



Static Site Generator or Headless CMS for a Marketing Website?

October 03, 2026

Table of Contents

Static Site Generator Overview

Headless CMS Overview

Feature-by-Feature Comparison

Performance and Hosting

Pricing and Value Analysis

User Experience: Authoring, Preview, and Review

Pros and Cons of Each

Static site generator

Headless CMS

Lock-In and Migration Risks

Use Case Recommendations

Quick Comparison Summary

Final Recommendation

Static Site Generator vs Headless CMS: Which Fits Your Marketing Website?

A static site generator (SSG) and a headless content management system (CMS) solve different parts of the marketing website problem. An SSG turns source content and templates into pages during a build. A headless CMS gives teams a content-editing and publishing environment, typically delivering content to a separately built website through an API. They can also be used together.

This comparison focuses on six decision criteria: authoring workflow, publishing independence, performance and hosting, preview and review, ongoing cost, and lock-in and migration risk. It considers the operating model rather than ranking specific vendors. The technical framing draws on the [Next.js documentation on static generation](#), the [MDN Web Docs performance guidance](#), [web.dev performance learning resources](#), and the [Jamstack site](#). Platform capabilities vary, so validate the details against the documentation for the tools you plan to use.



Both approaches can deliver a marketing website; they differ most in how teams create, approve, and publish content.

Static Site Generator Overview

A static site generator uses templates, content, and application code to produce pages ahead of a visitor's request. Content may live in files, a repository, or an external service. When it changes, a build process regenerates the affected site and publishes the output to a host or content delivery network (CDN). Static generation is a common fit for marketing pages, blogs, and documentation; see the [SSG overview in the Next.js documentation](#).

The key advantage is a relatively direct publishing path from build output to delivery. Pages can be served without running a full application server for every request. But a generator does not automatically provide a friendly editorial environment. If the content workflow relies on files and code review, making a small copy change may involve a developer, repository access, and a deployment.

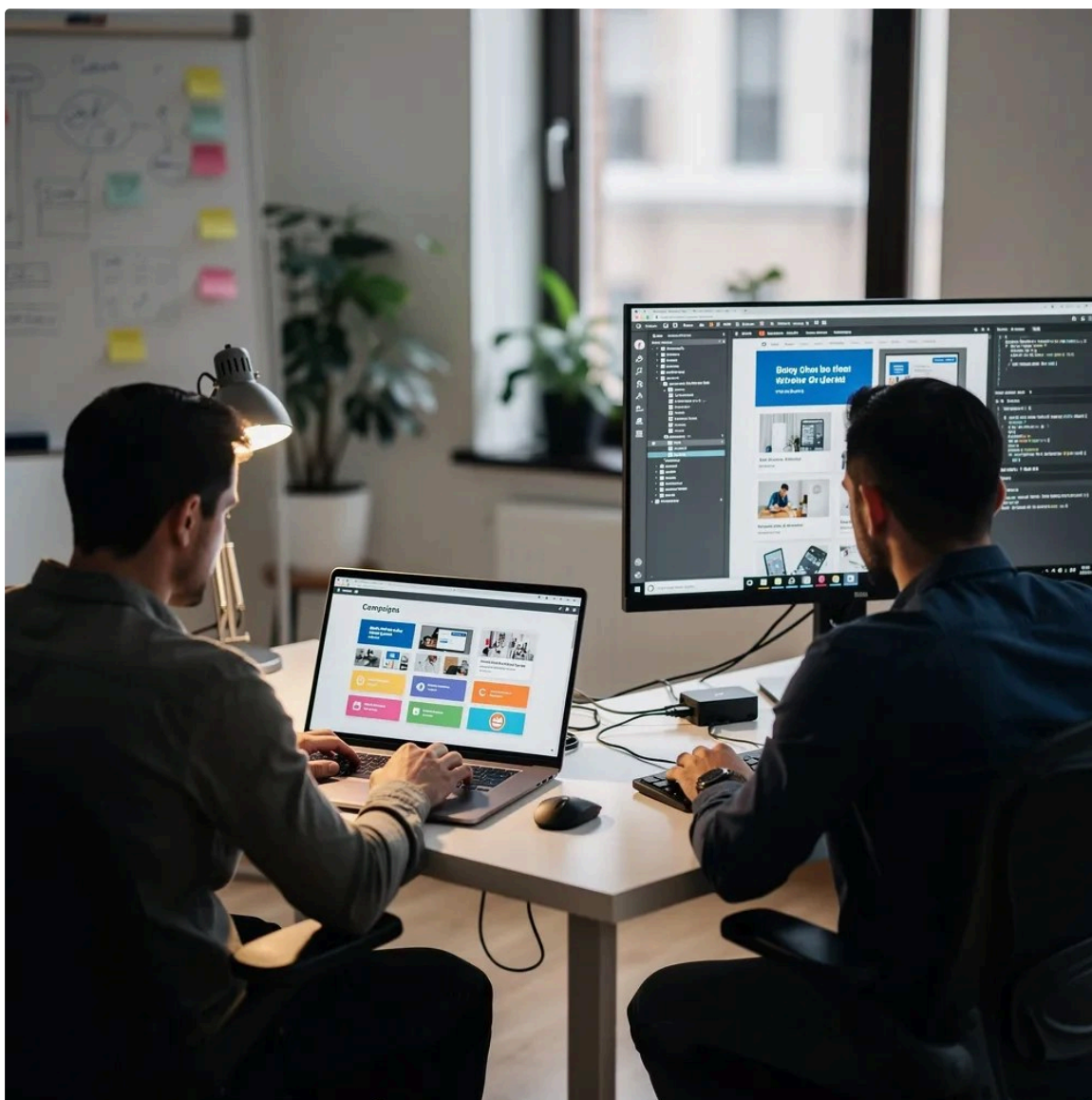
That trade-off can work well when a site changes infrequently, the team is comfortable with a code-based workflow, or engineering wants close control over templates and content structure. It becomes less attractive when marketers need to create and publish campaigns independently.

