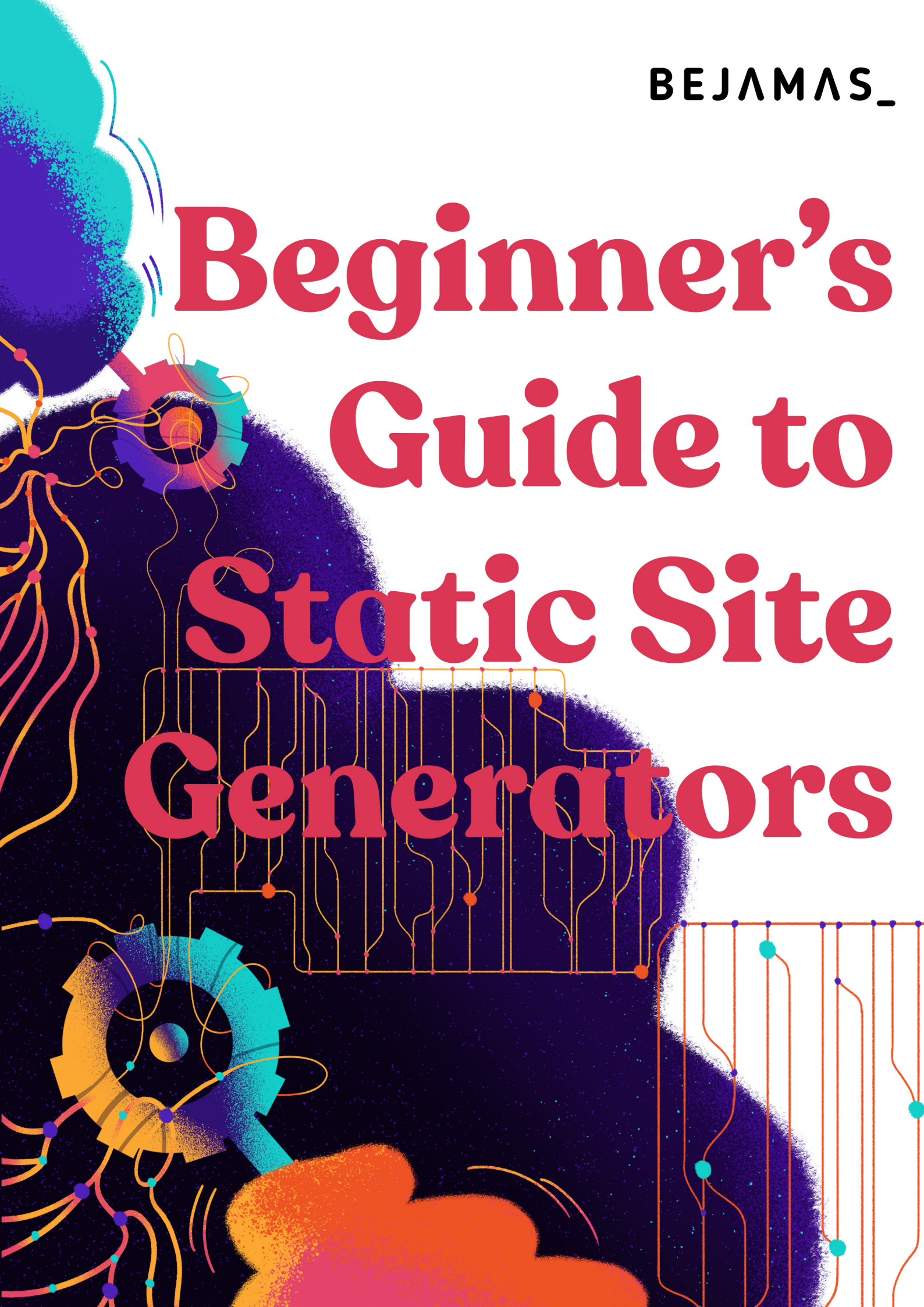
BEJAMAS_

Beginner's Guide to Static Site Generators



Contents

Let’s dive in!

B_

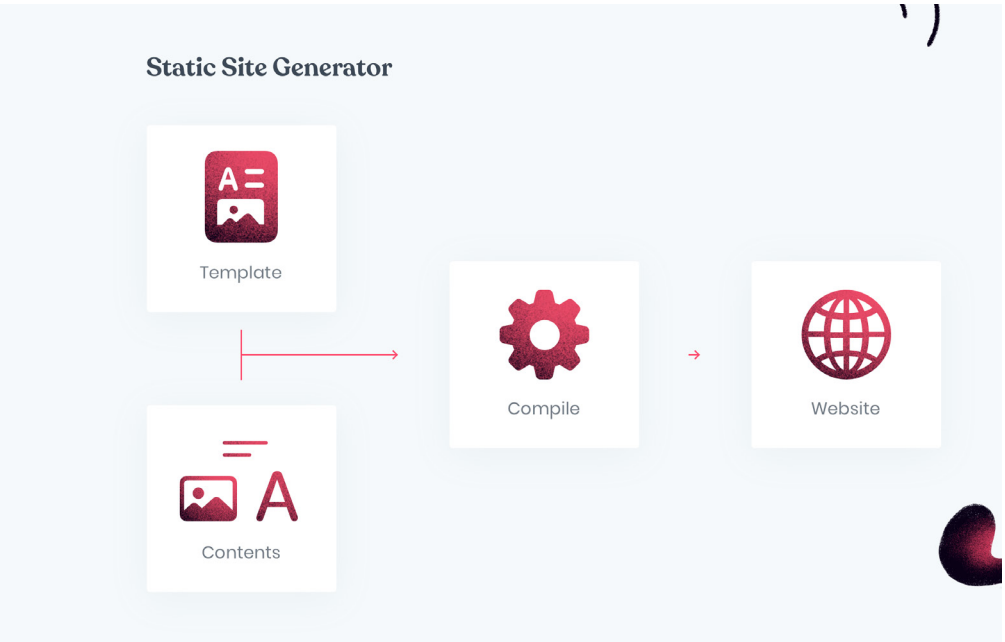
The Beginner's Guide to Static Site Generators

Rapidly evolving Jamstack ecosystem, and static site generators are certainly a big part of it, has sparked a huge interest in the approach and tools around it. We can't say it is mainstream, at least not yet, but having VCs competing with investments in the companies in the space certainly paves the road to much wider adoption.

Setting that aside, the advantages of using static site generators are impressive, with speed, security, and scalability among the top ones. Whatmore web developers are already adding dynamic elements with the use of 3rd party APIs, expanding the capabilities of static sites.
But we're getting ahead of ourselves. Let's deal with the basics first.

What is a static site generator?

A static site generator is a tool that helps you build static pages out of the input files. It takes your content (from a headless CMS, for example), applies a selected template, and generates static HTML pages out of it.

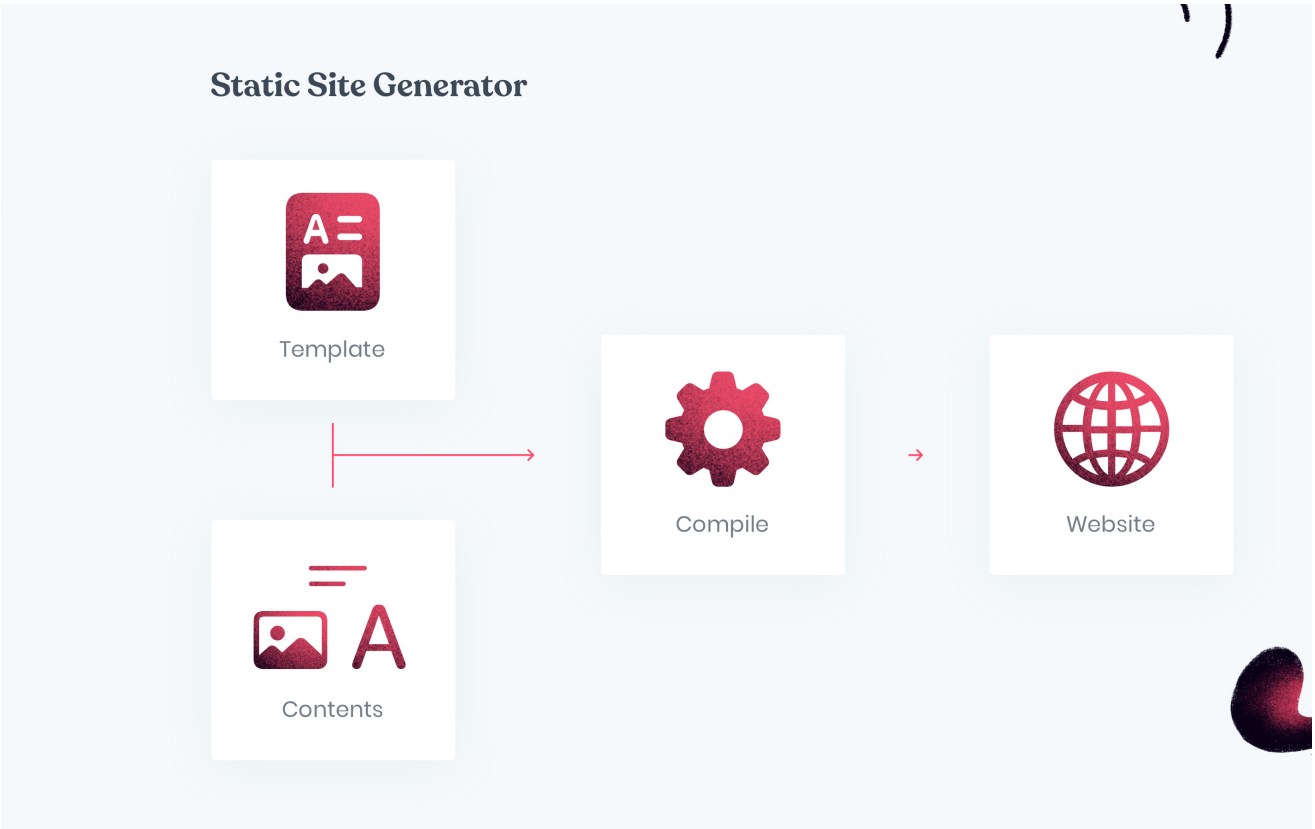


The biggest difference between the SSGs approach and what is now considered the traditional web dev approach embodied in the use of WordPress is that instead of building a page on demand each time a user visits the site, SSG does this at build time. Basically, an already built page, stored on a CDN, is served to the user when he visits the website.

This way, you'll get a website that loads fast but may lack the dynamic functionalities expected in today's websites. I'll get to that in a minute.

Why use a static site generator?

As already mentioned, the benefits of websites built in such a way are many. Being that you are serving static HTML pages, load speed and overall website performance is greatly improved when compared to the traditional dynamic websites (websites built with traditional CMSs like WordPress or Drupal). On top of that, it is incredibly easy to deploy and host static sites.



Since you are serving only static files, you are bound to have fewer security issues simply because the surface area for harmful attacks is reduced. And by having a simple server-side setup, maintenance and scalability are much easier as well.

Then there are the cost-benefits of using static site generators. Besides being globally and consistently fast and always online thanks to CDNs, serving static sites via CDN will lower your hosting bills substantially.

Disadvantages of static site generators

Surprisingly there are some disadvantages to static site generators. With the workflow being more developer-friendly and significantly different from that of the traditional monolith approach, you need more technical skills to work with SSGs. And using SSGs alone would be hard for content managers and editors, for example. Thankfully, with a growing tools ecosystem around static site development, this can be handled easily using the numerous available headless CMS solutions.

While adding dynamic elements and interactivity with users within pages can be a nuisance, there are already solutions in the ecosystem of Jamstack.

Finally, the sheer abundance of choice (I kid you not, right now there are around 600 listed SSGs) makes the decision making (which one to go for) more difficult than anticipated.

How to Choose Your Static Site Generator?

Choosing the right static site generator depends on your project needs, team experience involved in the project, features needed for your choice's project, and programming language.

What are you building?

It is not the same if you have a simple blog for a project or a feature-heavy eCommerce. And it doesn't help to know that SSGs come in all 'shapes and sizes.' Some are made with a specific front-end or back-end in mind. Some are made for creating fast photo gallery websites. Others help you create documentation pages/websites more easily. Most of them (more or less) can help you manage a typical website with a blog.

Who is it for?

Once your project needs are considered, think about who's gonna use/edit the website? Having a streamlined dev experience that usually comes with the use of SSGs is great, but are non-technical users going to work on this site once you are done? If so, think about easing things for them as well. Consider pairing your SSG with a headless CMS.

What language or framework?

The SSGs have vastly improved over the years. You can now find site generators that are based in a vast number of programming languages, use different templating languages and conventions, and run in all sorts of environments.

Do you code in JavaScript? There are Next.js and Gatsby among the most popular SSGs for you. Are you tired of heavy JS frameworks? Take a look at Eleventy. Do you like to write in Go? Then Hugo is the best option. If Ruby is the language of your choice, you should go with Jekyll. Vue is your thing? Check Nuxt.js or Gridsome.

While using preferred language or a framework can ease up the job, keep in mind project needs first.

What are the best Static Site Generators for 2020?

Advising on the best whatever nowadays is a double-edged sword. With that being the case, instead of ranking them as 'the best', I'd like to go through some of the most popular options we've worked with, and that way, hopefully, help you make the best decision for your project.

> *For the sake of clarity and ease of comparison, each static site generator review has the same structured. First, we talk about the history and the current state of the project.*
>
> *Next, we cover the best features and strengths. Then, we focus on the ecosystem and showcase a couple of the most popular projects.*
>
> *Finally, we'll show you how to get started and how to deploy your project with SSG in question. And in the end, as a recap, we'll go through the best features and use cases.*

Let's dive in!

Next.js

B_

Next.js

Next.js started out as a small, zero-config framework for server-rendered universal JavaScript web apps, built on top of React, Webpack, and Babel. Initially, it was focused on enabling a smooth universal JavaScript experience. Over the years, it evolved to a premier static site generator with support for TypeScript, dynamic pages, serverless functions, and more.

Next.js 1.0 - the beginning

It was fall of 2016 when I discovered the first version of Next.js. I created a simple file that looked like this:

```
// pages/index.js
import React from 'react'
export default () => <div>Hello world!</div>
```

I executed now in the terminal and the magic happened. Two lines of code, one command later, and my new hello world website went live. It was hard to compare that experience to developing custom WordPress themes. Next.js and Now (currently Vercel) felt incredibly easy. Almost like cheating.

A few months later, I shipped my first project using Next.js and the WordPress API. The project had some serious performance issues. Fetching data on every request from many API endpoints wasn't the best idea. I probably should export those pages to static ones, right? But it wasn't trivial in the first versions of Next.js.

Fast-forward to Next.js 9.3

Next.js evolved to support static site generation. Vercel evolved to support serverless.

Version 9.3 introduced three new data fetching methods:

- **getStaticProps** to fetch data at build time, this can be content for a single post
- **getStaticPaths** to specify a collection of dynamic routes, for example, a list of blog posts
- **getServerSideProps** to fetch data on every request.

Below is a simple example of static generation:

```
// pages/posts/[slug].js
// [slug] filename means it's a dynamic parameter
// it will be passed to getStaticProps `params` object
const BlogPost = ({ data }) => <Post content={data} />

// executed only at build time
export async function getStaticPaths() {
 const response = await fetch('https://api.domain/posts')
 const data = await response.json()
// all paths we want to pre-render
 const paths = posts.map((post) => ({
  params: { slug: post.slug }
 }))
 return { paths }
}
// executed only at build time
export async function getStaticProps({ params }) {
 const response = await fetch(`https://api.domain/posts/${params.slug}`)
 const data = await response.json()
 return { props: { data, fallback: false } }
}
export default BlogPost
```

The code above fetches from the API all blog posts at build time and creates `paths` array with `slug` for each of them. Data for each single blog post is fetched in `getStaticProps` based on the `slug` param.

If you are familiar with Gatsby, you can notice that `getStaticProps` is a bit like `createPages` in `gatsby-node.js`. But I think Next approach is easier to understand. The difference is that you don't need to specify a path to a template and passing `slug` in `context`. In Next, everything is in the same file. In this case, `slug` value is accessible through a query parameter. To learn more check the Migrating from Gatsby article.

If you add a new post you need to re-build the project unless you change `fallback` to `true`. If there is no existing HTML file on the CDN, Next.js will try to fetch content on the client and cache it. Fallback is very useful if you have a large collection of posts that updates frequently.

Image component

`Next.js 10`, among other things, has introduced a new, built-in image component and optimization. From now on you don't have to worry about shipping large images for mobile devices. Next handles resizing and generates a modern WebP image format that is approximately 30% smaller than JPG. If your website has a lot of images this can greatly reduce bandwidth and client-side performance.

