Why Astro?

Astro is the web framework for building **content-driven websites** like blogs, marketing, and e-commerce. Astro is best-known for pioneering a new frontend architecture to reduce JavaScript overhead and complexity compared to other frameworks. If you need a website that loads fast and has great SEO, then Astro is for you.

Features

Astro is an all-in-one web framework. It includes everything you need to create a website, built-in. There are also hundreds of different integrations and API hooks available to customize a project to your exact use case and needs.

Some highlights include:

- **Islands**: A component-based web architecture optimized for content-driven websites.
- **UI-agnostic**: Supports React, Preact, Svelte, Vue, Solid, Lit, HTMX, web components, and more.
- **Server-first**: Moves expensive rendering off of your visitors' devices.
- **Zero JS, by default**: Less client-side JavaScript to slow your site down.
- **Content collections**: Organize, validate, and provide TypeScript type-safety for your Markdown content.
- **Customizable**: Tailwind, MDX, and hundreds of integrations to choose from.

Design Principles

Here are five core design principles to help explain why we built Astro, the problems that it exists to solve, and why Astro may be the best choice for your project or team.

Astro is…

1. **Content-driven**: Astro was designed to showcase your content.
2. **Server-first**: Websites run faster when they render HTML on the server.
3. **Fast by default**: It should be impossible to build a slow website in Astro.
4. **Easy to use**: You don't need to be an expert to build something with Astro.
5. **Developer-focused**: You should have the resources you need to be successful.

Content-driven

Astro was designed for building content-rich websites. This includes marketing sites, publishing sites, documentation sites, blogs, portfolios, landing pages, community sites, and e-commerce sites. If you have content to show, it needs to reach your reader quickly.

By contrast, most modern web frameworks were designed for building *web applications*. These frameworks excel at building more complex, application-like experiences in the browser: logged-in admin dashboards, inboxes, social networks, todo lists, and even native-like applications like Figma and Ping. However with that complexity, they can struggle to provide great performance when delivering your content.

Astro's focus on content from its beginnings as a static site builder have allowed Astro to **sensibly scale up to performant, powerful, dynamic web applications** that still respect your content and your audience. Astro's unique focus on content lets Astro make tradeoffs and deliver unmatched performance features that wouldn't make sense for more application-focused web frameworks to implement.

Server-first

Astro leverages server-rendering over client-side rendering in the browser as much as possible. This is the same approach that traditional server-side frameworks -- PHP, WordPress, Laravel, Ruby on Rails, etc. -- have been using for decades. But you don't need to learn a second server-side language to unlock it. With Astro, everything is still just HTML, CSS, and JavaScript (or TypeScript, if you prefer).

This approach stands in contrast to other modern JavaScript web frameworks like Next.js, SvelteKit, Nuxt, Remix, and others. These frameworks were built for client-side rendering of your entire website and include server-side rendering mainly to address performance concerns. This approach has been dubbed the **Single-Page App (SPA)**, in contrast with Astro's **Multi-Page App (MPA)** approach.

The SPA model has its benefits. However, these come at the expense of additional complexity and performance tradeoffs. These tradeoffs harm page performance -- critical metrics like Time to Interactive (TTI) -- which doesn't make much sense for content-focused websites where first-load performance is essential.

Astro's server-first approach allows you to opt in to client-side rendering only if, and exactly as, necessary. You can choose to add UI framework components that run on the client. You can take advantage of Astro's view transitions router for finer control over select page transitions and animations. Astro's server-first rendering, either pre-rendered or on-demand, provides performant defaults that you can enhance and extend.

Fast by default

Good performance is always important, but it is *especially* critical for websites whose success depends on displaying your content. It has been well-proven that poor performance loses you engagement, conversions, and money. For example:

- Every 100ms faster → 1% more conversions (Mobify, earning +$380,000/yr)
- 50% faster → 12% more sales (AutoAnything)
- 20% faster → 10% more conversions (Furniture Village)
- 40% faster → 15% more sign-ups (Pinterest)
- 850ms faster → 7% more conversions (COOK)
- Every 1 second slower → 10% fewer users (BBC)

In many web frameworks, it is easy to build a website that looks great during development only to load painfully slow once deployed. JavaScript is often the culprit, since many phones and lower-powered devices rarely match the speed of a developer's laptop.

Astro's magic is in how it combines the two values explained above -- a content focus with a server-first architecture -- to make tradeoffs and deliver features that other frameworks cannot. The result is amazing web performance for every website, out of the box. Our goal: **It should be nearly impossible to build a slow website with Astro.**

An Astro website can load 40% faster with 90% less JavaScript than the same site built with the most popular React web framework. But don't take our word for it: watch Astro's performance leave Ryan Carniato (creator of Solid.js and Marko) speechless.

Easy to use

Astro's goal is to be accessible to every web developer. Astro was designed to feel familiar and approachable regardless of skill level or past experience with web development.

The `.astro` UI language is a superset of HTML: any valid HTML is valid Astro templating syntax! So, if you can write HTML, you can write Astro components! But, it also combines some of our favorite features borrowed from other component languages like JSX expressions (React) and CSS scoping by default (Svelte and Vue). This closeness to HTML also makes it easier to use progressive enhancement and common accessibility patterns without any overhead.

We then made sure that you could also use your favorite UI component languages that you already know, and even reuse components you might already have. React, Preact, Svelte, Vue, Solid, Lit, and others, including web components, are all supported for authoring UI components in an Astro project.

Astro was designed to be less complex than other UI frameworks and languages. One big reason for this is that Astro was designed to render on the server, not in the browser. That means that you don't need to worry about: hooks (React), stale closures (also React), refs (Vue), observables (Svelte), atoms, selectors, reactions, or derivations. There is no reactivity on the server, so all of that complexity melts away.

One of our favorite sayings is: **opt in to complexity.** We designed Astro to remove as much "required complexity" as possible from the developer experience, especially as you onboard for the first time. You can build a "Hello World" example website in Astro with just HTML and CSS. Then, when you need to build something more powerful, you can incrementally reach for new features and APIs as you go.

Developer-focused

We strongly believe that Astro is only a successful project if people love using it. Astro has everything you need to support you as you build with Astro.

Astro invests in developer tools like a great CLI experience from the moment you open your terminal, an official VS Code extension for syntax highlighting, TypeScript and Intellisense, and documentation actively maintained by hundreds of community contributors and available in 14 languages.

Our welcoming, respectful, inclusive community on Discord is ready to provide support, motivation, and encouragement. Open a `#support` thread to get help with your project. Visit our dedicated `#showcase` channel for sharing your Astro sites, blog posts, videos, and even work-in-progress for safe feedback and constructive criticism. Participate in regular live events such as our weekly community call, “Talking and Doc’ing,” and API/bug bashes.

As an open-source project, we welcome contributions of all types and sizes from community members of all experience levels. You are invited to join in roadmap discussions to shape the future of Astro, and we hope you’ll contribute fixes and features to the core codebase, compiler, docs, language tools, websites, and other projects.

Edit page

Previous
Tutorial: Build a Blog

Next
Astro Islands