugo in Action

Static sites and dynamic JAMstack apps

by Atishay Jain

ISBN 9781617297007

350 pages

\$39.99



Hugo in Action
Static sites and dynamic JAMstack apps
Atishay Jain

Chapter 1

Copyright 2019 Manning Publications
To pre-order or learn more about these books go to www.manning.com

For online information and ordering of these and other Manning books, please visit www.manning.com. The publisher offers discounts on these books when ordered in quantity.

For more information, please contact


Special Sales Department
Manning Publications Co.
20 Baldwin Road
PO Box 761
Shelter Island, NY 11964
Email: Erin.Twohey@manning.com

©2019 by Manning Publications Co. All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by means electronic, mechanical, photocopying, or otherwise, without prior written permission of the publisher.

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in the book, and Manning Publications was aware of a trademark claim, the designations have been printed in initial caps or all caps.

- ⊗ Recognizing the importance of preserving what has been written, it is Manning's policy to have the books we publish printed on acid-free paper, and we exert our best efforts to that end. Recognizing also our responsibility to conserve the resources of our planet, Manning books are printed on paper that is at least 15 percent recycled and processed without the use of elemental chlorine.

 Manning Publications Co.
20 Baldwin Road Technical
PO Box 761
Shelter Island, NY 11964

Cover designer: Marija Tudor

ISBN: 9781617297007

contents

- 1.1 Parts of the JAMstack 2
- 1.2 How does the JAMstack work? 5
- 1.3 How is JAM different from LAMP, MEAN, or MERN? 7
- 1.4 Why use the JAMstack? 9
- 1.5 Selecting the builder 11
- 1.6 Why choose Hugo? 13
- 1.7 Is speed really important? 17
- 1.8 What can we build with Hugo? 18

The JAMstack with Hugo

This chapter covers

- Basics of the JAMstack for building websites
- Principles of static site generators
- Understanding the Hugo static site generator
- Benefits the Hugo static site generator
- Use cases best suited for JAMstack and Hugo

If you've been associated with websites recently or have friends who work for companies with web-based products, you must know how much work it is to maintain a website. With the need for DevOps engineers, system administrators, and database architects, keeping a website running on the internet is a full-time job for an entire team, not only an individual. The upkeep of content is so difficult that creators, such as WordPress.com, are move at an unprecedented rate to manage hosting or even give away their content to platforms such as Medium or FaceBook.

The JAMstack provides a way of doing web development that minimizes the day-to-day overhead of maintaining websites. The term was coined by the co-founder and CEO of Netlify, Matt Biilmann, in 2016. It forgoes the databases by storing all the content into files that are compiled during deployment and then distributed over a Content Delivery Network (CDN). The dynamic, server-based content is provided by Application

Programming Interfaces (APIs) maintained by third parties or hosted by a cloud service provider with minimal day-to-day involvement by the website owner. This way, web developers are free from the tasks of handling security updates, Denial of Service (DoS) attacks, and the constant monitoring required to keep hackers at bay.

The JAMstack is heavily reliant on the core web technologies of HTML, CSS, and JavaScript. It offers the ability to get up and running in the modern web quickly. We can build websites offering great performance with low costs and requiring little maintenance. It can be used to build websites for a great variety of use cases, from individual blogs to business websites. The JAMstack can work in harmony with a server-based framework by providing full support for static content, while the user-generated, server-based content can still be provided by a traditional framework.

Hugo is among the most popular of the frameworks in the JAMstack and provides the best build speeds. It meets the promise of the tool with which we can enjoy web development without having the annoyances related to the setup, upkeep, or day-to-day maintenance. *Hugo is a rare tool with which the developer can choose when to have coffee, because it eliminates the waiting for compilation, updates, and deployment.* Hugo takes a template and all the content of the website in a markup format and converts it to the HTML that can be hosted as is.

I congratulate you for picking up this book and embarking on the journey to radically simplify your approach to web development.

1.1 Parts of the JAMstack

The JAMstack is different from traditional web development stacks in which a collection of tools and technologies are prescribed to develop websites. In the JAMstack, the static content is created in a markup language, stored alongside the template, and then compiled at deployment time so that it can be used as is when requested by a client.

The JAMstack doesn't prescribe any specific technology to be used to develop websites. It provides an approach to web development where the core of the website is prebuilt and the dynamic nature is added by client-side scripting.

The websites built on the JAMstack consist of three distinct parts: JavaScript, APIs, and Markup.

1.1.1 JavaScript

JavaScript provides the dynamic functionality to the JAMstack. It enables developers to react to user actions and modify the user interface at runtime. The word JavaScript doesn't necessarily mean the JavaScript language. It includes all approaches to client-side scripting, including languages that compile to JavaScript and runtimes built over WebAssembly. The JAMstack leaves the specifics the JavaScript framework and its management to the web developer.

In traditional Content Management Systems (CMS), the server plays a major role in handling user interactions. Fresh pages need to be generated even when a part of

the page needs to be modified. That isn't only unnecessary, it's suboptimal. Modern JavaScript is fully capable of storing the user state in the browser and updating the content based on it. It can communicate with the server and update the interface without the user needing to perform a reload or see a flicker in the interface.