To use <Image /> component import it from next/image:

```
import Image from 'next/image'
const Hero = () => (
  <section>
    <Image src="/assets/cute-hero.png" width={500} height={500} />
    <h1>Hello world!</h1>
  </section>
)
export default Hero
```

Next.js approach doesn't affect build time at all. All optimizations are done at request time. Currently, Next supports 4 cloud providers: **Vercel**, **Imgix**, **Cloudinary,** and **Akamai**.

File Structure

Next.js is **zero-config** from the very beginning.

```
├── package-lock.json
├── package.json
├── pages
│   ├── about.js
│   ├── blog
│   │   ├── [slug].js
│   │   └── index.js
│   └── api
│       ├── posts.js
│       ├── about.js
│       ├── blog.js
│       ├── contact.js
│       └── index.js
└── public
    └── logo.png
```

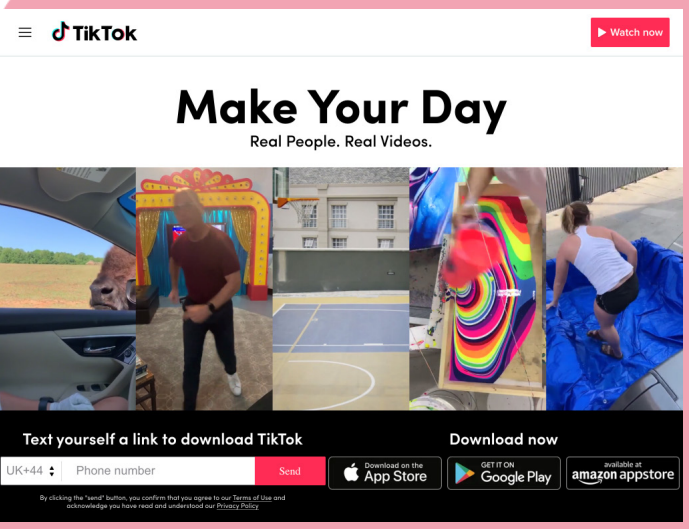
Every file in `src/api` directory is a lambda function.

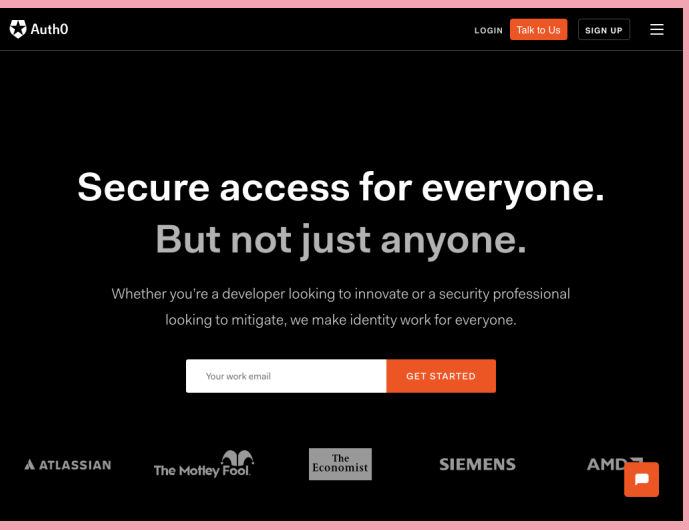
Ecosystem

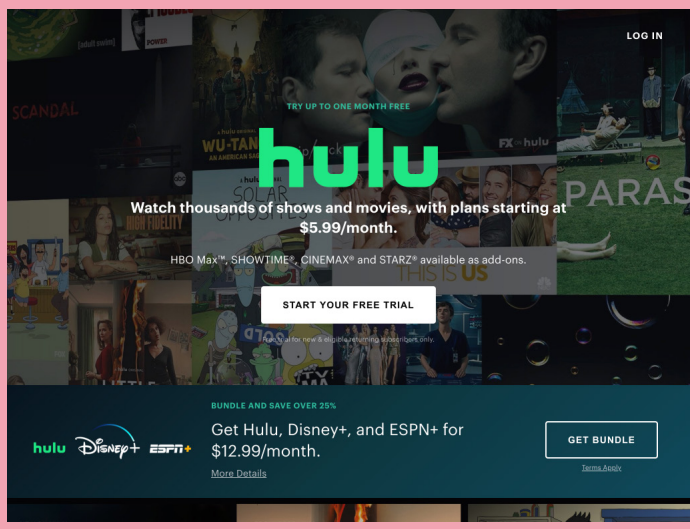
Next.js has a great, growing community. You can read some interesting conversations and discuss RFC on Github. Vercel's team is very clear about the framework direction and open to community suggestions.

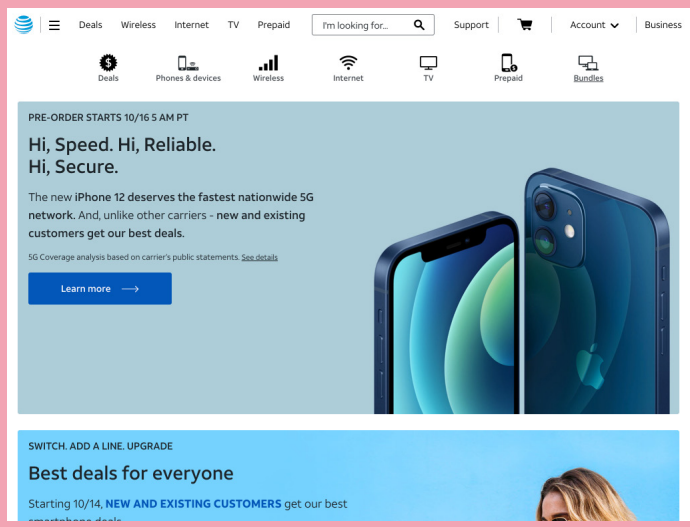
There is a lot of examples showing integration with different tools like Headless CMS, CSS-in-JS, or auth.

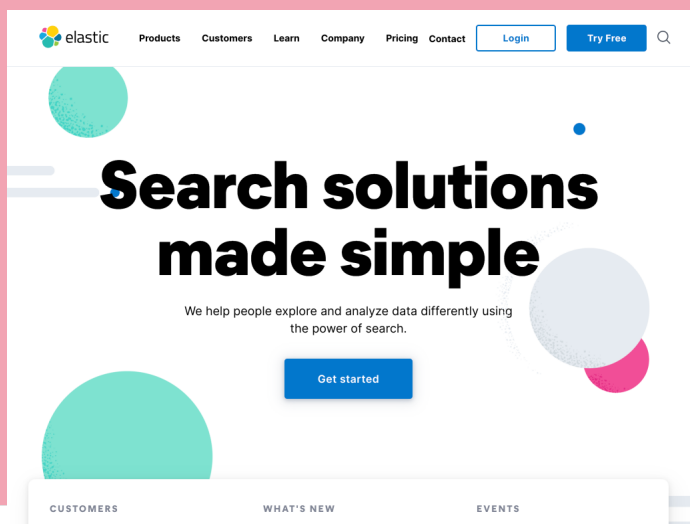
Showcase









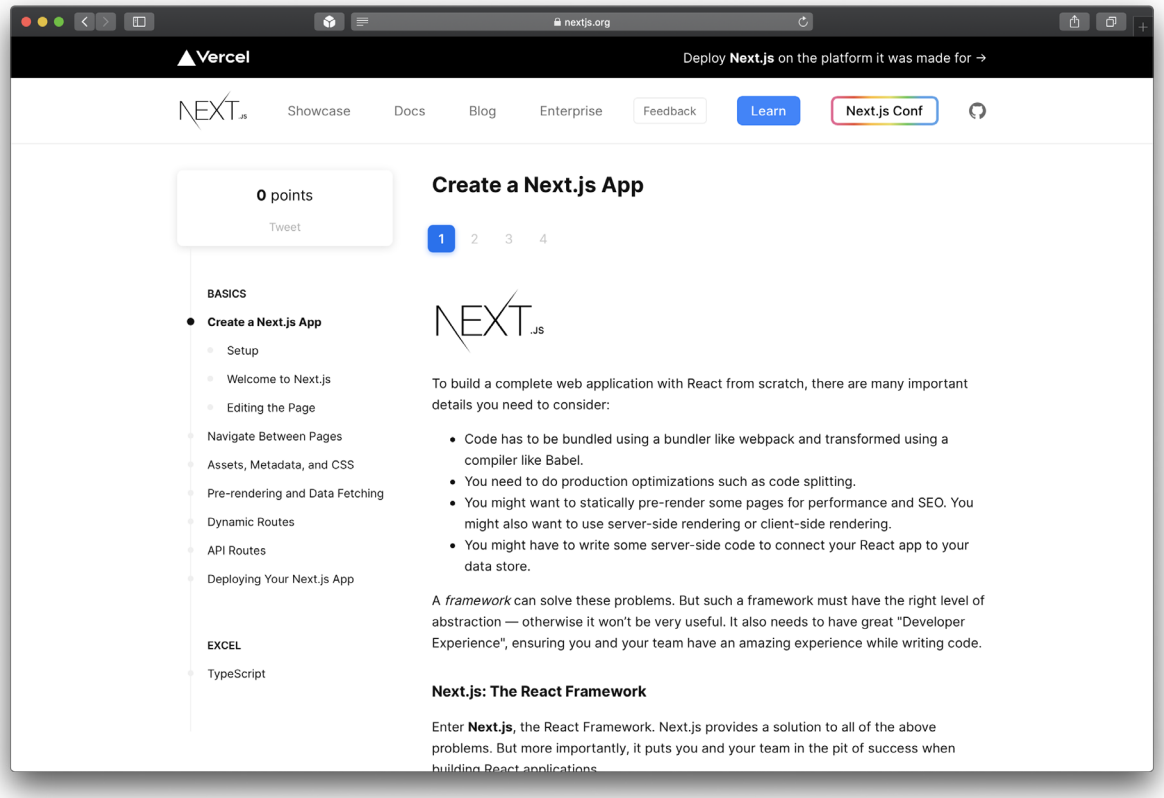


How to get started?

The easiest and recommended way to start a new Next.js project is to use `create-next-app`:

```
npx create-next-app
# or
yarn create next-app
```

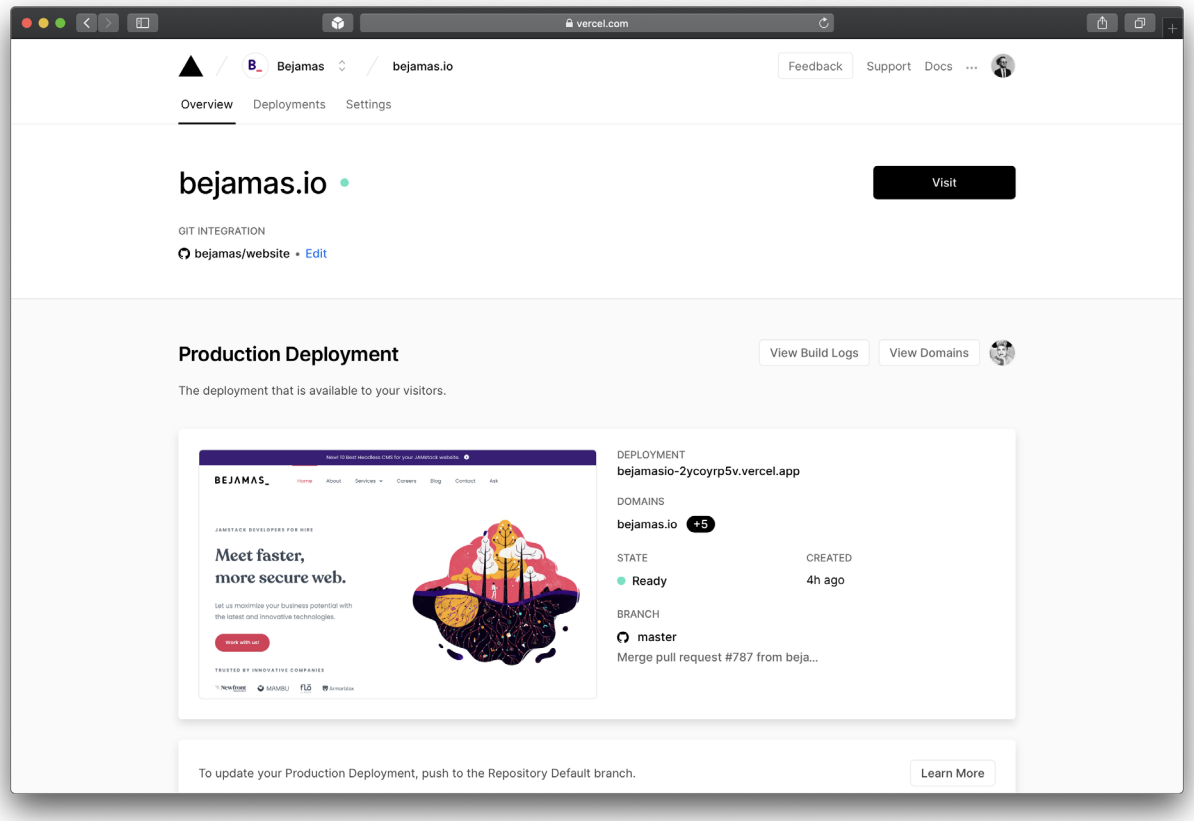
That one command sets up the project for you. Your development environment will be available at http://localhost:3000.
If you want to learn more, I highly recommend the official *Learn Next* tutorial.



It helps to understand how to navigate between pages, add static assets, and fetch data. Once completed you will be ready to build your first Next.js application!

Deploying Next.js

The recommended platform is, of course, Vercel. Its CDN is designed at the edge to support features like incremental static generation, where pages first are populated into the durable store (S3), then the pathname is purged to ensure users are able to see the most recent content.



Preview mode works seamlessly on the Vercel platform, as well as the fallback feature. You can connect your repository and everything works out of the box with no additional config needed.

If for some reason you don't want to use Vercel, it is possible to deploy Next.js on every modern hosting platform like Netlify, Render, AWS, DigitalOcean, and so on. Netlify

maintains a special next-on-netlify package that enables the server-side rendering of pages. Incremental static regeneration, fallback, and preview don't work exactly like on Vercel, so it's not a 1-to-1 replacement.

Conclusion

With Next.js you can build a full spectrum of websites and apps. From simple marketing pages and blogs to eCommerce and PWA with cookie-based authentication.

It's great for projects that require flexibility in how you build specific parts of the website. First-world static site generator with capabilities of dynamic enhancement. Hybrid mode is what really shines. Does it mean it's perfect for everything? Not at all. If you don't need React and don't plan to update the project frequently it might be overkill.

FEATURES	USE CASES
👍 Zero-config	✅ Hybrid apps. If you need both static and server-side rendered pages on each request.
👍 Great perceived performance thanks to instant route changes and prefetching.	✅ eCommerce. Especially, with big traffic and frequent content updates.
👍 TypeScript support.	✅ Landing pages and marketing websites. Especially, if you already use React and have a design system.
👍 Automatic code splitting.	✅ Websites with a lot of interactive React components.
👍 Dynamic API routes.	✅ Websites with a lot of advanced route transitions.
👍 Vercel team constantly works on reducing JS bundle size and optimizes performance.	
👍 Built-in integration with the new Web Vitals metrics. Next.js analytics.	
👍 Hybrid mode: both static generation and SSR.	
👍 Built-in image component and image optimization on demand.	
👍 First-class support for internationalization.	

Gatsby

B_

Gatsby

Gatsby is the most popular static site generator build on top of the most popular Javascript framework - React.js. It was created by Kyle Matthews who later, together with Sam Bhagwat founded Gatsby, Inc. with a mission to make building websites fun.

