Vite

Menu On this page

Why Vite

The Problems

Before ES modules were available in browsers, developers had no native mechanism for authoring JavaScript in a modularized fashion. This is why we are all familiar with the concept of "bundling": using tools that crawl, process and concatenate our source modules into files that can run in the browser.

Over time we have seen tools like webpack, Rollup and Parcel, which greatly improved the development experience for frontend developers.

However, as we build more and more ambitious applications, the amount of JavaScript we are dealing with is also increasing dramatically. It is not uncommon for large scale projects to contain thousands of modules. We are starting to hit a performance bottleneck for JavaScript based tooling: it can often take an unreasonably long wait (sometimes up to minutes!) to spin up a dev server, and even with Hot Module Replacement (HMR), file edits can take a couple of seconds to be reflected in the browser. The slow feedback loop can greatly affect developers' productivity and happiness.

Vite aims to address these issues by leveraging new advancements in the ecosystem: the availability of native ES modules in the browser, and the rise of JavaScript tools written in compile-to-native languages.

Slow Server Start

When cold-starting the dev server, a bundler-based build setup has to eagerly crawl and build your entire application before it can be served.

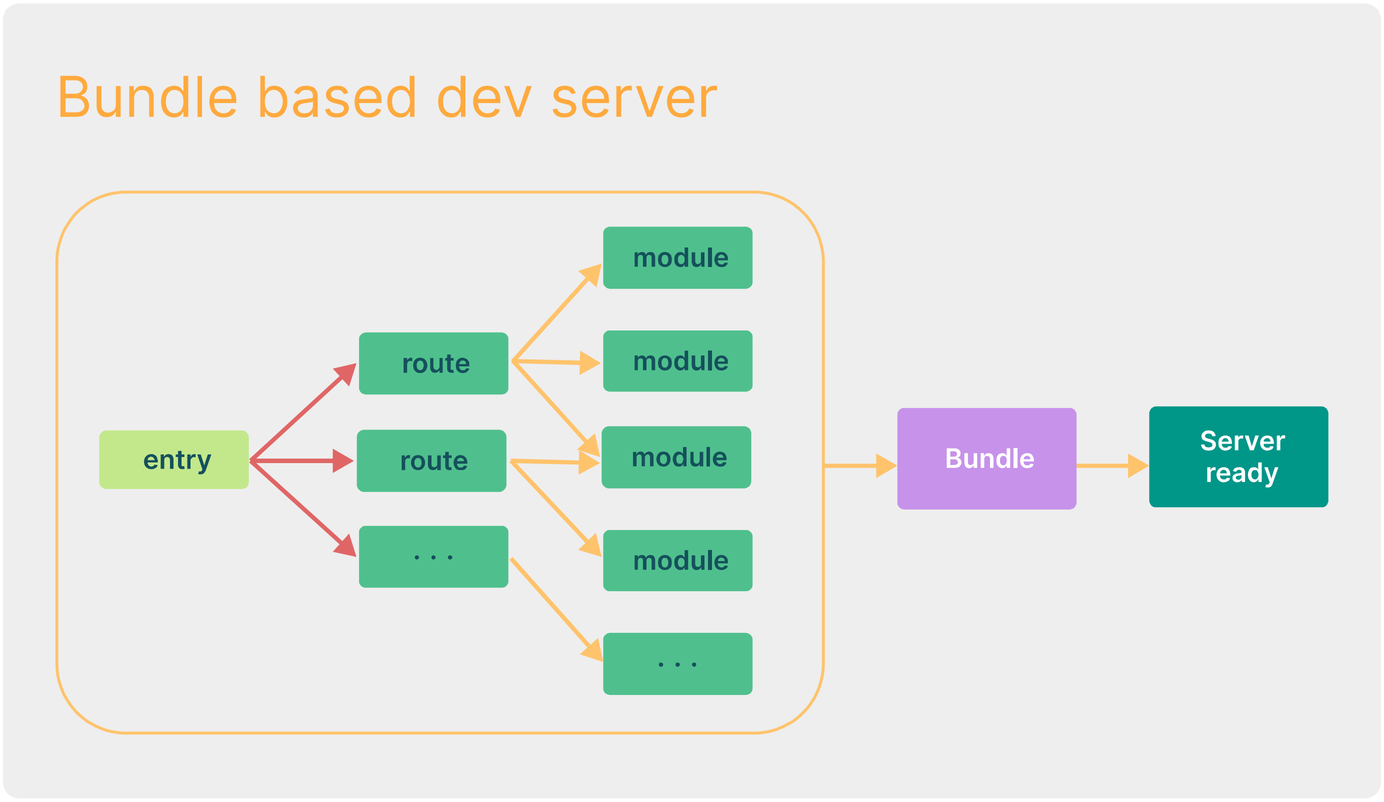
Vite improves the dev server start time by first dividing the modules in an application into two categories: **dependencies** and **source code**.