Traditional CMS have failed to adopt to these new trends, features, and capabilities of the web platform. While they do have JavaScript support, it has been patched over a system where the server still participates heavily in these dynamic interactions. The JAMstack prescribes using JavaScript for the use cases where it shines the best: providing interactive interfaces to the end user and communicating from the client to the server. Chapter 8 is dedicated to using JavaScript to enhance the website.

1.1.2 Application Programming Interfaces (APIs)

APIs provide a well-defined contract for communicating with a web service. APIs abstract the entire server functionality, and the client doesn't need to understand the server internals to consume the service. In the JAMstack, JavaScript is responsible for much of work that's traditionally done on the server, but certain tasks still need to be performed at a central location. This includes storage of the application state so that it can be restored on a different machine, computations that require more processing power than a single machine, and data that needs to be transmitted back from the viewer of the website to the servers. These tasks are performed using APIs. The APIs are created using backend technology that's used to create the entire CMS in the traditional stack. The JAMstack doesn't restrict the API style to be used for the communication, and we're free to choose our approach. Representational State Transfer (REST) is the most popular API style in the web platform. We'll use REST APIs in chapter 7 and building one in chapter 10.

Many CMS expose APIs to communicate with the underlying functionality that can be consumed using JavaScript. While technically these approaches fit in the JAMstack definition, the JAMstack advises minimizing the building of APIs to reduce the maintenance overhead. Many third-party API providers provide high-level APIs that developers can leverage without having to go through the overhead of building everything themselves. From handling forms to full text search, much functionality can be obtained at scale without writing custom code.

Even when we need to write custom backends, the cloud service providers make that task easier than building from scratch. With Function as a Service (FAAS), the service providers make it extremely easy to set up an API. The cloud service providers take the ownership of uptime, ongoing security updates, and scaling with user load along with maintaining performance and availability across the globe. The developer needs to understand the cloud platform, build an optimal function based on the constraints of the platform, and hand it over to the service provider. The ongoing work is minimal and needed mostly to enhance functionality or update any dependencies that the developer has chosen for building the function.

1.1.3 Markup

Markup forms the data layer of the JAMstack. Unlike traditional databases, Markup is stored in text files and meant to be readable and editable by humans in their raw form without going through a complicated Integrated Development Environment (IDE). Markup languages provide a means to write formatted documents along with the associated metadata and links to assets in a terse and readable way. Markdown is the most popular of the markup languages used for writing content in the JAMstack, and we'll go into detail on this in chapter 3. It can be accompanied with a variety of metadata languages, one of which is Yaml Ain't Markup Language (YAML), which we'll also discuss in chapter 3.

HTML (HyperText Markup Language) is also a markup language, and you're free to choose that for writing your data in the JAMstack. Human-readable languages such as Markdown make it easier to read and maintain. They're converted to HTML during rendering and enforce keeping the presentation (CSS) out of content.

A website built with a JavaScript based framework and a backend mostly hosted as cloud functions with no markup for content meets the definition of the JAMstack. Writing data as Markup provides many advantages over the approach of storing it in a database and fetching it using an API, especially if the data is unstructured. We can use a version control system like Git to version the data changes. Having the data along with the code eases migration across services and build environments. We can store all configuration together. Optimization and testing gets easier with the ability to boot build environments on demand. With unstructured content, most of the organization and querying capabilities of the databases aren't useful. Hosting blogs or generic web pages based on a database isn't the best use of resources. It's possible mostly because of the effort that has been put into optimizing databases and programming languages used to build these CMS for other use cases.

With the popularity of Git and GitHub, many developers are already familiar with markup languages, especially Markdown. Most readme files in code repositories are written in a markup language. These languages are stable, standardized, easy to learn, and easy to understand. You can find quite a bit of tooling available to write in these languages or migrate data across them. They also work well with diff and merge tools (which are used for version comparison across changes in Git), and most major programming languages have libraries to parse them, which provides extreme flexibility to programmers to manipulate data the way they like.

Using the JAMstack, we write our websites as text-based documents, leverage JavaScript and third-party APIs for our dynamic processing needs, and then build one-off first-party services if they're still required on the cloud to have minimal maintenance. Figure 1.1 shows the process of building a website with the JAMstack.