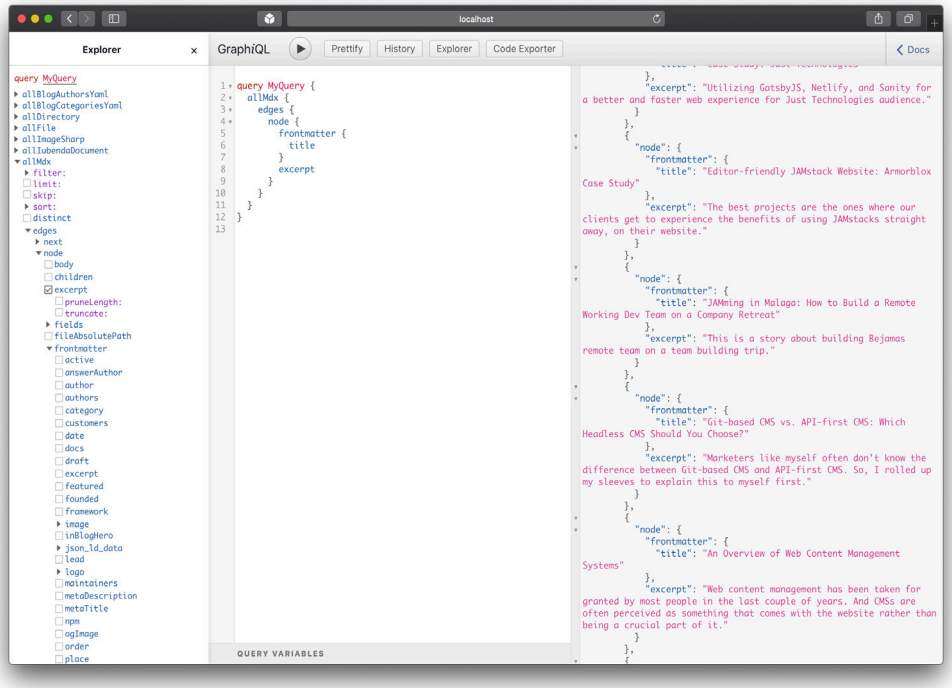
Gatsby Story

GatsbyJS was born in 2015 as a simple static site generator. During the last 5 years, it has become the most popular solution in the market. Gatsby helped Jamstack grow to what it is now.

In 2019 Gatsby, Inc. secured $15M in Series A funding and built its first commercial product - Gatsby Cloud, CI platform that focuses on fast incremental builds. I'll talk about this a bit later. Let's start with the SSG overview.

Content Mesh

One of the most opinionated things in Gatsby is data fetching. Gatsby creates an internal GraphQL API. It's a kind of *middleware* between the content sources and the frontend template.



You can connect Gatsby with every popular Headless CMS. A large collection of `gatsby-source` plugins makes it easy. You can *mesh* sources in a single GraphQL query. Everything becomes one, big graph.

You need to learn GraphQL and Gatsby API to enjoy the *mesh* concept. For simple static websites, it might be too much. Working with GraphQL can be painful, errors sometimes are weird. Debugging is hard. Some people like it, others don't.

If you ask me, GraphQL API is a good way to unify and standardize the way we build products at agencies like ours. No matter what CMS we pick, the way we generate pages and fetch data is similar.

For instance, take a look at how to fetch products from Shopify and content from Contentful:

```
import React from 'react'
import { graphql } from 'gatsby'
const HomePage = ({ data }) => {
 return (
  <div>
   All products:
   {data.allContentfulProducts.edges.map(({ node }) => (
    <div key={node.id}>
     <h2>{node.title}</h2>
    </div>
   ))}
  </div>
 )
}
export const query = graphql`
 query HomePageQuery {
  allContentfulProducts(filter: { node_locale: { eq: "en-US" } }) {
   edges {
```

```
        node {
          id
          slug
          title
        }
      }
    }
    allShopifyProduct {
      edges {
        node {
          id
          title
          productType
          vendor
          variants {
            id
            title
            price
          }
        }
      }
    }
  }
`

export default HomePage
```

You can fetch data from everywhere. Headless CMS, local filesystem, external GraphQL or REST API, Google Spreadsheet, Airtable, and so on. You can write your own `source-plugin` to create a custom integration with your backend.

GraphQL queries execute only during the build. If you change something in the CMS, you need to rebuild the project. You can still fetch data on the client like in any other React app. For example, using SWR or a simple fetch, but it won't be pre-rendered.

File Structure

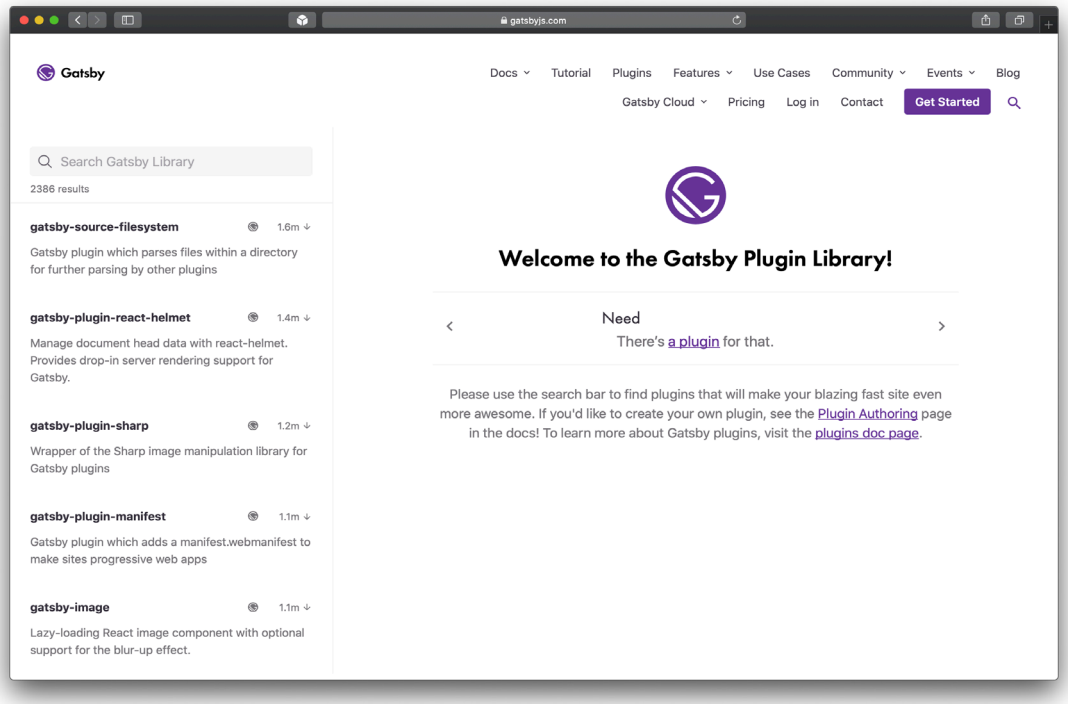
An example file structure shown below:

```
├── gatsby-browser.js
├── gatsby-config.js
├── gatsby-node.js
├── gatsby-ssr.js
├── package-lock.json
├── package.json
├── src
│   ├── html.js
│   └── pages
│       ├── 404.js
│       ├── about.js
│       ├── blog.js
│       ├── contact.js
│       └── index.js
└── static
    └── logo.png
```

The first four files you have access to Gatsby API. In other words, you can modify what Gatsby does under the hood. Add plugins, generate pages from API, adjust SSR, manage what is happening on init render in the browser, and so on. Explore full API's capabilities reading the official docs.

Ecosystem

Over the years the community of Gatsby devs has created thousands of themes, plugins, and starters. Gatsby has the biggest ecosystem in the Jamstack space. The last time I checked the plugins library, there were more than 2400 ready to install packages.

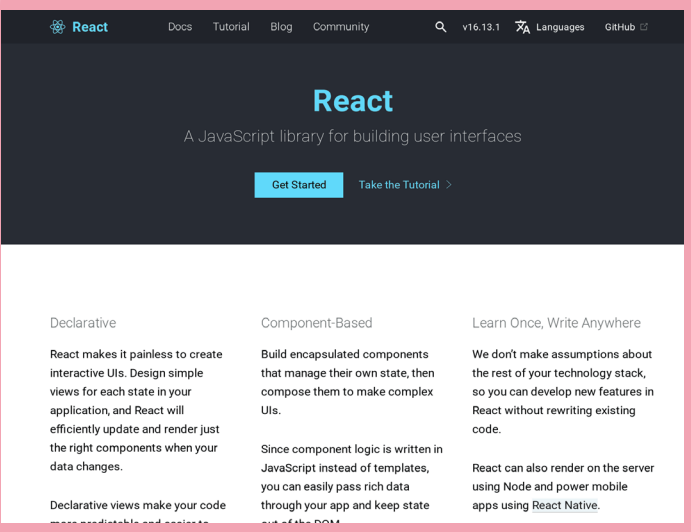


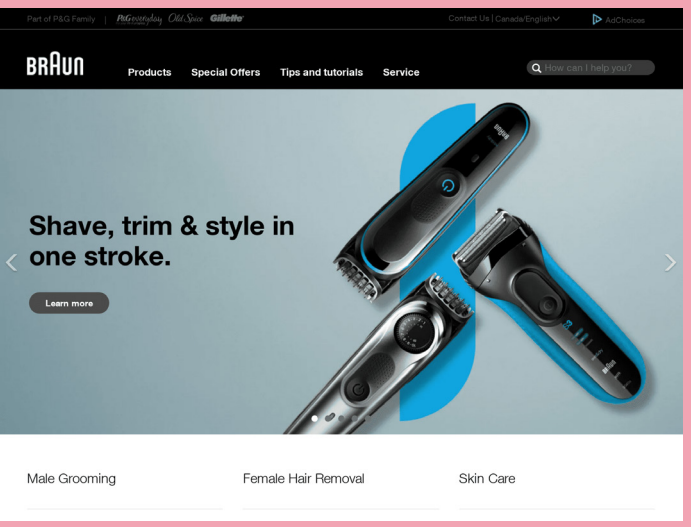
You may like plugins or not. But there is a great possibility that someone already had the same issue as you and published a solution for it.

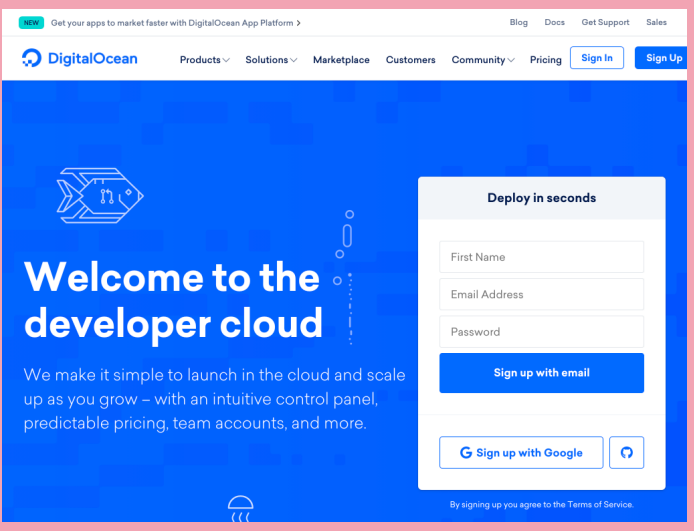
Among the most popular plugins are `gatsby-plugin-react-helmet`, `gatsby-plugin-sharp`, and `gatsby-image`. There are a bunch of plugins that improve the performance of pages like gatsby-plugin-preact. We've talked about **Gatsby plugins** made to tackle performance issues previously. Check it out.

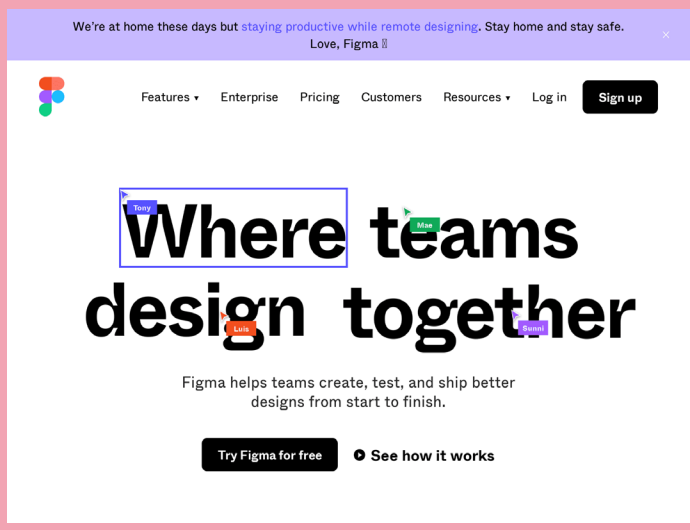
A large ecosystem enhances the developer experience. This basically means boring tasks like generating a sitemap or adding a content security policy already have a solution.

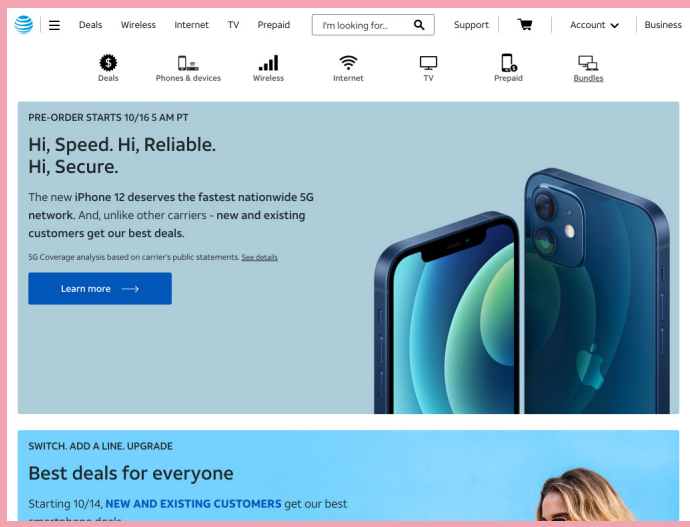
Showcase

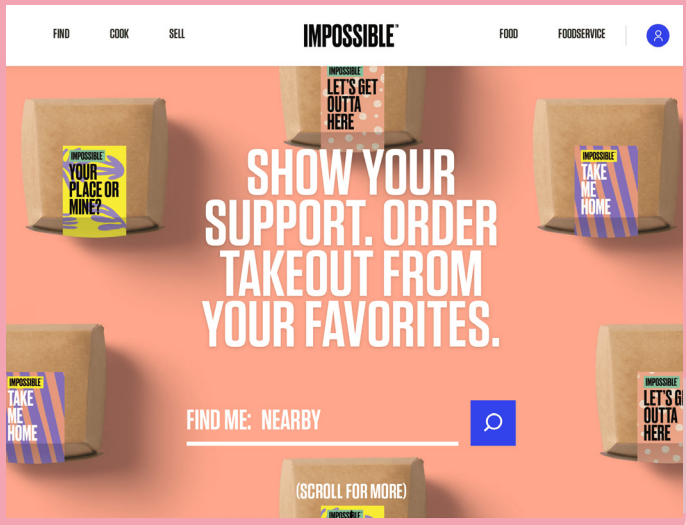

React
Docs
Tutorial
Blog
Community
v16.13.1
Languages
GitHub
React
A JavaScript library for building user interfaces
Get Started
Take the Tutorial
Declarative
React makes it painless to create interactive UIs. Design simple views for each state in your application, and React will efficiently update and render just the right components when your data changes.
Component-Based
Build encapsulated components that manage their own state, then compose them to make complex UIs.
Since component logic is written in JavaScript instead of templates, you can easily pass rich data through your app and keep state
Learn Once, Write Anywhere
We don't make assumptions about the rest of your technology stack, so you can develop new features in React without rewriting existing code.
React can also render on the server using Node and power mobile apps using React Native.


BRAUN
Products
Special Offers
Tips and tutorials
Service
Shave, trim & style in one stroke.
Learn more
Male Grooming
Female Hair Removal
Skin Care


Get your apps to market faster with DigitalOcean App Platform
Blog
Docs
Get Support
Sales
DigitalOcean
Products
Solutions
Marketplace
Customers
Community
Pricing
Sign In
Sign Up
Welcome to the developer cloud
We make it simple to launch in the cloud and scale up as you grow – with an intuitive control panel, predictable pricing, team accounts, and more.
Deploy in seconds
First Name
Email Address
Password
Sign up with email
Sign up with Google
By signing up you agree to the Terms of Service.


We're at home these days but staying productive while remote designing. Stay home and stay safe.
Love, Figma
Features
Enterprise
Pricing
Customers
Resources
Log in
Sign up
Where teams design together
Figma helps teams create, test, and ship better designs from start to finish.
Try Figma for free
See how it works


Deals
Wireless
Internet
TV
Prepaid
Support
Account
Business
PRE-ORDER STARTS 10/16 5 AM PT
Hi, Speed. Hi, Reliable. Hi, Secure.
The new iPhone 12 deserves the fastest nationwide 5G network. And, unlike other carriers - new and existing customers get our best deals.
Learn more
SWITCH. ADD A LINE. UPGRADE
Best deals for everyone
Starting 10/14, NEW AND EXISTING CUSTOMERS get our best


FIND
COOK
SELL
IMPOSSIBLE
FOOD
FOODSERVICE
SHOW YOUR SUPPORT. ORDER TAKEOUT FROM YOUR FAVORITES.
FIND ME: NEARBY
(SCROLL FOR MORE)

How to get started?

To install Gatsby globally on your computer, run the command:

```
npm install -g gatsby-cli
```

The fastest way to get started with Gatsby is to use one of many starters available.