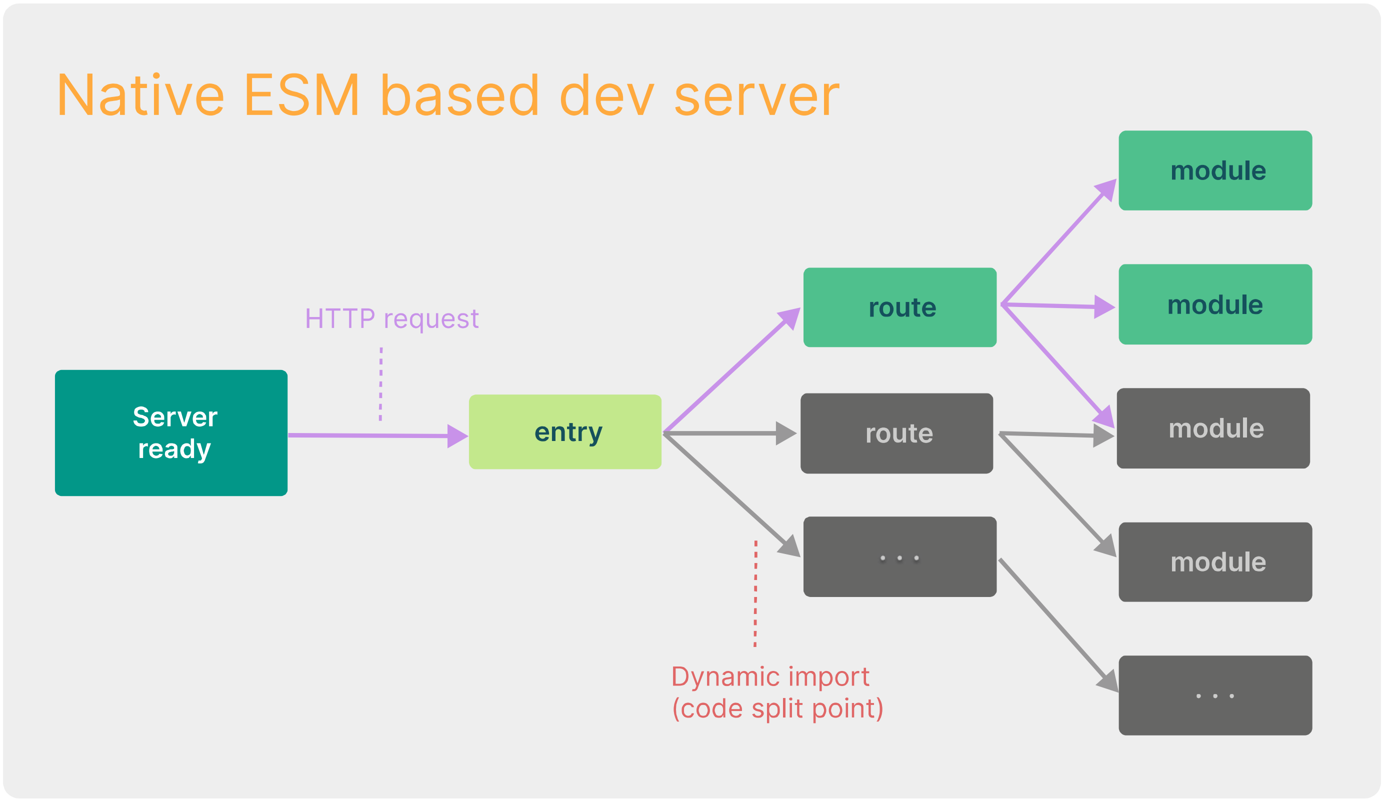
- **Dependencies** are mostly plain JavaScript that do not change often during development. Some large dependencies (e.g. component libraries with hundreds of modules) are also quite expensive to process. Dependencies may also be shipped in various module formats (e.g. ESM or CommonJS).