Headless CMS Overview

A headless CMS separates content management from the website's presentation layer. Editors work in a dedicated interface, while the website retrieves content through an API or a connected build process. Content can be structured into reusable fields and delivered to more than one channel, depending on the platform and implementation. The [Contentful developer documentation](#) describes one platform's APIs and integration options; it is an example, not a requirement to choose that product.

A CMS changes the content workflow, but it does not remove the need to build and operate the website. Developers still create templates, connect content models, handle integrations, and establish publishing rules. Some architectures rebuild the site when content changes; others fetch some or all content at request time. The delivery model affects performance, hosting, and preview behavior.

This approach tends to suit teams with regular publishing, multiple content contributors, approval requirements, or a need to reuse content. Its benefits depend on whether the editing interface, roles, and workflows fit the team. A CMS that adds complexity without giving editors useful independence can cost more than it saves.



A headless CMS can give nontechnical contributors a dedicated place to author and manage content.

Feature-by-Feature Comparison

Criterion	Static site generator	Headless CMS	Likely advantage
Authoring workflow	Often files, code review, or an added editor	Structured content in an editorial interface	CMS for frequent nontechnical editing
Publish without a developer	Depends on the setup; often limited by default	Possible when roles and publishing are configured	CMS
Delivery and hosting	Prebuilt pages can use simple static hosting	Depends on static builds, runtime APIs, or both	SSG for simpler delivery
Preview and review	May require preview builds or code review	Can support editorial previews and approvals	CMS when workflows are configured

Criterion	Static site generator	Headless CMS	Likely advantage
Ownership cost	Can reduce service costs but needs engineering time	Can reduce editorial friction but adds platform costs	Depends on labor and publishing volume
Migration risk	Content may be tied to custom code or file formats	Content models and APIs can create vendor dependency	No universal winner

Category winners: An SSG usually wins for a lean, developer-managed site with straightforward delivery. A headless CMS usually wins for an active content team that needs a reliable authoring and approval workflow. Neither automatically wins on security or compliance: assess access controls, audit capabilities, data handling, and operational responsibilities for the actual implementation.

Performance and Hosting

Static generation can make delivery straightforward: the build produces pages that a CDN or static host can serve directly. This reduces the need to render each page dynamically for each visit. It does not guarantee a fast site, though. Heavy images, large JavaScript bundles, third-party scripts, and slow client-side rendering can still harm the experience. The [MDN Web Docs performance resources](#) and [web.dev](#) offer guidance on measuring and improving these factors.

A headless CMS does not inherently make a site slower. If content is fetched during a build and pages are then served statically, delivery can resemble an SSG-only site. If pages depend on runtime API calls, performance also depends on the CMS endpoint, caching, network conditions, and application design. Preview functionality may use a different route from the public site, so test both.

The qualitative comparison below describes common tendencies, not benchmark results. Measure representative pages and user journeys before selecting an architecture.

Delivery pattern	Request-time work	Hosting complexity	Performance considerations
SSG with static hosting	Usually low for page delivery	Often low	Optimize assets, scripts, and build output
CMS-triggered static build	Usually low after publishing	Moderate: CMS plus build and hosting	Build duration and content freshness matter
Runtime CMS delivery	Higher when pages fetch content per request	Moderate to high	API latency, caching, and resilience matter



Source: noirlab.edu

Architecture, caching, and front-end quality-not the CMS label alone-determine real-world performance.

Pricing and Value Analysis

There is no dependable universal price comparison between an SSG and a headless CMS. A generator may be available without a separate license fee, but the total cost includes implementation, build infrastructure, hosting, monitoring, maintenance, and developer time. A low hosting bill is not a saving if routine campaign edits continually consume engineering capacity.

A headless CMS may add subscription or usage charges, as well as integration and administration work. In return, it can reduce the time editors spend waiting for code changes and give teams structured content, roles, and review processes. Pricing and feature limits differ by provider and plan; check current official pricing and documentation rather than relying on old estimates.