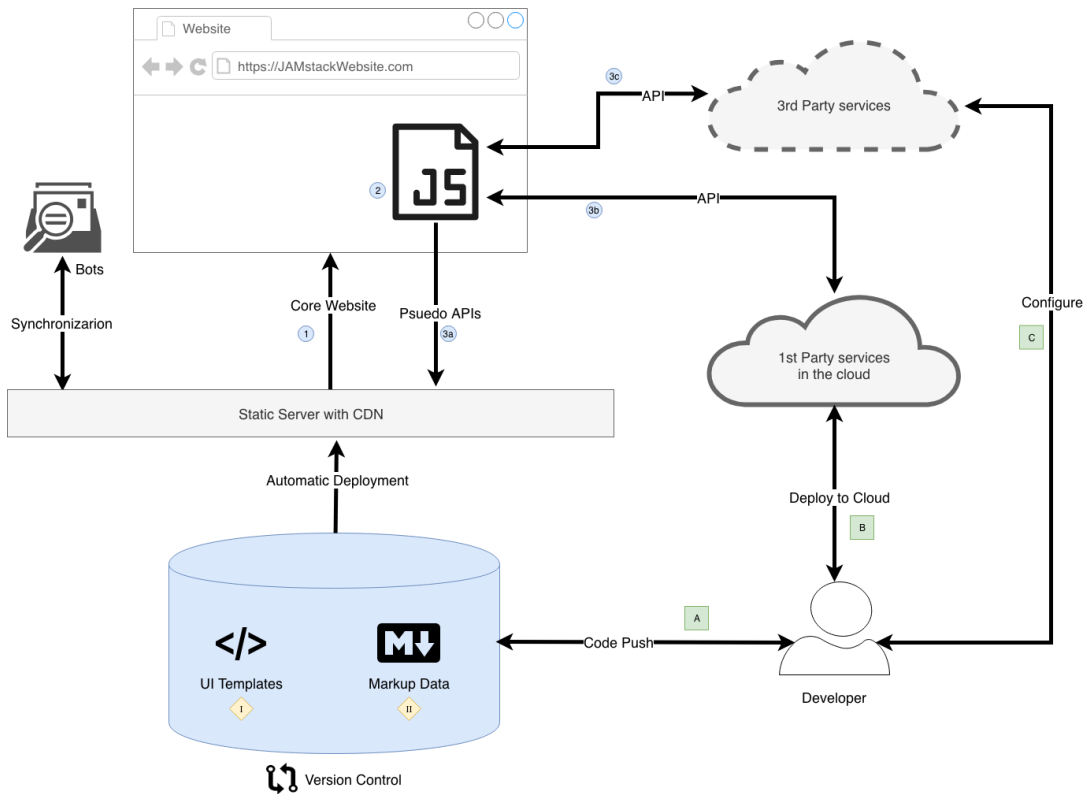


Figure 1.1 Building websites with the JAMstack. This figure describes the various parts of the JAMstack as described in section 1.2. Markers 1-3 describe the flow of information when the user navigates to the website from the main content to the JavaScript execution and accesses the various types of APIs. Markers 1 and 2 are the two main parts of the JAMstack based website, the user interface templates and the content in a markup language. A, B, and C describe the three things that the developer needs to do in order to update the website: update the code for the static portion, deploy the first-party APIs, and configure the third-party ones.

1.2 How does the JAMstack work?

The core of the website using the JAMstack lies in two folders. One folder contains the content in the form of Markup data (Item 1 in figure 1.1) that's shown to the user, while the other one contains the theme as a user interface (UI) templates (Item 2 in figure 1.1). These folders are normally managed in one version-controlled repository, and the build system automatically generates the HTML content and pushes it live. The website is managed by a Content Delivery Network (CDN). The CDN consists of a set of geographically distributed web servers that cache the website content and can provide that to the users with high performance and low cost. The CDN provider takes care of maintaining the uptime, availability across the planet, and scaling the operations with the load.

The content is organized into files, which have both the data to be presented to the user as well as metadata for organizing content such as its tags, menu placements, etc. The content is written in the markup language that provides support for formatting content as well as organizing it into sections.

The theme folder has the UI templates that convert the content into a webpage by adding common page elements as well as styling the content to match the looks of the website. It provides support for creating content pages as well as common pages such as content listings, the home page, contact page, etc. The theme folder can be swapped with a different theme to have a different look for the website, and the framework tries to ensure that minimal content changes are needed when switching across themes.

The websites built with this approach are in the commonly called static websites. Static websites are pre-compiled during deployment. In comparison, dynamic websites require server code to generate the HTML each time it's requested. Tools that help in building static websites are called static site generators.

The developer periodically pushes the changes to the theme and as well as web content that's published automatically (Item A in figure 1.1). The content folder is actively updated to add articles while UI changes come through the theme folder. The developer is also responsible for deploying the first-party APIs (Item B in figure 1.1) and configuring the third-party APIs (Item C in figure 1.1) used in the website.

Websites generally have two types of users: users who have proper browsers that execute JavaScript, or bots that might not support JavaScript execution and parse the HTML to form their view of the website.

When a user requests a webpage, the browser loads the core content of the website from the CDN (Step 1 in figure 1.1). The website also includes JavaScript that gets executed (Step 2 in figure 1.1). This JavaScript in turn may request data from the various APIs available (Step 3 in figure 1.1). There are three types of APIs that can be used to provide dynamic functionality via JavaScript:

- *Pseudo APIs*—These APIs are statically generated along with the website from the markup content. These APIs can provide raw access to data in a programming-friendly format such as JSON (JavaScript Object Notation) instead of HTML that's generated from the template. This way, we can build the entire UI in a client-side JavaScript-based framework (such as Angular, Vue, or React). They can also be used for providing dynamic functionality such as search to a static website on the client side without building a server layer.
- *Third-Party Services*—These services are built and managed by vendors external to the website developer, such as Netlify, PayPal, or Amazon. These vendors provide certain capabilities exposed as APIs on their platform that are maintained by them and can be used directly from the frontend.
- *First-Party Services*—For websites that depend on unique server-side processing for providing functionality, those services have to be built by website developers. These are also exposed through APIs that are accessed from JavaScript in the

JAMstack. For having a low maintenance overhead, it's advised to build these in a managed system, such as FAAS, where the major part of the maintenance work can be managed by a cloud service provider.

Doing it old school

The approach of writing content in a folder on disk and uploading it on a shared hosting provider that manages the content looks much like the early web where we used to upload HTML and PHP files over an FTP connection. The parallels are obvious, which raises the question of what's different this time.