To create a project from the starter, pick one from the gallery, copy its Github URL, and run:

```
gatsby new my-website https://github.com/gatsbyjs/gatsby-starter-default
```

To start the Gatsby project locally, run:

```
cd gatsby-site && gatsby develop
```

Your dev environment will be available on `http://localhost:8000`.

Deploying Gatsby

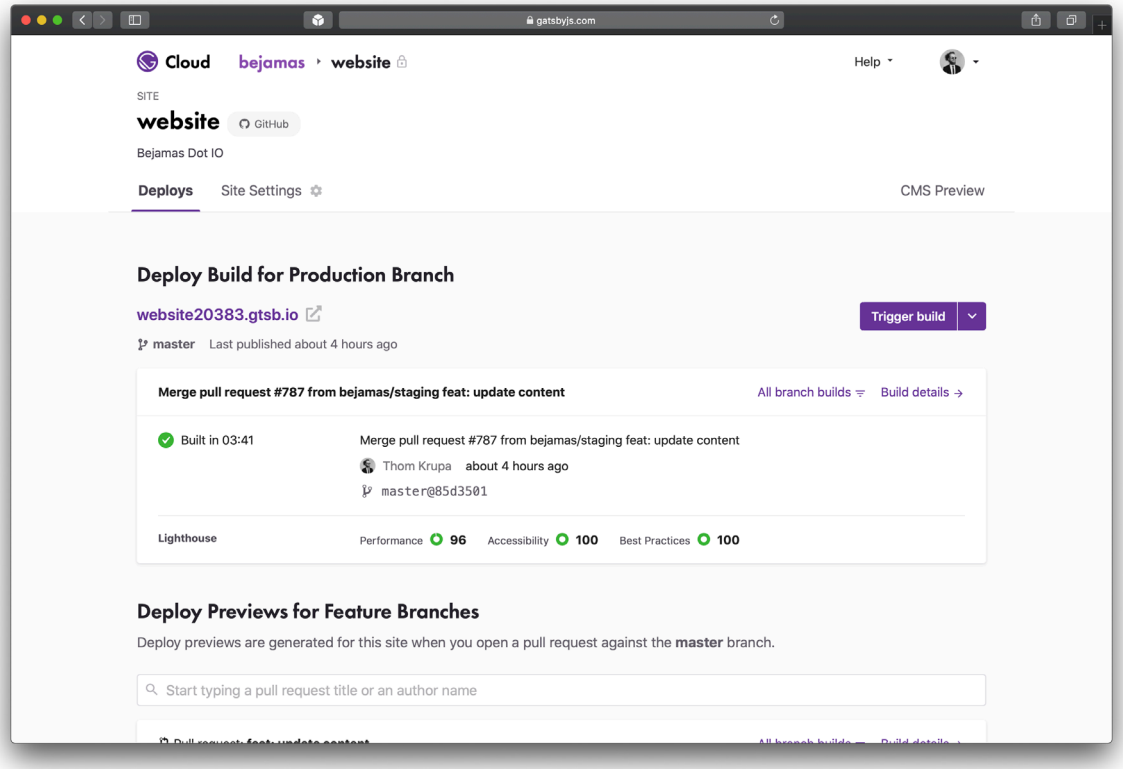
You can deploy Gatsby anywhere. The only command that needs to be executed is:

```
gatsby build
```

The `public` folder contains all static files ready to be published.

Everything works fines when your project contains a small amount of GraphQL queries and images. For a larger set of images - think a few hundreds - the build tends to be very slow. In some cases, CI/CD tool can fail with time out error.

If you want to improve the build time, the easiest solution is to switch to Gatsby Cloud. It is designed to support fast Incremental Build and has better cache management.



Gatsby Cloud handles the building part, but to deliver the website to users, you need to host it somewhere. It's easy to connect projects with any of the popular hosting providers. Gatsby integrates with Netlify, Vercel, Fastly, Google Storage, Azure, Firebase, and AWS S3. Pick the one that matches your existing infrastructure best.

Conclusion

Gatsby.js is a good fit if you use many data sources and want to use one internal API to rule them all. Gatsby, like Next, is based on React. It's easy to create interactive components, but it comes with JS cost. Client-side routing optimizes perceived performance, especially on high-end devices.

It's not the lightest static site generator because it requires loading a bunch of JavaScript. It might be tough to achieve a perfect Lighthouse score, but sometimes that's not the point. You can create a very fast website that will be very fast and pleasant to use for your audience.

FEATURES	USE CASES
👍 Great perceived performance thanks to instant route changes and prefetching.	✅ Marketing websites. Especially if you source content from many APIs and you want to take advantage of React ecosystem.
👍 Huge ecosystem of themes, plugins, and starters.	✅ eCommerce. React helps you with stuff like complex forms or product configurators.
👍 Automatic code splitting.	✅ Pre-rendered static website + SPA. Gatsby doesn't handle dynamic SSR pages, but you can fetch data client-side like in any other
👍 Large community that is keen to help you with issues related to Gatsby.	
👍 Increment Builds and fast builds on Gatsby Cloud.	
👍 Integrates with any type of content source.	
👍 Gatsby team pays attention to accessibility (a18y).	

Eleventy

B_

Eleventy

Ever since Zach Leatherman created Eleventy in 2018 it is gaining more and more followers especially from people tired of JS-heavy static site generators. Eleventy is zero-config by default and works with any project's structure. It is doesn't tie you up with any framework and supports up to eleven different template languages. Eleventy is simple(r). Eleventy adapts to you.

Many modern static site generators use some sort of JavaScript framework, with the most popular ones being React, Vue, and Angular. Eleventy uses JavaScript to generate static pages. No need for a client-side framework.

It doesn't mean you can't use any framework. It means it's optional and it's up to you if you want to use one.

Progressive Enhancement

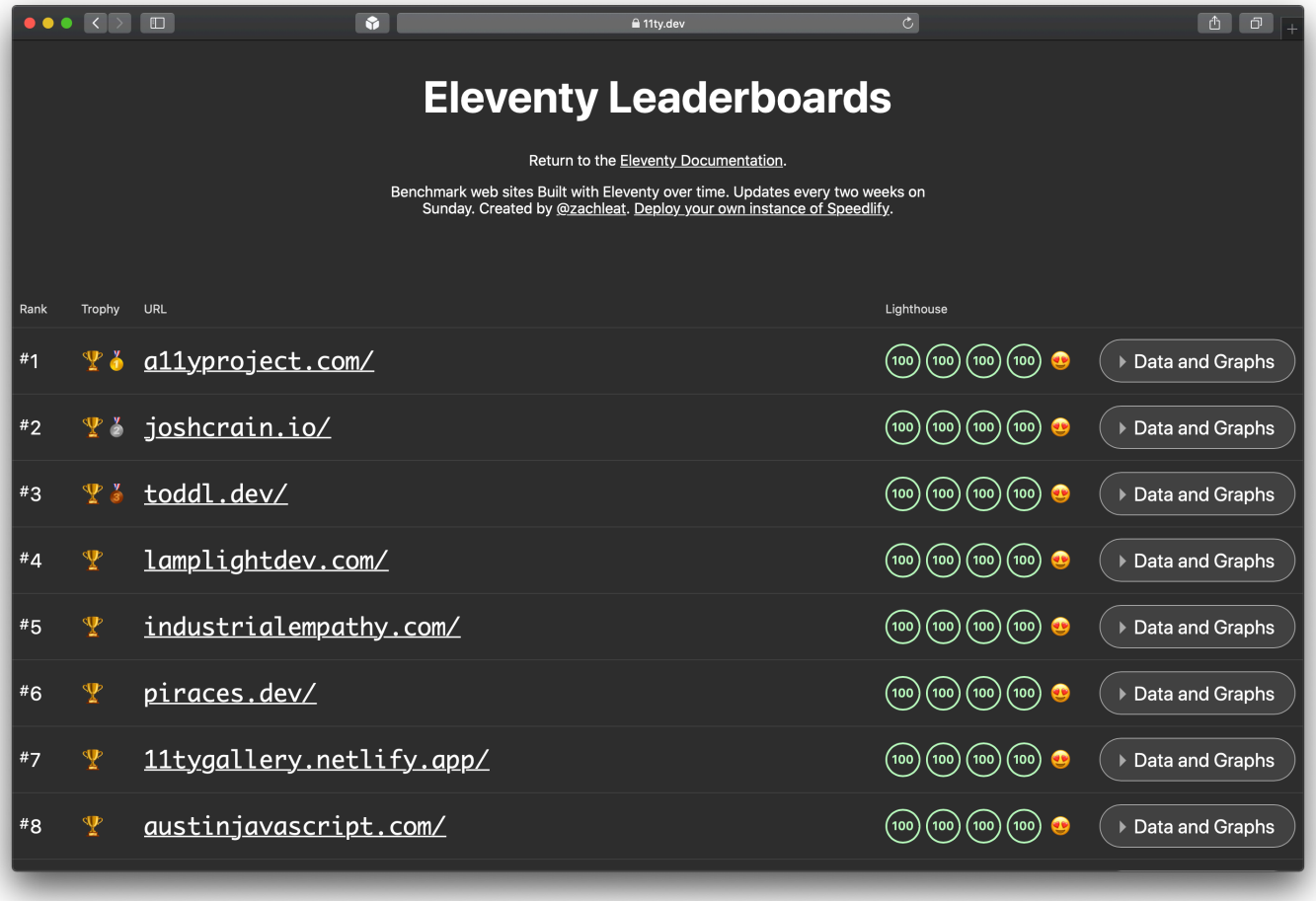
Long *time to interactive* (TTI) is a common issue in JS-based static generators like Gatsby or Next.js. The browser needs to download, parse, and execute a big chunk of JS to hydrate the application. That comes with a cost. It ends up with TTI, and input latency increases.

On a typical website, a lot of components are not very interactive, yet they need to hydrate. React team works on partial hydration that should help solve this issue, but the ETA is not clear.

In Eleventy, you can focus on providing the essential content first and add dynamic functionality step by step. Sometimes, vanilla JavaScript and pure CSS are enough to create things like animations and sliders.

Need for speed

Good performance is an essential part of every modern website. Eleventy developers love the speed. And speed loves Eleventy. Because there is no big load of JavaScript, Eleventy websites achieve great Lighthouse results. Take a look at the leaderboards created by Zach, the author of Eleventy.



Leaderboards update every 2 weeks and display a list of best-scoring projects. Most of those are very simple personal websites. So take this with a grain of salt, i.e., it doesn't really mean Eleventy *is* the fastest SSG on the market. I do believe a lot of those pages would have similar scores using other static site generators.

File Structure

Can be as simple as:

```
└── index.md
```

After running `npx 11ty/eleventy` command, it will turn out to:

```
├── _site
│   └── index.html
└── index.md
```

`_site` is ready to deploy the output folder. Neat, huh?

Let's try to create a simple blog. The example below is based on the official `eleventy-base-blog starter`.

```
├── .eleventy.js
├── .eleventyignore
├── _data
│   └── metadata.json
├── _includes
│   ├── components
│   │   ├── footer.njk
│   │   ├── head.njk
│   │   └── header.njk
│   └── layouts
│       ├── base.njk
│       └── post.njk
├── index.njk
└── posts
    ├── hello-world.md
    └── posts.json
```

Data Sources

Eleventy pulls data from multiple sources. The most straightforward one is Front Matter inside page or templates files.

```
---
title: Post title
author: John Doe
---

<h1>{title}</h1>
<span>by {author}</span>

...
```

Fetching data from Headless CMS

Every real-life project needs some sort of CMS. Not every content team is happy with editing md files and pushing it via git.

To fetch data from API, you need to create a JS file inside `_data` folder - that way, the data will be available globally.

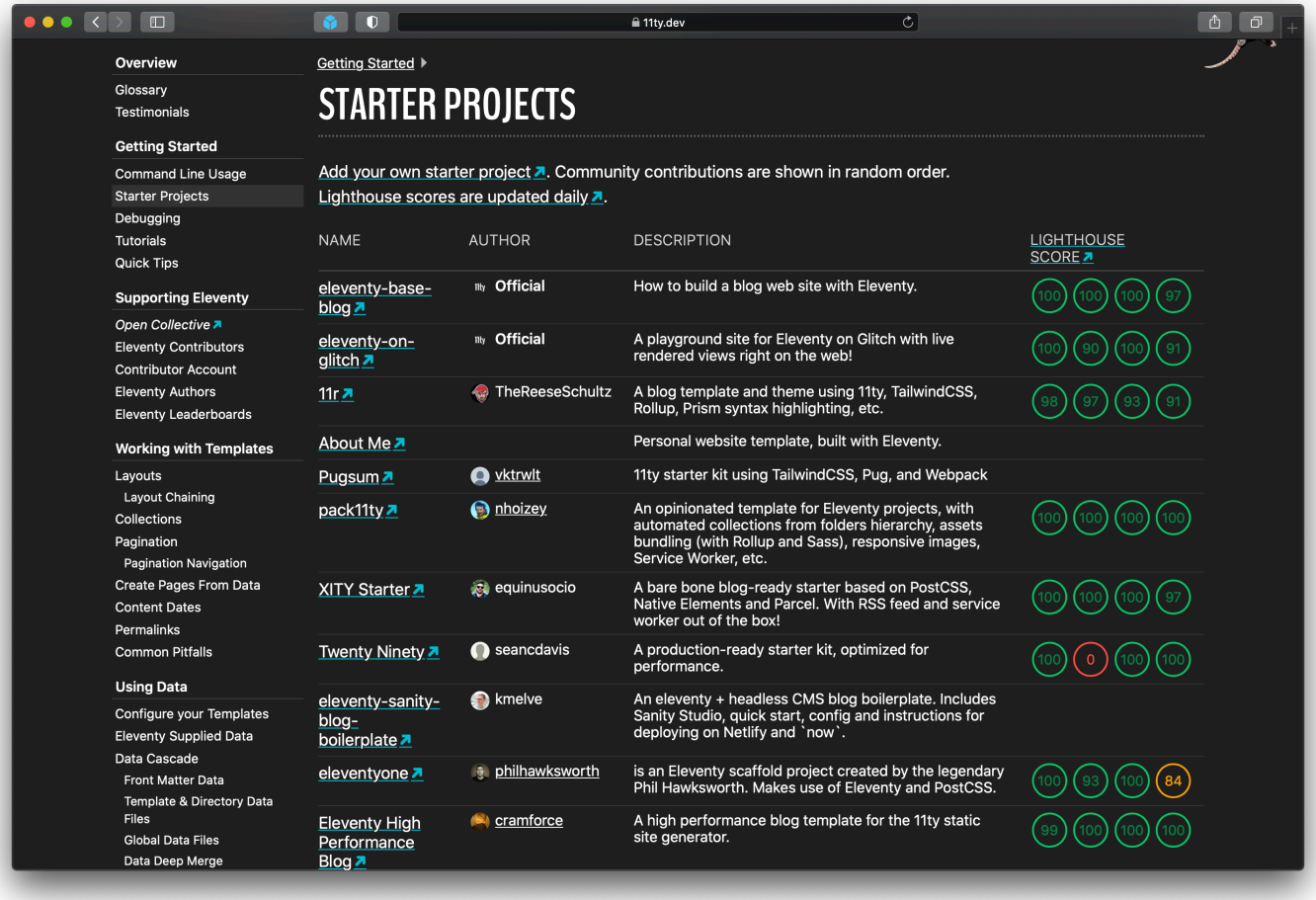
If you want to fetch data and use it in a specific *scope* you need to create a `*.11tydata.js` file inside template or directory folders.

```
module.exports = async function () {
  let response = await fetch(`https://api.domain/posts`)
  let data = await response.json()

  return data
}
```

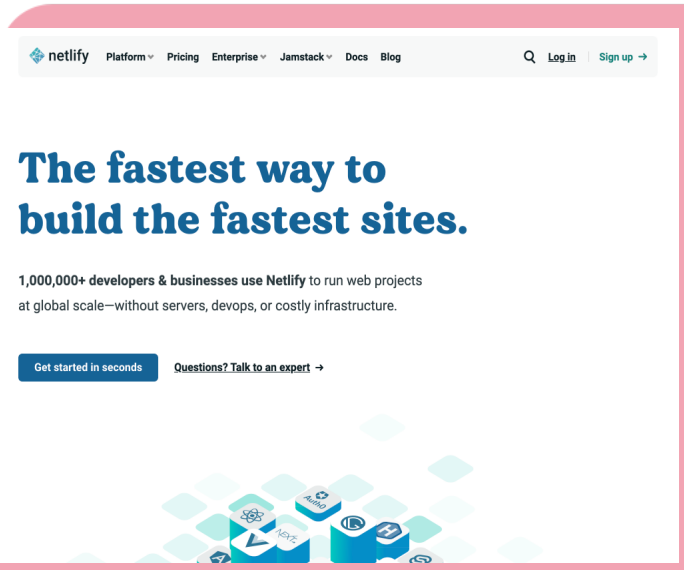
Ecosystem

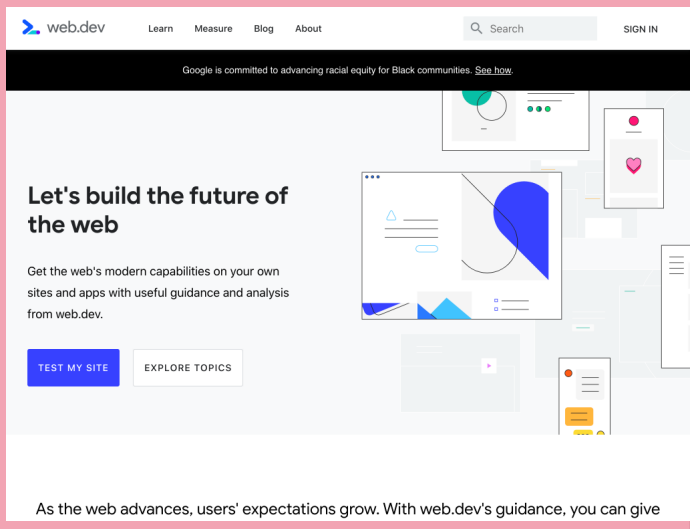
Eleventy does have a list of starters created mostly by the community. Cool thing is that you can see the Lighthouse scores of each started, updated daily.