Cost area	SSG-led approach	Headless CMS approach
Platform fees	May be minimal, depending on build and hosting tools	May include recurring CMS fees or usage charges
Engineering	Higher if editors need developer help for content changes	Higher at setup; potentially less routine editing support
Operations	Build and deployment pipeline still needs ownership	CMS, integrations, access, and delivery need ownership

Cost area	Ssg-led approach	Headless CMS approach
Best value when	Content changes are limited and engineering capacity is available	Publishing volume and editor independence justify the platform

Estimate total cost of ownership over several years. Include staff time, migration, training, vendor support, and the cost of publishing delays. The cheaper option is the one that supports your required content workflow with less total effort-not necessarily the one with the smallest software bill.

User Experience: Authoring, Preview, and Review

For authors, the most visible difference is where work happens. An SSG-only workflow may ask contributors to edit files, open a change request, and wait for a build and deployment. That can provide a clear record of changes, but it may be unfamiliar or cumbersome for a marketing team. An integrated editor can improve the experience, but it is an additional component that needs configuration and support.

A headless CMS offers a dedicated authoring environment, but "no developer required" is not automatic. Editors can publish independently only if the CMS permissions, content models, templates, and deployment process support it. Changes to page structure, a new component, or an unusual integration will still need engineering.

Preview and review deserve a practical test before a decision. Ask editors to create a draft, preview it, request approval, make a correction, and publish it. Check whether the preview matches the live site, whether reviewers can comment or approve, and what happens when a build or API request fails. A strong workflow makes routine content safe to publish without pretending every change is a content-only task.

Pros and Cons of Each

STATIC SITE GENERATOR

- **Pros:** Direct delivery of prebuilt pages; flexible control over templates and code; can pair with simple hosting; no CMS subscription is necessarily required.
- **Cons:** A code-centered workflow may restrict marketers; previews and approvals need deliberate setup; builds and content structures require engineering ownership.
- **Winner when:** The site is small or stable, content changes are occasional, and developers already own publishing.

HEADLESS CMS

- **Pros:** A purpose-built editorial interface; structured and reusable content; the potential for independent publishing and formal review; content can support multiple delivery channels.

- **Cons:** Subscription and integration costs may apply; content models require planning; the website still needs development and maintenance; platform-specific features can complicate migration.
- **Winner when:** Many contributors publish regularly, approvals matter, or content needs to be reused across sites or channels.

Lock-In and Migration Risks

With an SSG, lock-in can build up in templates, plugins, framework-specific features, build pipelines, and the format in which content is stored. Plain, well-organized content files may be portable, but a bespoke implementation can still make a move expensive. Keep content distinct from presentation where practical, document build steps, and test whether another rendering system can consume the content.

A headless CMS can centralize content, but its schemas, rich-text formats, APIs, permissions, and editorial features may not map cleanly to another platform. Export access alone does not guarantee a painless migration. Review how content can be extracted, whether assets and relationships export intact, and which features depend on proprietary behavior.

For either choice, reduce risk with a documented content model, reliable backups, ownership of domain and deployment credentials, and a tested export or restore process. If the website supports regulated or sensitive operations, review security controls and compliance obligations during procurement; do not infer compliance from the architecture alone.

Use Case Recommendations

- **Choose an SSG** for a compact brochure site, a campaign landing page with limited updates, or a documentation-style site where developers control releases and simple delivery matters most.
- **Choose a headless CMS** for a content-intensive marketing website with regular campaigns, several editors, approval requirements, or shared content across digital channels.
- **Combine them** when editors need a CMS but the public site benefits from prebuilt pages. A CMS can hold content and trigger builds, letting the team retain an editorial interface and static delivery.
- **Reconsider both assumptions** if the site needs highly personalized or frequently changing information at request time. Evaluate the required rendering and data architecture rather than forcing every page into a static publishing model.

Before committing, prototype one representative page and one complete publishing journey. Include a draft, preview, approval, publish, and rollback or correction. That small test can expose workflow and integration costs that an architecture diagram misses.

Quick Comparison Summary

Your priority	Better starting point	Reason
Simple hosting and developer control	Static site generator	Prebuilt output can be served with a lean delivery setup
Editor autonomy and repeat publishing	Headless CMS	Provides a dedicated content workflow when configured well
Editorial usability plus static delivery	CMS with static generation	Separates content authoring from efficient public delivery
Lowest total cost	Depends on team and volume	Compare platform costs with engineering and editorial labor

Final Recommendation

There is no overall winner for every marketing website. The static site generator is the better choice when the site is relatively stable, developers can own content releases, and a straightforward delivery path is the priority. The headless CMS is the better choice when a marketing team publishes often and needs structured authoring, review, and greater independence from engineering.

For many organizations, the most balanced solution is a headless CMS connected to a static build. It gives editors a dedicated place to manage content while allowing the public site to serve prebuilt pages. That approach still carries CMS and integration costs, so it makes sense only when the editorial gains justify them.

Decide based on your real content workflow: who creates, reviews, and publishes; how quickly changes must appear; and who will maintain the system. If content is rare, start with an SSG. If publishing is a core marketing operation, start with a headless CMS and verify that editors can complete the full workflow without developer intervention.



The right choice follows publishing needs: simplicity for developer-led sites, or editorial independence for active content teams.