The web has matured since we moved off to controlling full servers. Many features that required server code back then can now be built using frontend technology. The shared hosting has upgraded itself to the cloud, where you can scale not only hosting but arbitrary computation to the internet scale. Even the traditional servers are now hosted in the cloud.

The other major change from that era is the tooling. Tools such as FrontPage were built for designers and end users, which made the website a mesh of copy-pasted scripts that the website author didn't understand. Modern tools target developers with a focus on optimization, maintenance, and performance. Websites can be engineered in these tools rather than being meshed together.

We've learned from the early days of the web, and we have a much better system with enough power and flexibility to build any application desired without compromises.

Contents of the code repository for a JAMstack based website include:

- *UI template*—The templates to generate the HTML and CSS used in the website. Most builders have their own template engine based on the technology they're built with.
- *Content*—The textual data that's rendered with the HTML template.
- *Images, fonts, and other resources*—Non-textual data to help render the website and other data associated with it, including bundled content that's downloaded.
- *Config*—Website configuration present in a text-based format.

1.3 How is JAM different from LAMP, MEAN, or MERN?

Traditional web stacks such as LAMP, MEAN, or MERN consist of three main layers. The bottom-most layer in these stacks is the database layer, where a database such as MySQL or MongoDB is used to manage content. Above this is the application server layer written in a server-side programming language such as PHP or Node.js, where the business logic to communicate with the database resides. This is the web server layer that hosts images, CSS, and JavaScript files. Those JavaScript files may be written in plain JavaScript, in case of LAMP, or rely on a framework such as Angular or React in the cases of MEAN and MERN.

The three layers of the web stack are typically managed by different teams with different expertise. Apart from these teams is also a content team that creates and enters content into the database. The website and the content have a different flow and are managed and released differently. All this adds complexity to the entire process, possibilities of miscommunication, and waste of resources.

The JAMstack tries to merge all these roles and have a single system for everyone involved, thereby making it possible to have a single team manage a website. The rendering logic and the content live in the same code repository, and the database layer is replaced by a standard markup language. A team managing a JAMstack-based website can have sub-teams with different expertise, but with a shared source of truth, build, release and version management flow, the synchronization overhead is greatly reduced. People get an option to wear multiple hats, coordinate closely, and simplify the job of everyone else.

By standardizing on content markup and letting go of the database for content that isn't user generated, the need for the application servers is minimized. For any logic that's not possible by static files, the JAMstack recommends relying on third-party APIs and rolling your own if nothing is available. The frontend can communicate with these APIs using the JavaScript layer, and their release cycle of this home-grown application server can be decoupled.

A new release of a JAMstack-based website is inexpensive to create and distribute, and a team can provide hundreds of releases in a single day without creating any issues. The UI team has access to all the content that they can optimize for. The content team gets the benefit of being a part of the production process and can align the content with the UI without needing any special coordination. The backend work is trimmed to only the operations that need complicated backend logic.

Another difference in the JAMstack is the reduction of the render steps to show a website. All websites frameworks output HTML/CSS and JavaScript because that's the only thing that browsers understand. The difference in approach between all of them is deciding where this HTML is generated:

- *2000s Era (LAMP/Ruby on Rails)*—The bulk of the HTML is generated when the server receives the request. The server processes the request parameters, queries the database, and then generates the output HTML individually for each request.
- *2010s Era (MEAN, MERN)*—The rendered HTML is mostly generated in the browser. The server sends the JavaScript code to the browser, which requests additional data then computes the HTML to be rendered.
- *JAMstack*—The bulk of the HTML is generated when the content change is pushed onto the server and is served as is to the client on each request.

In the approach from the 2000s era, every request involves processing and consumption of resources at the server to generate the HTML. In the 2010s-era stacks, this processing cost is incurred in every client as well as on the server for each page. With the JAMstack, this cost is incurred only once per page per release. Because scrapers such

as the Googlebot index every webpage on every update, the JAMstack is the most resource-optimized way of creating a website.

The JAMstack doesn't prohibit server or client processing. It advises using those only when needed. Deploy-time processing is more efficient and safer. It's advised for all cases where it's practically possible.

1.4 Why use the JAMstack?

Pre-building of HTML content shown to the user has many unique advantages from minimal operations and great performance to cost reductions.

1.4.1 Minimal operations

With the content built before publishing and supplied as plain HTML, the number of moving parts in a website reduces drastically. When the hosting is managed by a third party, the developer doesn't have to worry about security updates, patches, or hot fixes. Server-side code can be patched by the hosting provider. The cloud hosts provide almost 100% uptime without any active involvement from the website owner. The developer has no need to be on call or think about servers, scaling, load balancing, uptime across continents or any other operations overhead. The developer can focus on the joy of building, and the business can focus on their core competency, rather than setting up a DevOps team.

The Acme Story: Act I Scene I, DevOps and the JAMstack

In this book we'll witness the transformation that takes hold within the technical team at Acme Corporation as they embark on the journey to embrace Hugo and the JAMstack. Acme Corporation is the leading supplier of digital shapes on the planet. Like most companies, it has a small IT department that manages, apart from other things, the company website, its online shopping platform, blogs, and parts of the company's web-based marketing data. The small technology team includes a web developer, Alex, and a system administrator, Bob. The technology team is overloaded with work and therefore follows the simple mantra: If it ain't broke, don't fix it. As a result, most of the web stack has remained the same from the early 2000s. The management has agreed that the patchy mobile website needs to go, and they don't want to host servers anymore.

