**Engineer**

# Static site generator

a program that writes every page out in advance

## What it is

A static site generator is a build tool that takes a folder of source files, usually pages written in Markdown and a set of templates, and turns them into a finished website made only of files: HTML, stylesheets, scripts and images that a web server sends exactly as they are. The term is often shortened to SSG, and “static” describes how the result is served, the same bytes to everyone who asks for an address, not how a page behaves once it reaches a browser.

It sits opposite the dynamic approach of a content management system or a web application, where a program on the server assembles each page when it is requested, usually by querying a database and filling in templates. A generator runs the same kind of templates over the same kind of content, but once per change instead of once per visit, and leaves the result in a folder that an ordinary web server, a storage bucket or a content delivery network can serve without knowing how the pages were made.

Hugo is one such generator. Its documentation introduces it as “a static site generator written in Go, optimized for speed and designed for flexibility” that “renders a complete site in seconds, often less”. The usual loop is to write with its embedded web server running, which rebuilds on every change and reloads the open browser pages, then build the site and deploy it to a host:

```
hugo server     # build, serve, rebuild and reload on every save
hugo --minify   # write the finished site into public/
```

Use the concept in a decision

Put “What is a static site generator?” to work in a real decision. We can help relate the concept to the system you are building or buying. Interfaces that load quickly, work on a phone and behave the same in every browser your customers happen to use.

**Explore our services** Get in touch →

Test the assumptions behind the term

Use the explanation and linked work to frame questions for your project. We can examine the assumptions with you before a tool is chosen. Directed full-stack GIS map development — full-stack mapping, from database to browser

**Directed full-stack GIS map development** Get in touch →

## Why it matters

The choice is about when the work happens. A generator moves it from request time to build time, and almost everything that follows comes from that.

Sending a file is about the cheapest thing a web server does, so a generated site answers quickly and can run on a very small machine, and because every visitor to an address gets the same bytes, caching is simple. With no application code or database running behind the pages, there is far less to secure and patch. The output is a folder, so moving to another host means copying files, and the source lives in version control, where every change to every page can be reviewed and traced.

The costs come from the same place. Every change, a corrected typo included, means a fresh build and a deploy, so build speed matters more as the page count grows. Anything that depends on who is asking, such as search, comments, sign-in or a form that stores what it receives, needs code in the browser, a separate service, or a way to be settled at build time, because there is no program on the server to do it. Time stands still between builds: Hugo by default leaves out a page whose date lies in the future, so a page scheduled for Monday appears with the first build on or after Monday. Even redirects change shape. With no program to send an HTTP redirect, Hugo’s default for a moved page is a small HTML file at the old address that sends the browser on, an approach its documentation calls portable across all hosting providers.

## What it looks like in practice

This site is built with one. The credits page names Hugo as the static site generator and credits it with the templates, the taxonomies, every language and the feeds. Of what reaches a browser, it says the HTML is written out in full before anyone asks for it: every page is a file on disk, not a template assembled

per request. Even the look is settled at build time, since the light theme's wall is drawn while the page is built and the site is dressed afresh each time it is published. The quality gate runs locally, on the computer the site is edited on, without a hosted build service anywhere in the chain; it treats every diagnostic as an error, and in October 2026 the site built 2,268 pages. They are served from a single-core server with 512 MB of memory, where the web server and the three other services that answer for the site read the same folder of files, so one deploy updates all four.

The portfolio entry on building this site in Hugo documents the August 2026 implementation stage and gives the reasoning. Three languages, no separately billed application service and no visitors handed to third parties ruled out almost every content management system, because a backend that runs is something to secure, patch, pay for and explain on a privacy page. The site then had 108 template files, 53 of them partials, published every page in four representations from one content tree, and built that tree seven more times for focused subdomains by pointing the generator at a different content directory. The write-up records the trade-off of adding capability at build time rather than in the browser: beyond the recorded scoped JavaScript exception for a portfolio filter, interactive features had to be solved at build time or omitted, which is why a search index was costed and parked rather than shipped.

## Where it goes wrong

**A deleted page stays online.** A page removed from the source is still served at its old address, and search engines go on finding it. Hugo does not clear its output folder before a build: existing files are overwritten but never deleted, so `public/` can keep pages from earlier builds, drafts and expired pages among them, and anything that copies that folder to a server copies them too. Its `resources/` cache has a cousin of the problem: this site's design rules warn that it can serve a stale CSS bundle when only the source CSS has changed. Build into a clean folder, with Hugo's `--cleanDestinationDir` flag or a delete step of your own; this site's build removes both `public/` and `resources/` before it starts.

**A finished page never appears.** A post is written, committed and deployed, the build reports success, and the site has no page for it. By default Hugo leaves out content marked as a draft, content whose date or publish date lies in the future, and content whose expiry date has passed, and none of that counts as an error. A date typed a day ahead is enough. Check those four fields first, keep `--buildDrafts`, `--buildFuture` and `--buildExpired` for previews, and for a scheduled page arrange a build after its date.

**Every page claims to have changed at once.** The sitemap gives nearly every address the same last-modified time, which tells a crawler the whole site changed and says nothing about which pages did. A generator has to take each page's modification date from somewhere, and the file's own timestamp is the wrong source when a step rewrites files whose content stays the same. This site's configuration records that its sync step rewrites all 189 content files on every run, so 172 of the 176 sitemap addresses shared one timestamp. Take the date from version control instead: Hugo reads the date of a file's last Git revision when `enableGitInfo` is on, and a re-sync that produces identical bytes makes no commit, so a re-sync after a month of edits touched 8 of 186 files.

**Files nobody meant to publish go out.** Anyone who guesses the name can fetch a file that lists other files. A generator publishes whatever sits in the folders it reads, minus what an exclusion rule removes, and a rule that does not reach every folder leaves the rest exposed. This site's configuration records that its site-wide ignore list did not reach a module mount: the build published all six `.DS_Store` files the macOS Finder had left under icons/, images/ and webmanifest/, on every host, and each listed the names beside it, so the one in images/ named a folder the site withholds. A filter missing one space after its `!` once excluded every image in a mount instead. Exclude on every mount that publishes, then check the built output for files that do not belong; two of this site's checks fail a build that serves one.

## Further reading

- Hugo: Introduction — what Hugo is, what it is used for, and how its write, build and deploy loop runs.
- Hugo: Basic usage — building and serving a site, why `public/` keeps old files, and which content a build leaves out.
- Hugo: Configure front matter — how Hugo picks a page's dates, including the `:git` and `:fileModTime` tokens.
- Hugo: Configure modules — mounts, the `files` filter each one takes, and the rule that defining a mount removes the default one.
- Hugo: URL management — aliases, and the redirect files a static host serves in place of a server redirect.



Glossary

Choose for your actual constraints

Describe the workflow, the existing interfaces and the users who need the change. We can define a testable delivery and a handover that lets your team maintain it.

**See what it costs** Get in touch →

# Make the terminology practical

Ask how “What is a static site generator?” applies to your organisation. A concrete example from your current system is enough to start the discussion.

**Explore our services**

welcome@engineer.company +45 50 10 61 84 Copenhagen, Denmark

Take this page with you

Contact card

Carefully Crafted With Love by D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Copyright © 2025 – present · Engineer ApS · All rights reserved.

Printed from https://engineer.company/glossary/ssg/