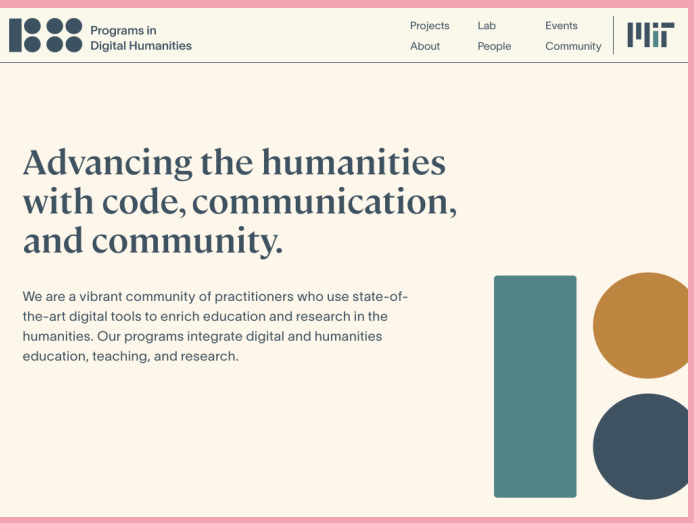
Eleventy doesn't have a lot of ready to use plugins, but there are a few that add functionality like RSS, PWA, Tailwind, Table of Contents, and more.

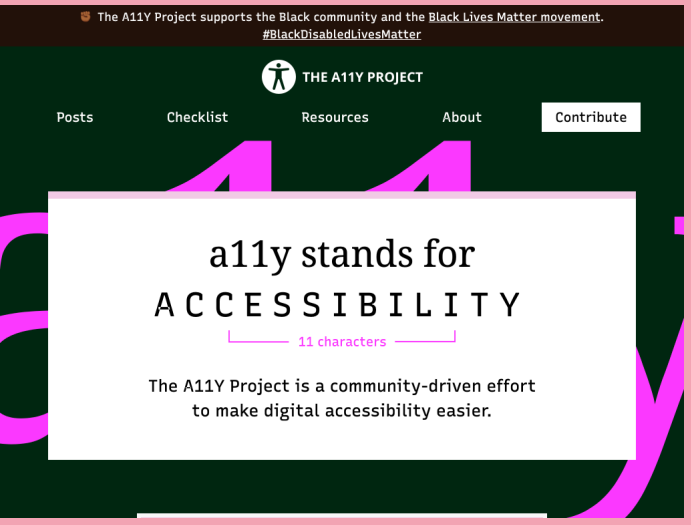
Showcase





As the web advances, users' expectations grow. With web.dev's guidance, you can give





How to get started?

To install Eleventy globally, run the command:

```
npm install -g @11ty/eleventy
```

Next, create a folder and an `index.md` file.

```
mkdir my-11ty-website && echo '# Hello world' > index.md
```

To start the Eleventy project locally, run:

```
eleventy --serve
```

Open `http://localhost:8080` to view your website.

Conclusion

FEATURES	USE CASES
👍 Zero-config, works with your existing project structure.	✅ Marketing websites with progressive enhancement
👍 Supports 11 template languages.	✅ Blogs and documentations
👍 No client-side JavaScript required.	✅ eCommerce websites with mostly static content
👍 Encourages progressive enhancement rather than big JS payloads.	
👍 Fast builds. Generate 50k pages per 1 minute.	

Hugo

B_

Hugo

Written in Go to deliver the fastest build time on the SSG market. Created by Steve Francia and Bjørn Erik Pedersen the first public release was on July 5, 2013. Currently, maintained by Bjørn. Hugo has many fans because of its speed and flexibility that allows you to create a large website that's easy to scale.

Like Eleventy, Hugo encourages progressive enhancement rather than full JS hydration. It's not tied up with any JavaScript framework.

Data fetching

Hugo supports only local flat files like markdown. If you prefer to edit content through a nice user interface, take a look at Git-based CMSs. Popular options are NetlifyCMS and Forestry.

To fetch data from an external API, for example, a headless CMS, use `getJSON` function.

```
<div>
 {{ $posts := getJSON „https://API.DOMAIN/posts" }}
 {{ range first 10 $posts }}
   <article>
     <h2>{{ .title }}</h2>
     <div>{{ .content }}</div>
   </article>
 {{ end }}
</div>
```

File Structure

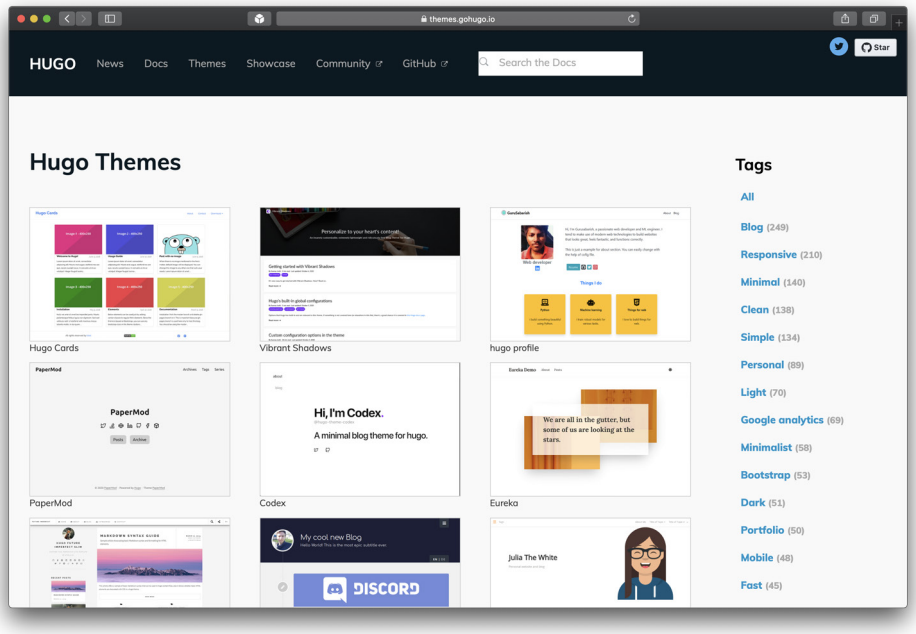
```
├── archetypes
│   └── default.md
├── config.toml
├── content
├── data
├── layouts
├── static
└── themes
```

Hugo has a few high-level concepts such as `archetypes`. Think of it as *templates* or *content schema*, the place when you can define fields for sections of your website.

To learn more about specific folders, check out the official docs.

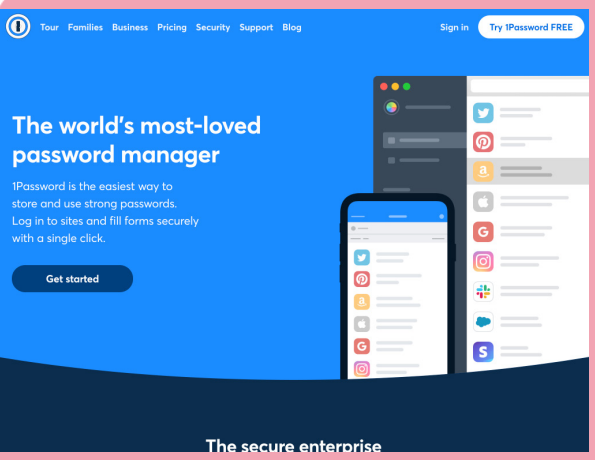
Ecosystem

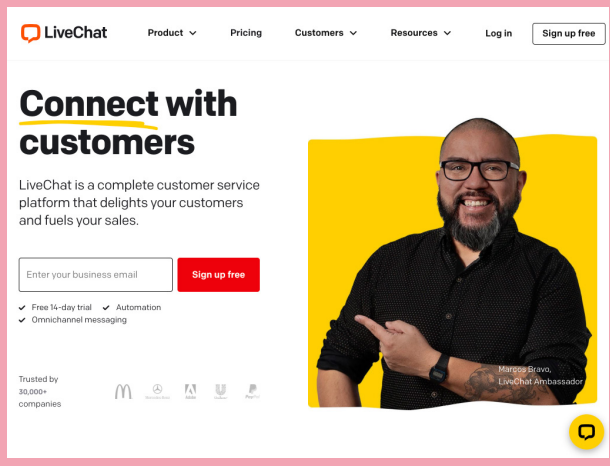
Because of its build performance and simplicity Hugo has gained many fans over the years. In case you have an issue, you can get an answer pretty fast at the official public forum.

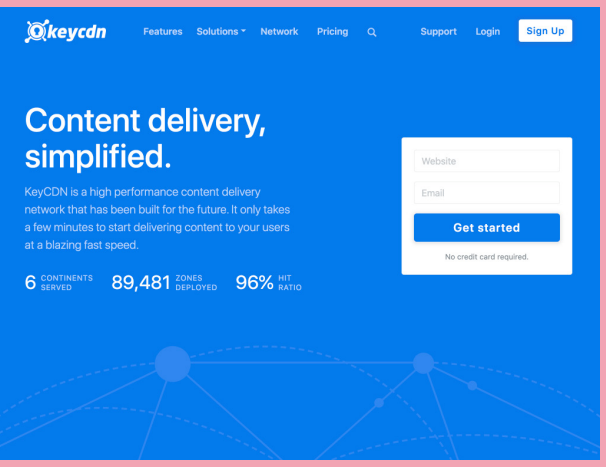


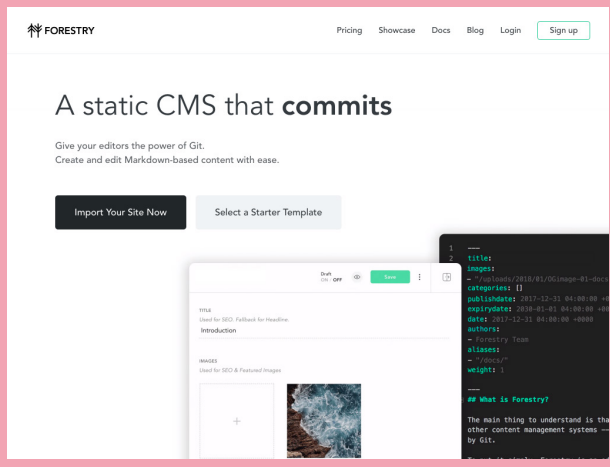
Hugo maintains a large collection of themes. Those are not official projects. Most of them are created by Hugo developers from all over the world. Offered on MIT license - free to use and modify. There is a variety of categories to explore: simple blogs, documentation, landing pages, resume, and many more.

Showcase









How to get started?

Installation depends on the operating system you use, so go and check the installing docs. You don't need to have installed Go to use Hugo.

Once you have Hugo installed, you can run:

```
hugo new site my-hugo-website
```

Now you can choose a theme or create a new one.

```
hugo new theme my-theme
```

To add content, run:

```
hugo new posts/hello-world.md
```

To start development server, run:

```
hugo server -D
```

Your development environment will be available at http://localhost:1313/.

Conclusion

Hugo might feel a little bit old-fashioned for those of you who like to use modern JS frameworks like React, Vue, or Svelte. But Hugo's age is its strength as well. Battle--tested on many big projects by well-known companies gives you the confidence it won't disappear tomorrow.

FEATURES	USE CASES
👍 Incredible fast build time. Hugo is unbeatable when it comes down to building times. I personally don't know of any Static Site Generator faster than Hugo, which makes it a great choice for big websites.	✅ Marketing websites and landing pages
👍 Cross-platform. Hugo has binaries for Windows, Linux, FreeBSD, NetBSD, macOS, and Android for x64, i386, and ARM architectures.	✅ Documentations
👍 No client-side JavaScript required. Hugo is not tied with any JS framework.	

Nuxt

B_

Nuxt

What started as an e-commerce site experiment became a higher-level framework for production-ready Vue applications. The prototype was released to the public a few weeks after the first official release of Next.js. Nuxt was created to solve many of the same problems with building apps for Vue as Next does for React. Today it comes pre-configured with the most crucial elements and intelligent defaults based on well researched best practices to give end-users the best experience.

When first introduced to the public, Nuxt was incredibly lightweight and could only be installed as a template on top of the Vue CLI, a command-line interface to help you develop your application. Even so, it already allowed for server-side rendering as well as to generate a static website.

Nuxt 1.0

It was released on the 8th of January 2018 with the help of a few additional contributors and hosted on Now (Vercel). Their documentation is made with Nuxt.js itself and at the time of this release, was already translated into 6 different languages. New features ranged from layout transitions to improved middleware functionality.

2020 brings more opportunities

In May, the NuxtJS company announced its decision to close a $2m seed round. It's important to note that the creators of the Nuxt.js framework have stated that it will always be an open-source and community-driven project. To stay up-to-date with the latest developments be sure to check out their blog.

File Structure

```
├── assets
│   └── README.md
├── components
│   ├── Logo.vue
│   └── README.md
├── content
│   └── hello.md
├── layouts
│   ├── README.md
│   └── default.vue
├── middleware
│   └── README.md
├── pages
│   ├── README.md
│   └── index.vue
├── plugins
│   └── README.md
├── static
│   ├── README.md
│   └── favicon.ico
├── store
│   └── README.md
├── README.md
├── jsconfig.json
├── nuxt.config.js
├── package-lock.json
└── package.json
```

This is roughly the default structure you'll end up with when using `create-nuxt-app` depending on how you answer the setup questions. Every directory contains a `README.md` file explaining what should go inside and includes a link to the documentation. Nice!

In the above structure, there is a directory called `content` which may seem a bit vague, but it was only added because I chose to install the module `Nuxt Content`, a git-based headless CMS. I will demonstrate how to fetch and use the content from `hello.md` later.

Fetching data

Here is a basic example of how to fetch data from `Contentful`. First we need to install the Nuxt module. We also need the `dotenv` module to manage our local environment variables for security purposes. Nuxt has added `.env` to our `.gitignore` by default.

```
npm install contentful-module @nuxtjs/dotenv
```

Next, we set up our private keys in a `.env` file at the root of our project:

```
CONTENTFUL_SPACE_ID=123
CONTENTFUL_ACCESS_TOKEN=abc
```

Then we let Nuxt know about them and add our Contentful **configuration to** `nuxt.config.js`:

```
require('dotenv').config()

export default {
 env: {
  CONTENTFUL_SPACE_ID: process.env.CONTENTFUL_SPACE_ID,
  CONTENTFUL_ACCESS_TOKEN: process.env.CONTENTFUL_ACCESS_TOKEN
 },
contentful: {
  default: 'master',
```

```
    activeEnvironments: ['master'],
    environments: {
     master: {
      space: process.env.CONTENTFUL_SPACE_ID,
      accessToken: process.env.CONTENTFUL_ACCESS_TOKEN,
      environment: 'master'
     }
    }
   },
   modules: [
    'contentful-module'
   ],
   build: {
    transpile: ['contentful-module']
   }
```

Lastly, we fetch our data inside our page:

```
  export default {
async asyncData ({ app }) {
  const data = {}
  await app.$contentful.client.getEntries({ content_type: 'homepage' })
   .then((res) => { data.intro = res.items[0].fields.intro })
  return { intro: data.intro }
}
```

`asyncData` will merge its return value into the component's local state otherwise known as `data`. We then just need to display it by simply adding the following to our `<template>`:

```
<p>{{ intro }}</p>
```

I am fetching a single field for example purposes, but already you can see the difficulty you will run into when working with the `Contentful` data structure. It is strongly recommended to use GraphQL and Nuxt provides a module, https://github.com/nuxt-community/apollo-module, to use `vue-apollo` for this.