Bob: Hey Alex. I have some great news.

Alex: They let you take the old hard disks home?

Bob: We're moving the existing system to the cloud. It'll be so awesome.

Alex: I haven't even completed my investigation on the JAMstack. How come?

Bob: I convinced the management over coffee. I'm moving to DevOps.

Alex: I really like the JAMstack. It's cheaper, faster, and doesn't even need a lot of DevOps.

Bob: Not doing. Dude I need a job.

1.4.2 Great performance

The pre-built HTML provided as a static website can be hosted completely on a CDN. This way, the content can be easily cached and served from a server located close to the end user. This eliminates the round trip to an application server and the database query that can become a bottleneck. Most site generators targeting the JAMstack generate the HTML at compile time, which means that the HTML is already available to render when the user requests. The JavaScript layer can add functionality if needed, but the website is functional even with a single HTTP request. Generating Accelerated Mobile Pages (AMP) is also easy, and a basic website with the JAMstack can provide a 90%+ performance score on most audits. If a developer is sensitive to performance while building the theme, the JAMstack-based website can meet all criteria for a 100% score in these audits.

1.4.3 Lower costs

With the removal of the database and the application servers from the hosting stack, the hardware costs are greatly reduced. With the operations becoming automatic, most DevOps requirements aren't present. All this translates to major cost savings. You can have a website for free using static site hosts such as GitHub Pages and Netlify. The website can also be hosted on all major cloud providers such as AWS S3, Google Cloud Storage, or Azure Storage at extremely low costs. You don't need to have IT or DevOps teams to manage the fleet of servers.

1.4.4 Developer productivity

The entire JAMstack-based site can be managed by a version control system such as Git. You don't need to set up complicated development environments. Running the code locally is one command away, and most websites can be deployed by a simple push to a server many times a day. No complicated tooling is required to build a website, and all of the content as well as the theme can be managed by a simple text editor. This gives the developer the time and flexibility to focus on the content of the website.

1.4.5 Longevity

HTML/CSS is the most stable technology built, and browsers bend backward to continue to support all features that they've supported since the 1990s. If you host a JAMstack-based website and vanish from the internet for a decade, it will still be there in mostly the same state where you left it when you come back. The internet isn't as forgiving to any other technology stack as it is to plain HTML/CSS/JavaScript hosted on a static server. You can even continue to use the static site generator in a virtual machine without updating the version forever. Because the generator isn't hosted, security vulnerabilities in the generator don't impact the website, and you can build it offline, away from all the hackers.

1.4.6 Tooling

With fewer moving parts and a well-defined structure, the tooling for the JAMstack is much more advanced and powerful than the other stacks. One-click deployment is readily available with hands-off support for scaling through Netlify, GitHub Pages, etc. Having the entire website present as code also means that there's nothing to hide. There are no complicated configurations for security or performance, no extra management overhead for different layers in the stack, and no special IDE to get up and running.

Updating on the fly

When new to the JAMstack, it may seem like a major limitation to be unable to update the content of the website on the fly. Most traditional systems provide an admin mode to update the website on the fly, but nothing seems prescribed in the JAMstack.

This isn't a limitation as there's no need for any special tooling to update a JAMstack-based website. The content is written in a markup language that's so friendly and easy to use that we can provide updates in any text editor. Most version control providers such as Github, GitLab, and Bitbucket provide the ability to commit new changes from the browser that can be automatically built and deployed into production.

With this approach, we get the benefits of having a full version control system for our content alongside the ability to choose the editor to write it in. As a bonus we can update the theme wherever and whenever desired. Textual content, automatic deployment, and continuous integration make sure we don't miss the admin mode in a WordPress instance.

1.5 Selecting the builder

The JAMstack doesn't prescribe a specific technology, and the developer is free to choose the technology of his/her liking to build the website. You can find a huge list of static site builders with various tradeoffs to choose from. Hugo is one of them.

1.5.1 Jekyll

Jekyll is the oldest of modern static site builders and is the most popular. Written by Tom Werner, the co-founder and former CEO of GitHub, Jekyll has been the first-class citizen of GitHub since the beginning. With this tight partnership, it has benefitted immensely from GitHub's success. Built in Ruby, Jekyll has a wide variety of plugins to choose from and a huge community ecosystem. You can find a great deal of help on Stack Overflow and a plugin to almost anything you want to build. While still

extremely popular, Jekyll has several pain points that prompts users to look around. Jekyll can become extremely slow as the website grows. While the Ruby language has improved immensely in performance ever since 2.0 and Jekyll has had multiple versions targeting performance, the core of the platform built while exploring the new domain is difficult to improve. Performance isn't a feature that can be built on the top, and while most developers start with Jekyll for their first static website, many leave the ecosystem due to the day-to-day pains of building with it.