  Vite pre-bundles dependencies using esbuild. esbuild is written in Go and pre-bundles dependencies 10-100x faster than JavaScript-based bundlers.

- **Source code** often contains non-plain JavaScript that needs transforming (e.g. JSX, CSS or Vue/Svelte components), and will be edited very often. Also, not all source code needs to be loaded at the same time (e.g. with route-based code-splitting).

  Vite serves source code over native ESM. This is essentially letting the browser take over part of the job of a bundler: Vite only needs to transform and serve source code on demand, as the browser requests it. Code behind conditional dynamic imports is only processed if actually used on the current screen.





Slow Updates

When a file is edited in a bundler-based build setup, it is inefficient to rebuild the whole bundle for an obvious reason: the update speed will degrade linearly with the size of the app.

In some bundlers, the dev server runs the bundling in memory so that it only needs to invalidate part of its module graph when a file changes, but it still needs to re-construct the entire bundle and reload the web page. Reconstructing the bundle can be expensive, and reloading the page blows away the current state of the application. This is why some bundlers support Hot Module Replacement (HMR): allowing a module to "hot replace" itself without affecting the rest of the page. This greatly improves DX - however, in practice we've found that even HMR update speed deteriorates significantly as the size of the application grows.

In Vite, HMR is performed over native ESM. When a file is edited, Vite only needs to precisely invalidate the chain between the edited module and its closest HMR boundary (most of the time only the module itself), making HMR updates consistently fast regardless of the size of your application.

Vite also leverages HTTP headers to speed up full page reloads (again, let the browser do more work for us): source code module requests are made conditional via `304 Not Modified`, and dependency module requests are strongly cached via `Cache-Control: max-age=31536000,immutable` so they don't hit the server again once cached.

Once you experience how fast Vite is, we highly doubt you'd be willing to put up with bundled development again.

Why Bundle for Production

Even though native ESM is now widely supported, shipping unbundled ESM in production is still inefficient (even with HTTP/2) due to the additional network round trips caused by nested imports. To get the optimal loading performance in production, it is still better to bundle your code with tree-shaking, lazy-loading and common chunk splitting (for better caching).

Ensuring optimal output and behavioral consistency between the dev server and the production build isn't easy. This is why Vite ships with a pre-configured build command that bakes in many performance optimizations out of the box.

Why Not Bundle with esbuild?

Vite's current plugin API isn't compatible with using `esbuild` as a bundler. In spite of `esbuild` being faster, Vite's adoption of Rollup's flexible plugin API and infrastructure heavily contributed to its success in the ecosystem. For the time being, we believe that Rollup offers a better performance-vs-flexibility tradeoff.

Rollup has also been working on performance improvements, switching its parser to SWC in v4. And there is an ongoing effort to build a Rust-port of Rollup called Rolldown. Once Rolldown is ready, it could replace both Rollup and esbuild in Vite, improving build performance significantly and removing inconsistencies between development and build. You can watch Evan You's ViteConf 2023 keynote for more details.

How is Vite Different from X?

You can check out the Comparisons section for more details on how Vite differs from other similar tools.

Suggest changes to this page

Next page
Getting Started