Content by Nuxt

Nuxt provides its own headless CMS which is able to handle Markdown, CSV, YAML, JSON, and XML. As you can see, fetching Markdown from the default `hello.md` is incredibly simple:

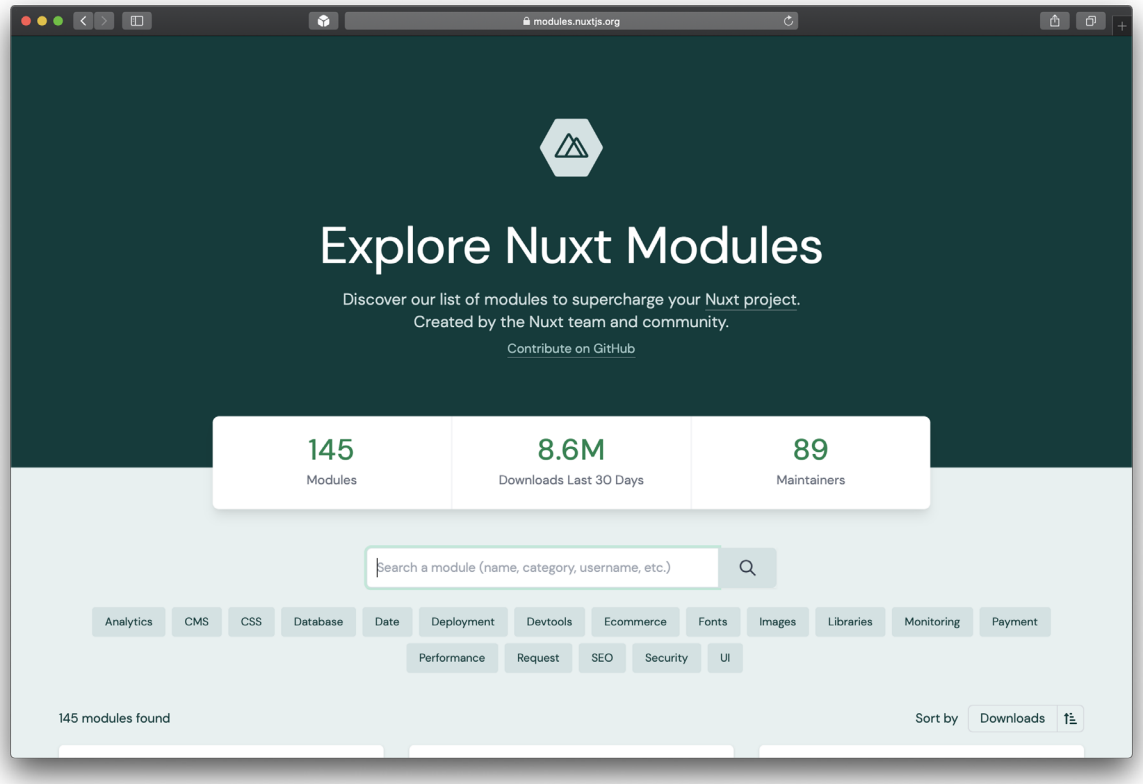
```
async asyncData ({ $content }) {
  const page = await $content('hello').fetch()
  return { page }
}
```

To display content use the provided component:

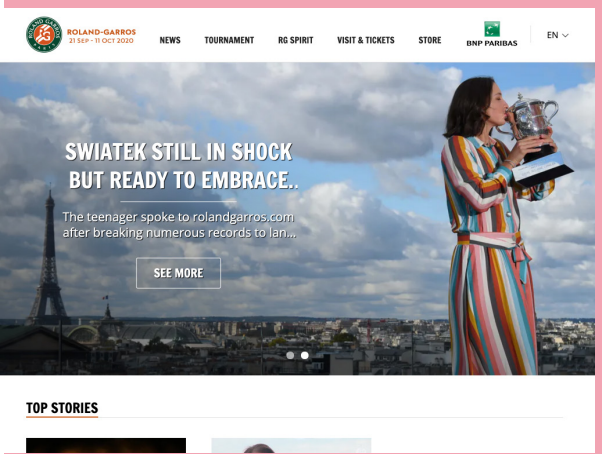
```
<nuxt-content :document="page" />
```

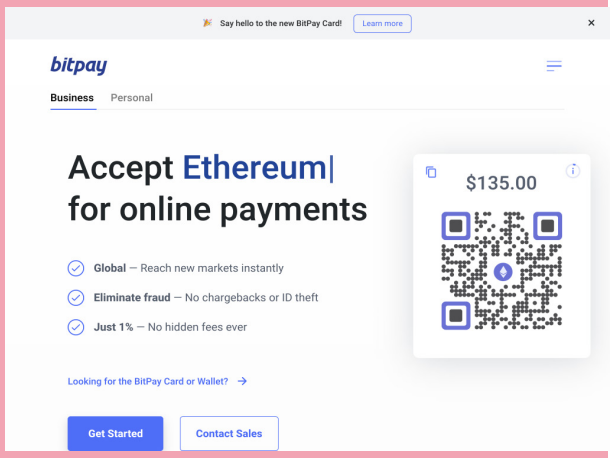
Ecosystem

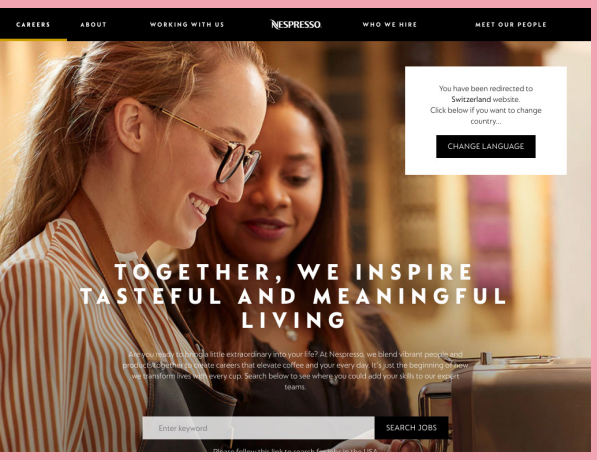
To extend the functionality you can use many available Nuxt modules. The library contains over 140 different modules, from **PWA** integration to various headless CMS like Prismic, Storyblok, Sanity, and many more.

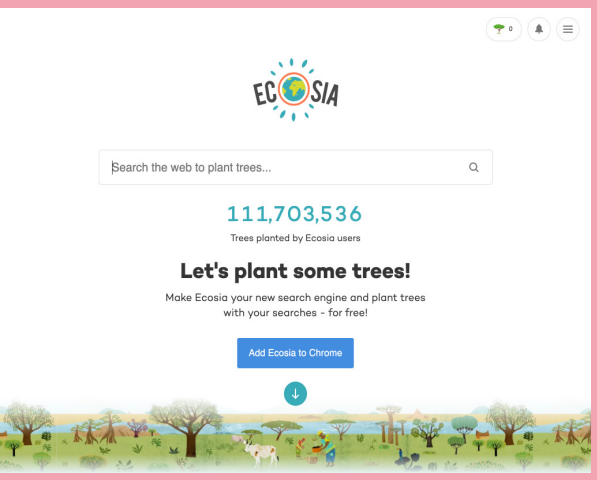

modules.nuxtjs.org
Explore Nuxt Modules
Discover our list of modules to supercharge your Nuxt project.
Created by the Nuxt team and community.
Contribute on GitHub
145
Modules
8.6M
Downloads Last 30 Days
89
Maintainers
Search a module (name, category, username, etc.)
Analytics
CMS
CSS
Database
Date
Deployment
Devtools
Ecommerce
Fonts
Images
Libraries
Monitoring
Payment
Performance
Request
SEO
Security
UI
145 modules found
Sort by
Downloads

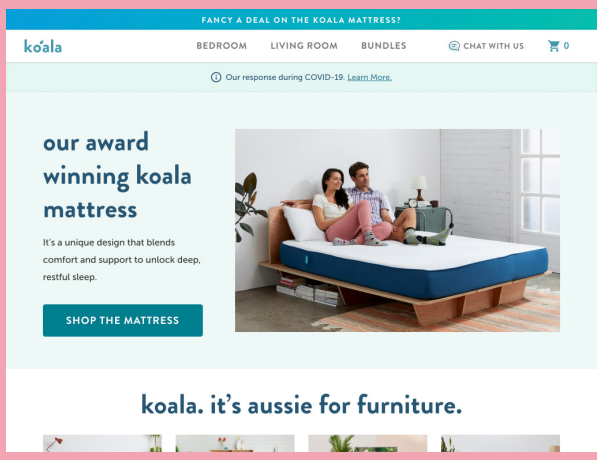
Showcase

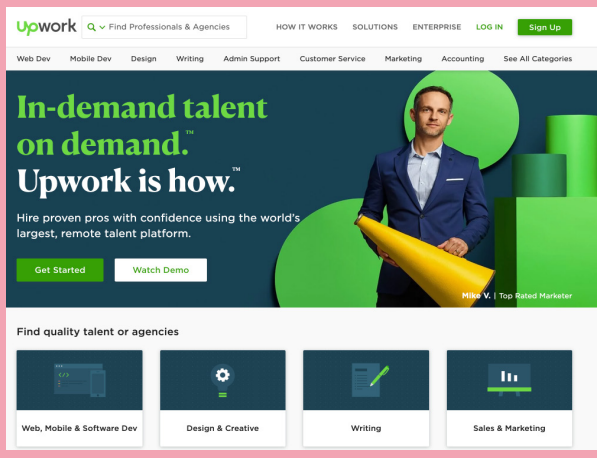

ROLAND-GARROS
NEWS
TOURNAMENT
RG SPIRIT
VISIT & TICKETS
STORE
BNP PARIBAS
EN
SWIATEK STILL IN SHOCK
BUT READY TO EMBRACE..
The teenager spoke to rolandgarros.com
after breaking numerous records to lan...
SEE MORE
TOP STORIES


Say hello to the new BitPay Card!
Learn more
bitpay
Business
Personal
Accept Ethereum
for online payments
Global — Reach new markets instantly
Eliminate fraud — No chargebacks or ID theft
Just 1% — No hidden fees ever
$135.00
Looking for the BitPay Card or Wallet?
Get Started
Contact Sales









How to get started?

The easiest way to get started is by using `create-nuxt-app`:

```
yarn create nuxt-app <project-name>
# or
npx create-nuxt-app <project-name>
```

You will be asked a series of questions about the various options and technologies

you would like to use for this project. Once complete all dependencies will be installed and you can navigate to your project and immediately launch it:

```
yarn dev
# or
npm run dev
```

Deploying Nuxt.js

If you navigate to the faq page and scroll the left-hand side menu you will find a section on deployment. Deploying to some of the most popular platforms like Vercel, Netlify, and AWS requires only a couple of steps and you're good to go.

Conclusion

Nuxt.js is a full-service framework. You can use it as an SSG, SPA, or server-side rendered app. It comes packaged with some of the most advanced features you will find with any JavaScript-based framework. This includes the ability to run middleware before navigating to a new route or even server-side middleware which allows you to register additional API routes without the need for an external server! The options are endless and the flexibility is truly impressive.

FEATURES	USE CASES
👍 Nuxt automatically generates your routes with zero configuration.	✅ Simple static sites to large, complex server-side rendered applications
👍 Preview mode when used as an SSG.	✅ Quickly and easily get a blog or documentation site up and running
👍 Typescript support	✅ eCommerce site
👍 Active community constantly improving the framework	✅ Admin dashboard application
👍 Automagically skip Webpack build step when no code has been changed	

Scully

B_

Scully

Scully is the Angular answer for Jamstack. Created by experts at HeroDevs it is probably the only static site generator based on Google's JS framework, opening the possibilities of Jamstack for all Angular projects.

Scully is quite a fresh solution - the first commit, created by Jorge Cano, was on 12 December 2019. After a few months of work, there is a stable 1.0.0 release, which means it is production-ready for real-world projects.

Angular integration

Scully integrates with any existing Angular project - you only need to run one setup command. This approach makes it trivial to turn your website static.

An interesting feature of Scully is the implementation of machine-learning - it uses Guess.js to find all routes of your site and pre-render each page to plain HTML and CSS.

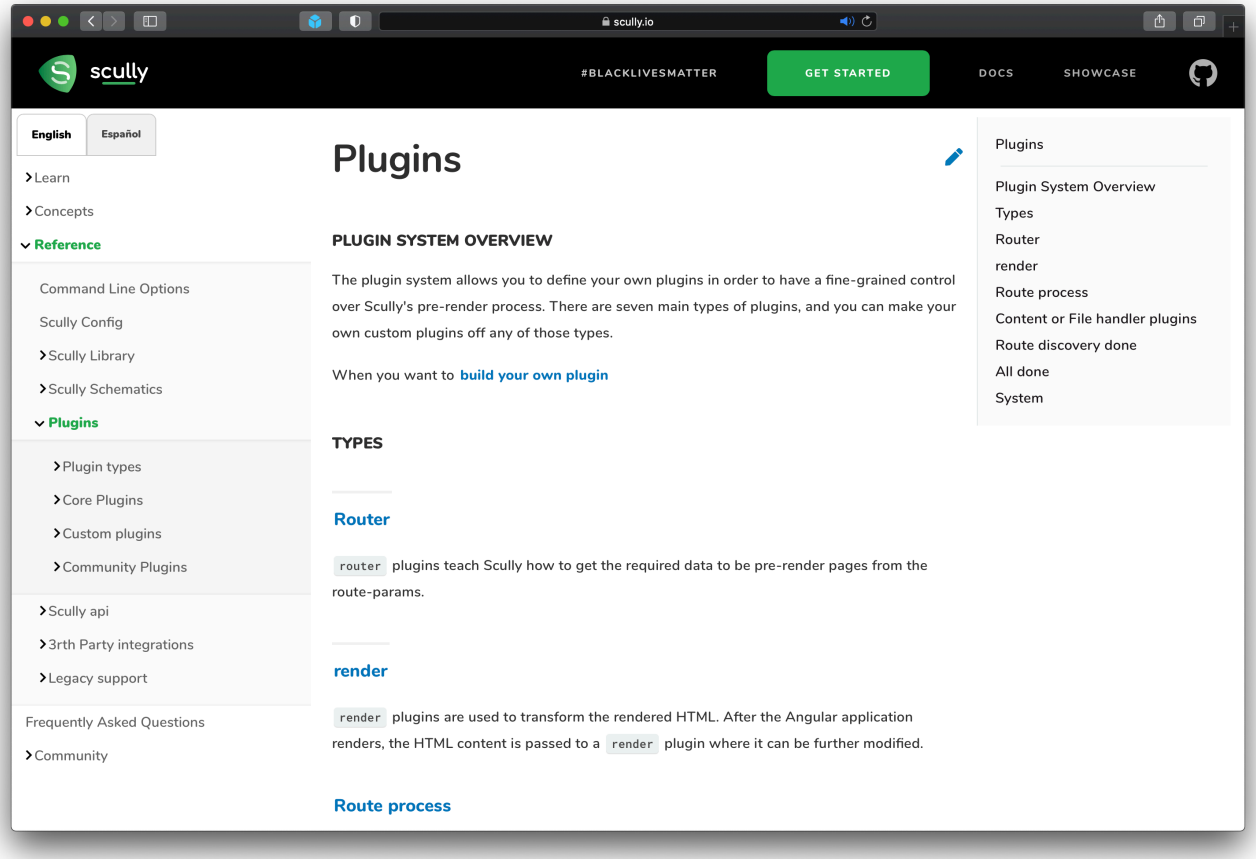
File structure

```
├── ... Angular application files
├── dist
├── .scully
│   └── settings.yml
├── scully
│   ├── plugins
│   │   └── plugin.ts
│   └── ts.config.json
└── scully.[your-project-name].config.ts
```

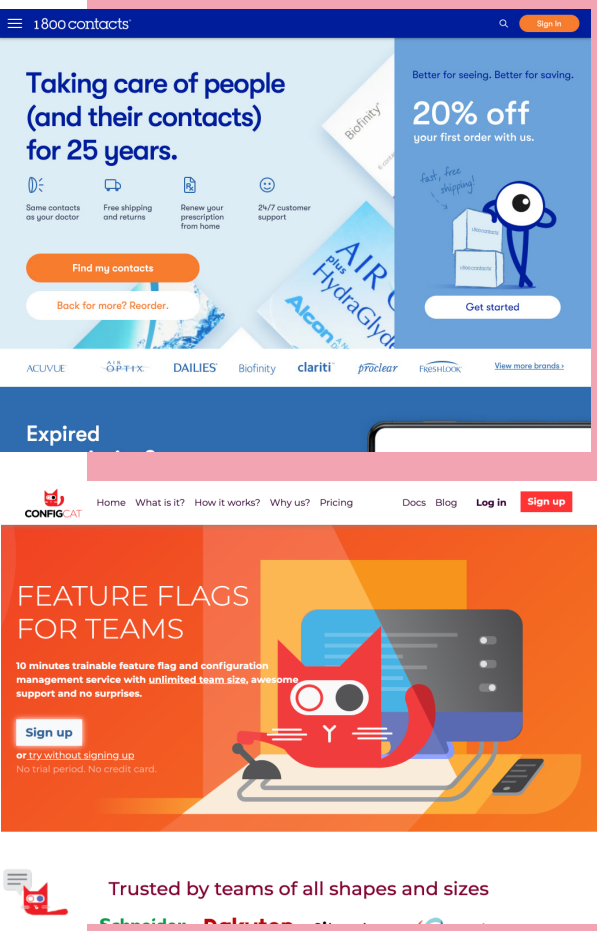
Running Scully in your project creates only a few elements and all of them are simple configuration files where you can change the project's settings, add plugins, and customize the tool's behavior.