1.5.2 Gatsby

Gatsby is a relative newcomer in the world of static site builders that has gained popularity extremely quickly. Built with React and using GraphQL, Gatsby comes with a strong plugin ecosystem and a lightweight core. It relies on plugins for most of its functionality and provides a versatile core that can be used for any workflow. It builds a progressive web app that's managed in the browser with the data load going through the GraphQL interface. Gatsby is the right choice for developers already well versed with React and GraphQL and are willing to maintain the dependency tree across a fast-moving ecosystem across React, Webpack, and Gatsby itself.







1.5.3 Hexo, Pelican, VuePress, Nuxt and others

The static site ecosystem has a long tail of frameworks built using multiple languages, frameworks and flavors. Hexo is written in pure JavaScript. Pelican is written in Python. Nuxt and VuePress allow you to use the Vue.js JavaScript library. If you strongly like an ecosystem, you can find a static site generator available in your favorite language and framework.

1.5.4 Hugo

Hugo is the fastest of the static site generators, where the development team prides in building a system that can render a complicated website with hundreds of pages in less than a second. Written in Go (Golang), Hugo is distributed as a single binary with all batteries included. With the lack of reliance on plugins, the core team has taken on the responsibility of standardizing most of the features providing the benefit of engineering them for maximum performance. Its template language is a full programming language that can be used to build anything. The documentation is well-maintained, and the community is active in the forums. While not yet 1.0, Hugo has a huge number of features all baked into the single module. It's used by many popular websites with millions of monthly users.

Table 1.1 Web builder technology comparison

Area						
Approach	Word-Press	Custom (Rails)	Custom (MERN)	Jekyll	Gatsby	Hugo
Takes Content in Markdown	✗	✗	✗	✓	✓	✓
Can Version Control Content	✗	✗	✗	✓	✓	✓
Auto scales with load	✗	✗	✗	✓	✓	✓
Can quickly preview content(including launch of preview mode)	✓	—	—	✗	✗	✓
Small set of dependencies	✓	✓	✗	✓	✗	✓
Minimal DevOps	✓	✗	✗	✓	✓	✓
Low update effort	✓	—	—	—	—	✓
Large Plugin Ecosystem	✓	✓	✓	✓	✓	✗

1.6 Why choose Hugo?

Hugo is among the oldest of static site frameworks that has continued to climb in popularity over the years. Its creator, Steve Francia, has immense experience with the Drupal Content Management System and he has brought some of the best practices and rectified Drupal's design flaws with Hugo.

Hugo lies at the sweet spot between a framework such as WordPress, which is built primarily for non-technical audience, and Rails or Express.js, which provide power to generate generic software but require ongoing maintenance effort. With Hugo you get the flexibility that's one level below Rails/Express.js with the maintenance effort that's similar to picking up a third-party hosted WordPress instance with the added advantage of wonderful performance right from the start. Hugo is built for users who don't mind getting their hands into the code but who do need to maintain sanity and have a life outside the project they're building. Developers choosing Hugo rarely move off to other approaches of website building. It comes due to variety of reasons.

You're not alone

Hugo is extremely popular in the industry and has been used at scale for websites such as Bootstrap (<https://getbootstrap.com>), Lets Encrypt (<https://letsencrypt.org>), Smashing Magazine (<https://www.smashingmagazine.com>), Netlify (<https://www.netlify.com/>) and 1Password Support (<https://support.1password.com/>).

Smashing Magazine migrated their huge website with thousands of pages from WordPress to Hugo for its great performance and ease of use.

1.6.1 Hugo is fast

Hugo is the fastest static site builder available. While we may not be able to appreciate this when starting a project, this is extremely important in our day-to-day lives. Waiting for compilation or refresh is a major reason of developer frustration and can mean the death of a hobby project. This becomes even more important when the technology changes force us to go through a major change in our website template. The advent of mobile devices brought death to a huge number of WordPress themes, where updating each and every aspect was so painful that the developer gave up. Even with a decade worth of content, a Hugo-based website will continue to provide a respectable performance for development. A rewrite or a facelift of a Hugo-based website is much easier and more fun than with any of the slower frameworks.

1.6.2 Hugo is built for performance

The community that sprang up looking for and working with the fastest static site builder has a natural tendency to look for performance in everything. Therefore, the core principles of performance orientation flow in the entire ecosystem. You can find advice on how to improve the performance of your website in easy-to-do steps in the community forums. If you find a random script from the internet for doing something with Hugo, a high likelihood exists that it will be optimized for performance.

The core performance of Hugo also impacts its output. The performance primitives are available for other uses. Developers can learn from the approach that Hugo uses for optimizing their own workflows. The quality of an average Hugo-based website is much better than an average website in general.

1.6.3 Hugo is self-contained

A plugin heavy system appears to provide much flexibility and many capabilities until the concept of maintenance rolls in. Your site could go bad if one plugin is abandoned, even though the framework is actively maintained. This has been a classic problem with frameworks like Rails, where each major version became a huge pain for migrating all the plugins. The same can be seen in the popular ecosystems of the past, such as JQuery, Backbone, and AngularJS where plugins are not updated. Even Jekyll, which is extremely popular and actively maintained has a huge problem of *plugin rot*.

Hugo and the Go language

A major skepticism among developers while choosing Hugo is the fear of the Go language. Go isn't a mainstream language and has a small community in comparison to the other options. But this shouldn't impact your decision to use Hugo. *There is no need to learn Go or understand how it works to be successful with Hugo.* This book doesn't have a single line of Go code. We don't need to learn how most of our tools work internally to successfully use them. Go is an internal detail for the users of Hugo that they don't need to worry about while using it.