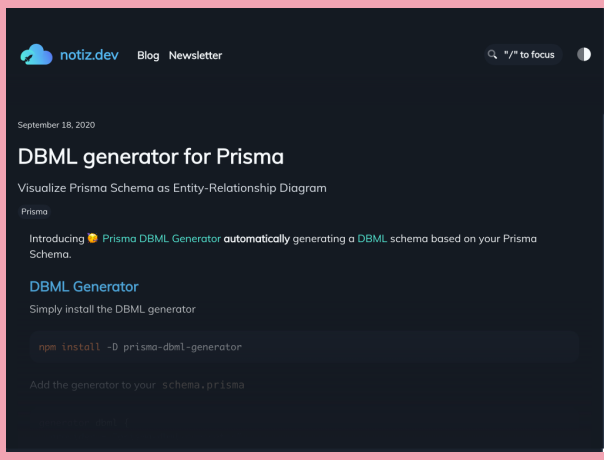
Ecosystem

Despite Scully being quite young it already provides a way of creating custom plugins, integration with API, and 3rd-party scripts. There is a small library of plugins with an in-depth description of the installation and setup of each one.



Showcase





How to get started?

If you already have an Angular application created, run just one command:

```
ng add @scullyio/init.
```

Alternatively, for non-Angular workspaces, install Scully package with

```
npm install @scullyio/init
```

and then create a project using

```
nx g @scullyio/init:install -- --project=<projectName> command.
```

To build the project, run:

```
ng build --prod
```

After the project is built you can run Scully using npm run scully. That's it. A more in--depth description of the process, along with solutions to potential problems, you'll find in Scully's documentation.

Conclusion

For now, Scully is the only static site generator made specifically for Angular. It is a great choice if you want to turn an existing project into a static website or you just want to try out the Jamstack approach in Angular enviroment.

FEATURES	USE CASES
👍 A flexible and extensible plugin system to bake your own functionality into Scully's processes.	✅ Existing Angular projects.
👍 Use of machine learning to find routes of your website and create plain HTML and CSS.	
👍 Easy setup and integration with existing Angular projects.	

Gridsome

B_

Gridsome

Built for the Jamstack workflow and highly inspired by Gatsby (with a similar architecture) but in Vue.js style. Optimized for speed and ease of use. Gridsome, while still new, is a solid choice capable of integrating with any data source you want.

On the 10th of October 2018, the Gridsome team officially announced the first beta release soon to be published. The goal was to create a Vue.js alternative to Gatsby and much of their popularity they own because they were able to build on and learn from the knowledge and advancements of the Gatsby team that had already worked for over three years.

File structure

```
├── src
│   ├── components
│   │   └── README.md
│   ├── layouts
│   │   ├── Default.vue
│   │   └── README.md
│   ├── pages
│   │   ├── About.vue
│   │   ├── Index.vue
│   │   └── README.md
│   ├── templates
│   │   └── README.md
│   ├── favicon.png
│   └── main.js
├── static
│   └── README.md
├── README.md
├── gridsome.config.js
├── gridsome.server.js
├── package.json
└── yarn.lock
```

If you open your `gridsome.config.js` file, all you will see is some comments and these settings:

```
module.exports = {
  siteName: 'Gridsome',
  plugins: []
}
```

Another critical file to take note of is `gridsome.server.js`. This one allows you to hook into Gridsome Server where you can access the various APIs, load data from local files or external APIs or programmatically create pages.

Data fetching

Let's see how we might fetch data from `Contentful` for a single page. First, we need to install the Contentful plugin:

```
npm install @gridsome/source-contentful
```

Next, we set up our private keys in a .env file at the root of our project:

```
CONTENTFUL_SPACE_ID=123
CONTENTFUL_ACCESS_TOKEN=abc
```

It's important to note that Gridsome does not provide you with a Git repository, meaning not only you'll need to set one up for yourself, but you must make sure to add .env* to your .gitignore file. This is absolutely critical so that you do not share your private keys. The reason for the * is you can have different variables for different environments. Use the filename convention '.env.developmentand.env.production`.

To use the `Contentful` plugin modify yourgridsome.config.js configuration. It should look like this:

```
module.exports = {
 siteName: 'Gridsome',
 plugins: [
  {
   use: '@gridsome/source-contentful',
   options: {
    space: process.env.CONTENTFUL_SPACE_ID,
    accessToken: process.env.CONTENTFUL_ACCESS_TOKEN,
    host: 'cdn.contentful.com',
    environment: 'master',
    typeName: 'Contentful'
   }
  }
 ]
}
```

Like Gatsby, Gridsome provides us with a GraphQL data layer. If you are unfamiliar with GraphQL there is a handy playground that you can access at `http://localhost:8080/___explore`. After running `gridsome develop` that will pretty much autocomplete a query for you to fetch your Contentful data. Here's how to structure your query in `Index.vue`:

```
<page-query>
query {
 products: allContentfulProducts {
  edges {
   node {
    id
    title
   }
```

```
  }
 }
}
</page-query>
```

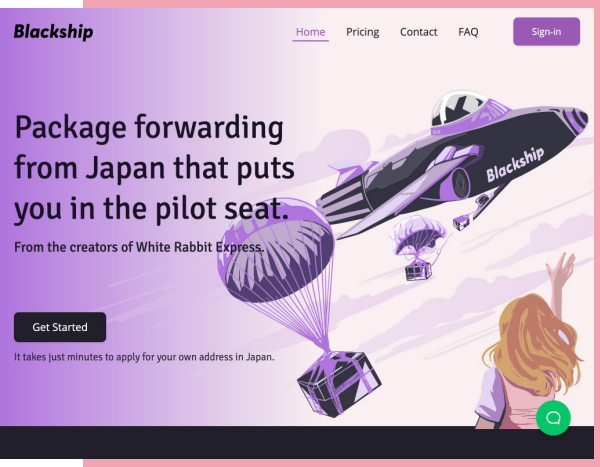
To list our products we simply loop through the `edges` in our `<template>`:

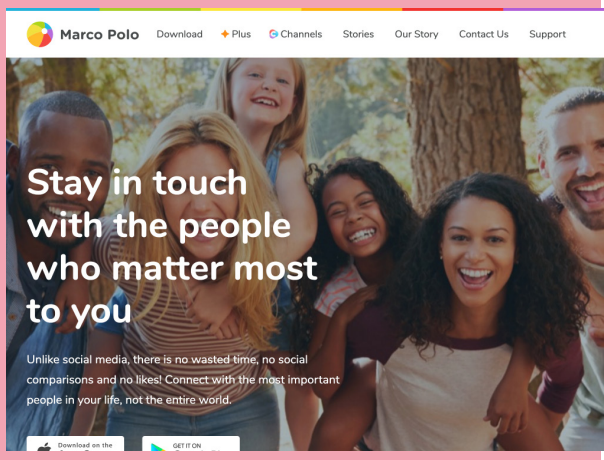
```
<ul>
 <li v-for="edge in $page.products.edges" :key="edge.node.id">
  {{ edge.node.title }}
 </li>
</ul>
```

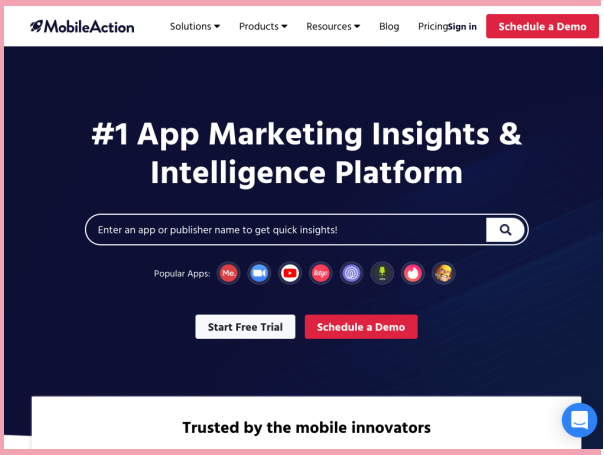
Ecosystem

Gridsome's collection of plugins doesn't come anywhere close to that of Gatsby's, but there are a number of useful ones that can help you connect to a popular CMS of your choice, set up `Google Analytics` and generate a sitemap for example.

Showcase







How to get started?

As with Gatsby, the first step is to globally install the Gridsome CLI:

```
npm install --global @gridsome/cli
```

They do have a collection of 3 official Starters and 20 others added by the community. I recommend you use the default starter replacing new-project with your project name of choice:

```
gridsome create new-project
```

Navigate to your project directory and run:

```
gridsome develop
```

You should now be able to open your site at `http://localhost:8080`.

Deploying Gridsome

The Gridsome team recommends connecting a deploy service that builds your site from a selected Git-based repository. The top services listed include AWS Amplify, Vercel, and Github Pages. At the top of the list, however, is Netlify and the process could not be simpler as explained here.

Conclusion

Gridsome is extremely lightweight and a perfect choice for getting a simple static site up and running quickly. It can also handle complexity and reports to be able to "generate thousands of pages in seconds". Gridsome could be worth considering for large sites as well. But remember, it's still early days for this SSG. Personally, I think Nuxt.js is a safer bet for a Vue-based site. Keep an eye out for future improvements as the Gridsome team works on new versions.

FEATURES	USE CASES
👍 Integrates with any type of content source	✅ Small to large static sites optimized for performance out-of-the-box
👍 Automatic code splitting.	✅ SPA-feeling and PWA front-end solutions for the headless CMS of your choice.
👍 Great perceived performance thanks to instant route changes and prefetching.	
👍 Ecosystem of useful plugins.	

Jekyll

B_

Jekyll

Jekyll is a static site generator written in Ruby. It uses the basics (HTML, Markdowns, Liquid templating) to quickly create fast websites. That puts it at a bit of a disadvantage in the current JavaScript-hungry web world, but despite that, it is still one of the most popular and well-known SSGs. Its popularity is high not only due to speed and simplicity, but also its involvement and adoption in the functionality of Github Pages.

The first version of Jekyll was published on December 19th, 2008. For many people, it is a date that started the trend towards the static web. Jekyll's community quickly grew and currently, there are almost one thousand contributors, not including a plethora of free tools, plugins, and resources. In 2017 Jekyll claimed first place in the Top Ten SSGs by Netlify, citing its popularity, speed, and great support as the reason.

Best features

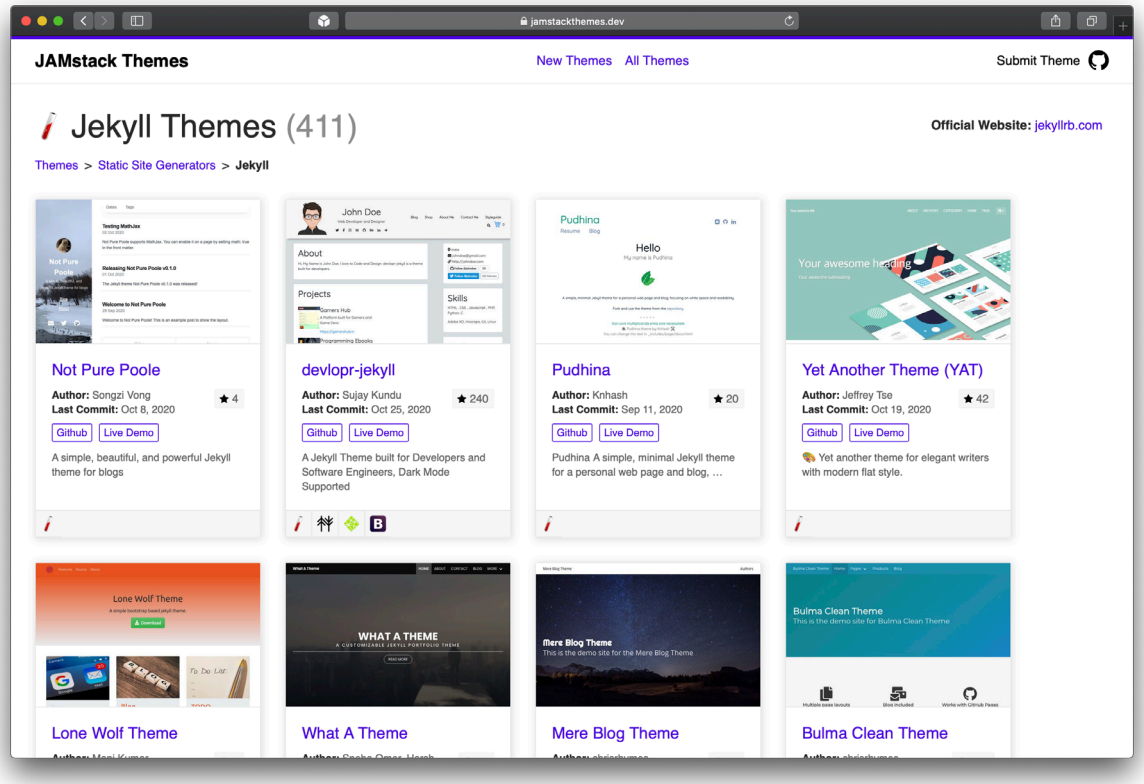
The idea couldn't be simpler: Jekyll takes your content (markdown files) and based on defined templates (HTML with Liquid tags and CSS) generates a static website ready to be served. Its straightforwardness is great for creating blogs or small personal projects.

File structure

```
├── _config.yml
├── _data
├── _drafts
├── _includes
├── _layouts
├── _posts
├── _sass
├── _site
└── index.html
```

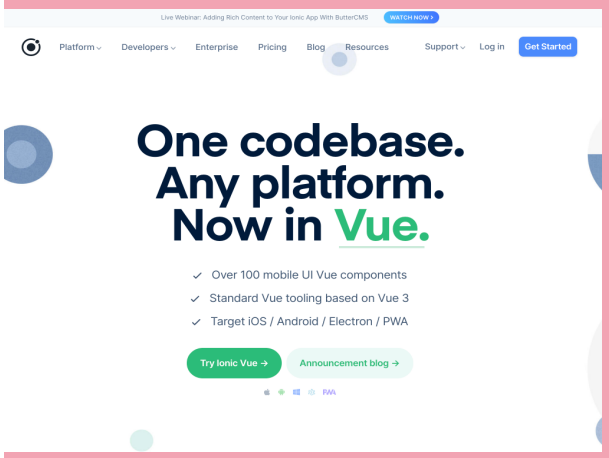
Most of the folder names speak for themselves. In short - dynamic content is stored in the `_posts` folder, and based on the information from `_data`, templates from `_includes` and `_layouts`, Jekyll generates static files and puts them into the `_site` folder. For an in-depth description of each folder visit this documentation page.

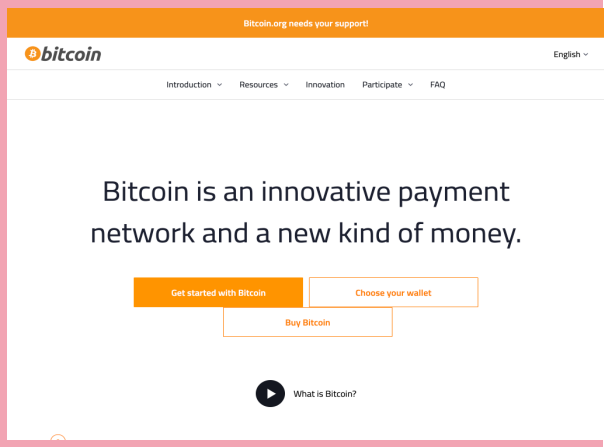
Ecosystem

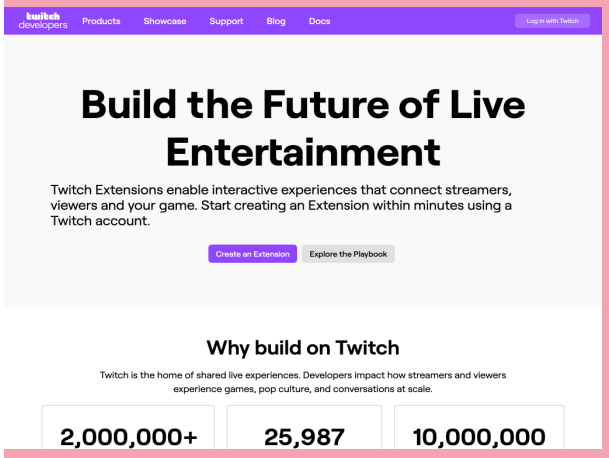


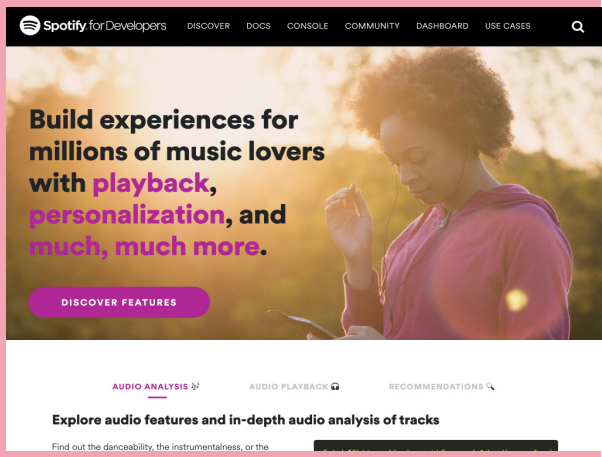
There's a bunch of useful plugins, themes, and resources created by the Jekyll community. It's not as vast compared to other SSGs, but it is definitely enough for most use-cases.

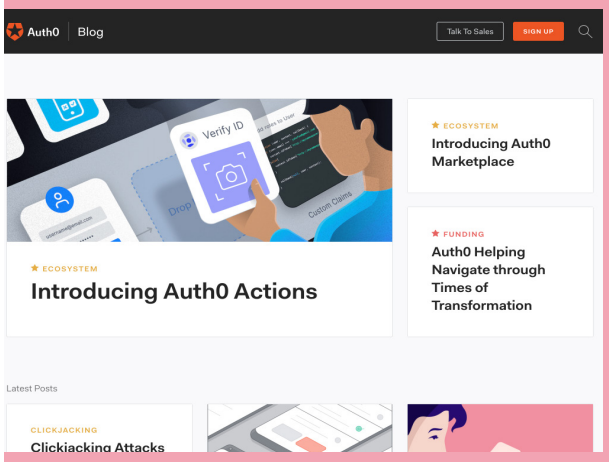
Showcase

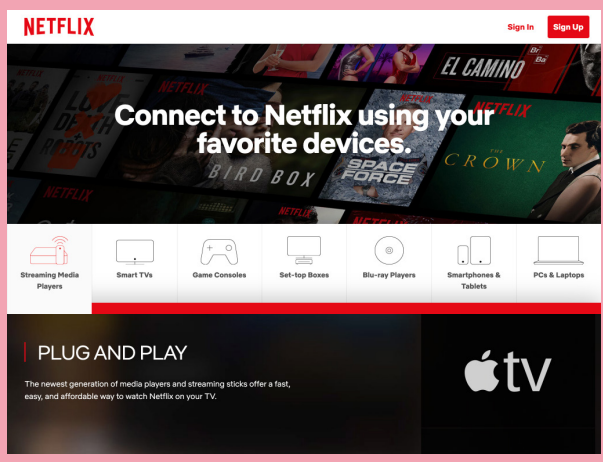












How to get started

There's a bit of setup required, especially if you never have used Ruby before. It also depends on the OS you're using. Luckily there's a step-by-step guide on how to do all the setup necessary: https://jekyllrb.com/docs/.

For Windows, it is quite easy, even if it's not an officially supported platform:

1. Download and install a Ruby+Devkit version from RubyInstaller Downloads

2. Run ridk install command, that will install gems, with all necessary extensions

3. In new command prompt run gem install jekyll bundler

4. Everything's ready, now you can create an example project.

To create a new project:

1. Run jekyll new my_project

2. Go into the newly created directory: cd my_project

3. Run bundle exec jekyll serve to build the website and make it available at http://localhost:4000. You will be presented with the default template website.

Conclusion

Jekyll is one of the simplest static site generators you could find. It is a great alternative to traditional CMSs, especially for websites with a lot of static content. It is also the best SSG when you are not an experienced programmer or don't want to deal with high-level concepts. The huge Jekyll community provides many resources and makes the implementation of many use-cases trivial.

FEATURES	USE CASES
👍 Great for small, content-only websites (blogs, documentation, portfolios, etc.)	✅ Small, content-only websites (blogs, documentation, portfolios, documentation, etc.)
👍 Lightweight and fast	✅ Simple projects where you want to avoid the use of JavaScript or any complex frameworks - just HTML, CSS, and Markdown.
👍 Free hosting with Github Pages. Use of GitHub makes your content automatically versioned in Git. In addition, the platform takes care of SEO, redirects, and SSL certificates.	
👍 Very good documentation makes it easy to get into. The growing community constantly creates new tools, plugins, and resources.	
👍 Comparably low skill floor - Jekyll is easy to learn and use due to its simplicity and use of basic concepts (HTML, CSS, markdown, templating). There's no need to learn whole frameworks, state management, concepts like virtual DOM, rehydration, etc.	
👍 Jekyll's philosophy ensures that it will be relevant for quite some time, even if, by web's standards it is already a grandpa.	

Bridgetown

B_

Bridgetown

Bridgetown was born in March 2020 as a fork of Jekyll. Web studio backing this project was focused on creating something more appropriate to the new Jamstack approach.

Bridgetown gets rid of all deprecated Jekyll configurations. It comes with a default configuration of Webpack to handle building and exporting frontend assets such as Javascript and Typescript, CSS/SCSS, and related files that are imported through Webpack (fonts, images, etc.)

Data Sources

Sourcing data is very straightforward in terms of local files. Default config comes with support for posts in `_posts` folder. What makes making blog-aware websites very fast.

To fetch data from an external source you need to know the basics of Ruby, as it required writing a function (plugin in fact) which will be run during the build process, for example:

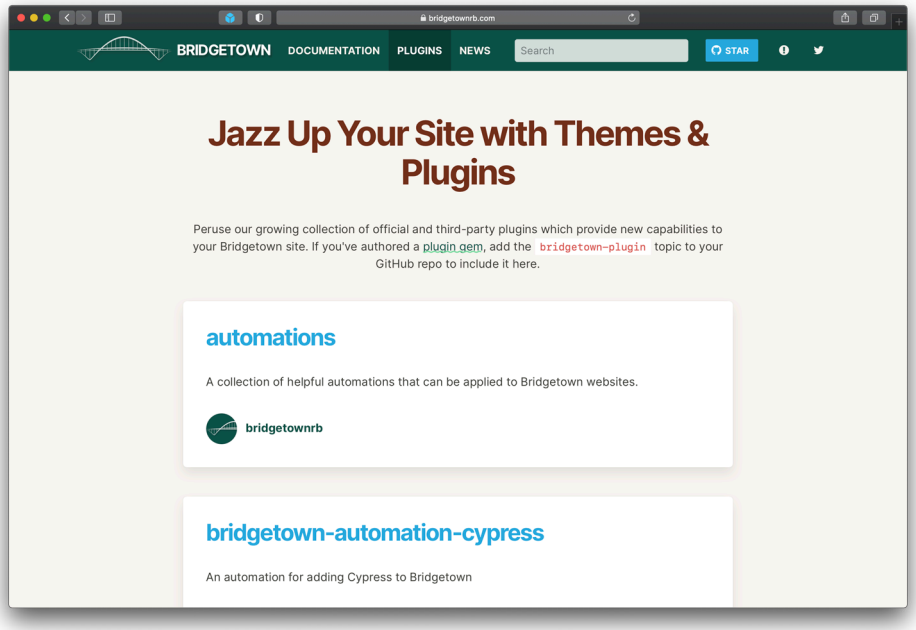
```
class LoadPostsFromAPI < SiteBuilder
  def build
    get "https://domain.com/posts.json" do |data|
      data.each do |post|
        doc "#{post[:slug]}.md" do
          front_matter post
          categories post[:taxonomy][:category].map { |category| category[:slug] }
          date Bridgetown::Utils.parse_date(post[:date])
          content post[:body]
        end
      end
    end
  end
end
```

File structure

```
├── frontend
│       ├── javascript
│       └── styles
├── node_modules
├── plugins
├── src
│       ├── _components
│       ├── _data
│       ├── _layouts
│       ├── _posts
│       ├── 404.html
│       └── index.md
├── bridgetown.config.yml
├── Gemfile
├── Gemfile.lock
├── start.js
├── sync.js
├── yarn.lock
├── webpack.config.js
└── package.json
```

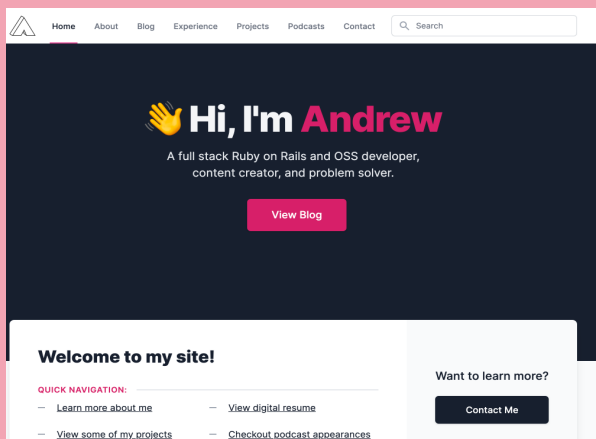
Ecosystem

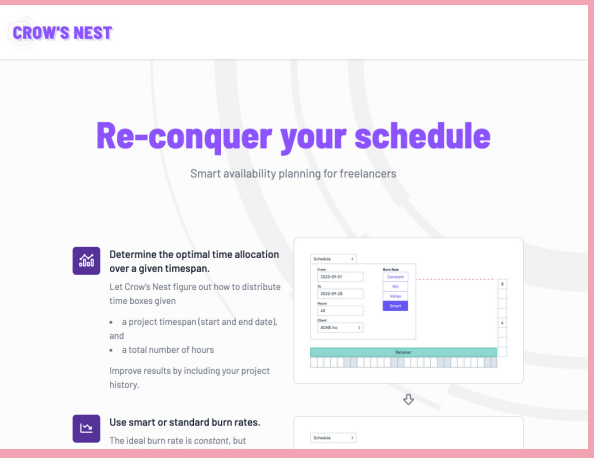
One of the core features of Bridgetown is that it can be extended by custom plugins. Starting from fetching data from external API, adding support for new markup or template language to taking full control over the build process.

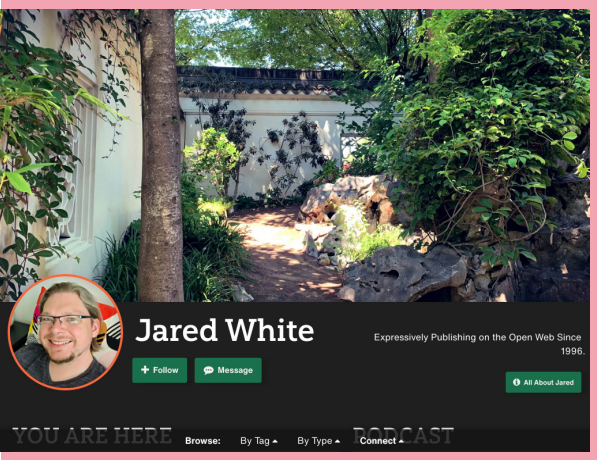


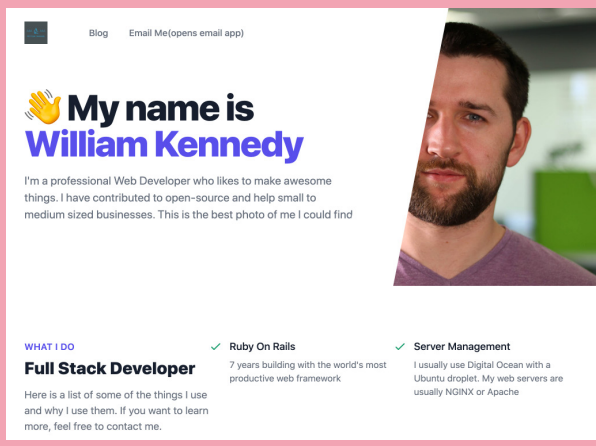
Bridgetown despite being very fresh, has a pretty lively Discord community. Everyone is welcome to ask questions or just chat.

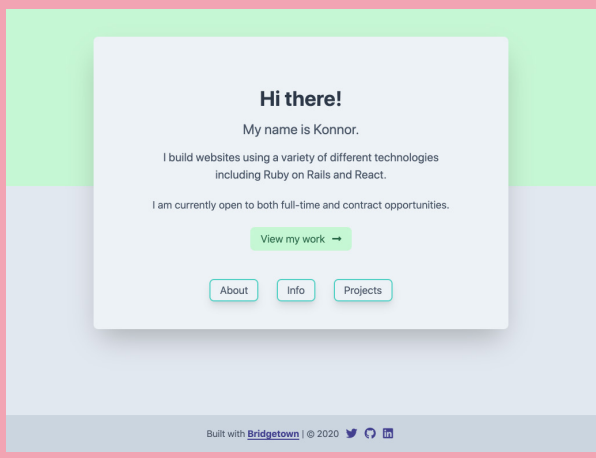
Showcase











How to get started

First, you need to install Ruby, that's the harder step if you don't have installed. You can find here instructions on how to make it done.

Next, you can install Bridgetown

```
gem install bridgetown -N
```

To create a new Bridgetown website, run:

```
bridgetown new mysite
```

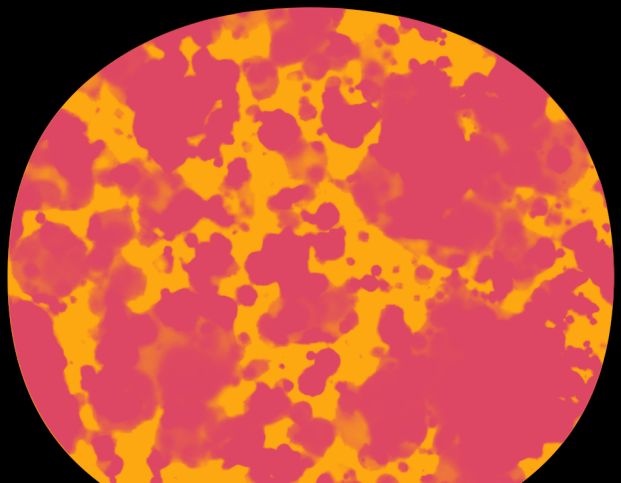
The build is a very simple process, you need to run only one command and the production bundle will be created in the "output" folder which can be later pushed to any static files hosting provider, eg: Netlify or Vercel.

Conclusion

Bridgetown is still way behind the competitors, but it's worth keeping an eye on. Especially, if you like Ruby. It's still in its early days but in the future, it can bring some fresh fruit.

FEATURES	USE CASES
👍 Easy setup for simple blog-aware websites.	✅ Blogs / content-based websites
👍 Encourages progressive enhancement.	✅ Personal website. It's a great way to test the Bridgetown SSG.
👍 Shallow learning curve to build a production-ready website.	✅ Jekyll website that needs an upgrade to modern JS toolset
👍 No need to know any additional JS framework, but can work with any JS framework.	
👍 Use easy-to-learn liquid template engine, but you can use different ones like ERB, HAML or Slim.	
👍 Comes with taxonomy pages, paginations and multilanguage setup out-of-the-box.	
👍 Webpack config out-of-the-box	

Final words



Final words

In a Nutshell

With static site generators, you get the benefits of using a traditional CMS, with the simplicity and performance of static HTML. They're reliable, scalable, can potentially save you time and money, and can handle high volumes of traffic well.

From our own experience, if you're seeking simplicity and a short learning curve - you should go with **Eleventy**. If you are building a big site, and you want to be able to change and add many new posts, choose **Hugo**. In any other case, I'd suggest you go with **Gatsby** or **Next.js**, **Gridsome** or **Nuxt** if you are into Vue.

Stuck with a project? Not sure which one of these static site generators is best for you? **Let's get in touch**!

We'll be more than happy to chat through your requirements and advise you on the best path forward.