Go is language built with parallel computation in mind and is optimized for building software. Hugo benefits immensely from Go's speed without forcing upon its users all the complexities. Hugo users only have to use the Go template language, which despite the name, is a different language from Go itself. It's Turing complete (that is, it can be used to write programs that involves formal logic, the basis of all modern computation) and allows us to write anything we want including modules and functions without the complexity of multi-threaded code. *Even the Go template language may not be needed if you aren't planning to write your own theme or shortcodes.* You can write content in a markup language and pick a theme off the shelf to build your website.

Most other JAMstack-based website builders are written in a single-threaded sequential flow. This allows them to have plugins at the cost of performance. With most major features available within Hugo, you don't compromise on a slow framework.

While Go isn't mainstream, it's used to develop important foundational technology such as Docker and Kubernetes. Therefore, it doesn't have a major risk of being abandoned. By Go not being mainstream, the Hugo developers have unique leverage to influence the programming language and get the features that improve Hugo. Hugo's creator, Steve Francia, has led the project management and strategy for the Go Programming language and its best features have rubbed onto Hugo.

Being self-contained has allowed Hugo to bypass issues that have plagued other projects. The core team has been able to standardize optimal approaches to perform tasks that have been made available natively. The Hugo team has optimized Hugo without needing lower-level API compatibility. They continue to write complicated multi-threaded logic for the standardized workflows to eke out the few extra seconds that the users can spend elsewhere. The users get much more support than from the plugin authors and have lesser fears of abandonment of their core workflows.

Being self-contained doesn't mean Hugo isn't extensible. The Go template language is powerful and users can share snippets of code as modules that can be reused and can perform complicated logic using this language.

1.6.4 Hugo is distributed as a single file

Hugo packages all its core dependencies and resources in a single executable file. This makes downloading Hugo, transferring it to another machine, or backing it up extremely simple. In systems where due to security concerns, each file has continued

scrutiny, and a single binary file with no other dependencies really shines. Developers can check-in the Hugo binary with their source code if they need use it in a restricted environment. With a single file taking care of everything, there are no dependencies to update and no build system needed to manage. This is in stark contrast to JavaScript-based static site builders that have hundreds of dependencies, each of which might need to be vetted by a security team for use in an enterprise environment.

1.6.5 Hugo can be extremely low maintenance

With fewer moving parts (plugins and operating system dependencies), a small installation step, no database, and no complicated hosting steps, the maintenance churn with Hugo can be much less than the other approaches of web development. Each dependency is maintenance. You can get a powerful website with low maintenance with only Hugo and a hosting provider. While Hugo has had updates where backward compatibility has broken, you're free to take the updates when you have time and don't have to go down and fix arcane plugins. The churn in Hugo is also reducing as it approaches the stability of 1.0. This cannot be said about most other ecosystems in the web development world.

1.6.6 Hugo can save you from analysis paralysis

Hugo is opinionated and built with many tools and techniques to get up and running quickly. While the powerful template system allows you to roll your own solution to a problem, most of the common ones have already been dealt with. Hugo has a generic implementation for an approach to pagination, to categorizing content into unlimited types of categories as well as to getting core website elements such as menus. Hugo comes with a built-in templates such as YouTube embeds and Instagram images that can save you time getting started. Eventually you might want to customize them to fit your own needs but getting up and running is easy with Hugo because of the well-documented and popular approach of solving most problems.

1.6.7 Hugo is powerful

Despite being opinionated, Hugo is versatile. The Go template language that Hugo extends is a powerful and flexible language. This provides the ability to developers to write proper programs within Hugo. The standard library provided by Hugo is huge and growing. It comes with great performance right from the start. Even if you write bad code, the core performance of the built-in functions will be able to save grace for a long time. With access to APIs during website generation, Hugo provides big power without losing on the performance of the generated output.

With Hugo, you can write functions anywhere in your website including while developing content (as custom shortcodes embedded in the markup) to do special processing. You can encapsulate that into something that can be reused or leave one-time snippets of code on specific pages.

Hugo has web development primitives but not using them doesn't seem like fighting the framework. If you don't use the built-in features of Hugo and decide to roll your own, they may not work across themes, but you can still be successful with Hugo.

1.6.8 Hugo is scalable

Hugo already caters to websites with multi-lingual content having thousands of pages and millions of monthly active users. Hugo has a proven record of handling the scale of the biggest and heavily used websites on the internet. There are already enough primitives and capabilities to scale the Hugo based website from a developer to a team. Hugo supports a wide variety of input and output formats and has a variety of features to enable automation of day-to-day life of a non-technical member of the team.

1.6.9 Hugo is a community project

Hugo is maintained by a community of volunteers with no parallel commercial interest in the project. This allows for the direction of the project to be in the best interest of the community. Hugo cannot pivot, get acquired, or shut down at the whim of a corporation.

1.7 Is speed really important?

The importance of build performance cannot be emphasized enough. Hugo employs many techniques to speed up build times, such as having a multi-threaded core with support for caching at all layers to prevent as much rework as possible. This results in freeing up the developer from the burden of noticing the build time.

When you launch Hugo in the watch mode (a special mode for development) the website comes up in less than a second. It reloads with the speed of typing without having to go through the entire step of setting up fancy hot module replacements for live reload. This feature isn't only for the theme with a dummy content but for the entire website that can be pushed to production. This provides the flexibility to edit the website in the five minutes you might have between other chores. In other frameworks, getting up and ready is itself a task. It takes so much time to get to the development mode, that for minor bugs that are found while browsing, we tend to ignore them simply for the effort.

With the entire data traveling with your website and getting recompiled on the fly, the developer gets the freedom to experiment and see the results instantly, as well as to push to production without dedicating extra mental effort. Same is the case with data entry. A big burden with static site builders with slow build time is that committing data is something that the content writer needs to plan for, since getting up and running itself can take five minutes where you can lose the chain of thought. The content flexibility of WordPress and the runtime performance of the JAMstack are not an either/or with a framework like Hugo.

With this being done at the framework level and all the primitives exposed, as a developer you start to rethink your website building strategy. Does this code need to go into JavaScript that has to run on every one of the billion customer machines that visit this page, or can we write this such that it runs once and saves the results as SVGs or precomputed HTML so that the customers don't have to re-execute? These minor tweaks while building go a long way in improving the website performance.

1.8 What can we build with Hugo?

The JAMstack is a versatile concept and can be applied to a huge variety of problems. Hugo is applicable to most of them and has been a poster child for the success of the JAMstack with its ability to handle scale. It shines when the information needs to flow from the server to the client and the client is mostly used for consumption of the said information rather than creation. This fits the traditional definition of publishing where the content creators provide content via a medium (like the web) to the consumers. The following sections explain things that Hugo specializes in.

1.8.1 Personal websites and blogs

Gone are the days of hiding away from the internet. Through one means or the other, everyone in the modern world who has an internet user as a friend or relative is already present on the internet. Rather than trying to hide from it, the right approach is to embrace the internet and control your online impression rather than letting it depend on others.

Hugo is well-suited for getting up and running with a personal website. Big goals for personal websites are low maintenance, low costs, and the flexibility to showcase your own personal tastes. Throughout this book, we'll see how we can build something extremely low maintenance, almost-free hosting, and enough flexibility to customize as much as you desire. Add to that you get great performance, ability to update when and where you want, full SEO support, and a quick start.

You can pick up any one of the publicly available Hugo themes to get started and be up and running with a decent website in minutes. You'll be surprised how many features are available without any customization. Once you're there, it's easy to fork the theme and start customizing it to leave your unique impression on the internet.

1.8.2 Non-technology business website

Hugo scales to teams updating content in parallel without any problem. Businesses whose core competencies aren't building websites need something easy to maintain, with low costs and great performance. They also want flexibility and control. Hugo ticks all these boxes. It's well thought out and easy to understand for any vendor team who may be added to the website development on a short-term contract. Hugo provides few instances where a developer could write bad code that would slow down the website. The entire mechanism is flexible enough to add the one custom page that

the business needs immediately without having to go through and rip apart the entire website.

With the JavaScript and API layer of the JAMstack, Hugo websites can be extended to provide features reserved for dynamic websites updating on the fly on a server. You'll see in this book how we can build low-cost and low-maintenance features such as shopping carts and pay screens keeping the rest of the website managed statically.

1.8.3 Documentation websites

Hugo has great support for reading structured data from a file on disk like a CSV or a JSON and then creating a website out of it. You can still apply your own custom themes. It has built-in support for syntax highlighting and can scale to a large number of pages extremely easily. This makes it well suited to write custom websites that could read from the API docs and prepare a neatly formatted version of the specification.

1.8.4 Hybrid JAMstack-based websites

All websites have pages that are meant to display content. These include pages such as the privacy policy, a generic About Us page, a blog, a product listing page, and a newsroom where the company releases press statements. For all this content, which is meant to be consumed rather than operated upon, Hugo and the JAMstack can help keeping the content running at a low cost, with high availability and a great performance. The pages which are specifically based on a server technology can be delivered separately or built in JavaScript communicating with the servers using APIs exposed by them.

While Hugo can tackle many of the challenges that are faced while building a website, Hugo isn't perfect for all use cases of web development. In case where a large amount of user generated content is required, such as building a social network, the benefits of Hugo don't come into play. Similarly, for building an app like a photo editor in the browser, Hugo isn't the right choice. In these cases, Hugo can be used for building the static portions of the website such as the corporate pages, the company blog, the About Us page, and the privacy policy, the core functionality of the website is something where Hugo doesn't help much.

Summary

- The JAMstack is an approach to web development where instead of storing the content in a database and querying at runtime, most content is stored along with them theme as files and compiled into the website during deployment.
- The static content in the JAMstack is written in a markup language that compiles to HTML. The dynamic content is available in the form of APIs that can be accessed by Javascript.
- The JAMstack provides massive savings in terms of cost, operations, and maintenance, and we also get a fast website.

- Hugo is a framework to help build these so-called static websites that provide good build performance and are available as a single binary.
- Hugo meets the promise of low ongoing maintenance, a great developer experience, and scalability to a huge team.
- Hugo especially shines at places where the information flow is from the server to the client, such as personal or company websites, news posts, blogs, documentation, etc.
- For places where the information flow is from the client to the server or personalized based on the user, Hugo follows the JAMstack and that information can be added using the Javascript layer that communicates to the servers ideally hosted on the cloud.