Dependency Pre-Bundling

- The Why
- Automatic Dependency Discovery
- Monorepos and Linked Dependencies
- Customizing the Behavior
- Caching
  - File System Cache
  - Browser Cache

When you run `vite` for the first time, Vite prebundles your project dependencies before loading your site locally. It is done automatically and transparently by default.

The Why

This is Vite performing what we call "dependency pre-bundling". This process serves two purposes:

1. **CommonJS and UMD compatibility:** During development, Vite serves all code as native ESM. Therefore, Vite must convert dependencies that are shipped as CommonJS or UMD into ESM first.
   When converting CommonJS dependencies, Vite performs smart import analysis so that named imports to CommonJS modules will work as expected even if the exports are dynamically assigned (e.g. React):

   ```
   // works as expected
   import React, { useState } from 'react'
   ```

2. **Performance:** Vite converts ESM dependencies with many internal modules into a single module to improve subsequent page load performance.
   Some packages ship their ES modules builds as many separate files importing one another. For example, `lodash-es` has over 600 internal modules! When we do `import { debounce } from 'lodash-es'`, the browser fires off 600+ HTTP requests at the same time! Even though the server has no problem handling them, the large number of requests creates network congestion on the browser side, causing the page to load noticeably slower.

By pre-bundling `lodash-es` into a single module, we now only need one HTTP request instead!

::: tip NOTE Dependency pre-bundling only applies in development mode. :::

Automatic Dependency Discovery

If an existing cache is not found, Vite will crawl your source code and automatically discover dependency imports (i.e. "bare imports" that expect to be resolved from `node_modules`) and use these found imports as entry points for the pre-bundle. The pre-bundling is performed with Rolldown, so it's typically very fast.

After the server has already started, if a new dependency import is encountered that isn't already in the cache, Vite will re-run the dep bundling process and reload the page if needed.

Monorepos and Linked Dependencies

In a monorepo setup, a dependency may be a linked package from the same repo. Vite automatically detects dependencies that are not resolved from `node_modules` and treats the linked dep as source code. It will not attempt to bundle the linked dep, and will analyze the linked dep's dependency list instead.

However, this requires the linked dep to be exported as ESM. If not, you can add the dependency to `optimizeDeps.include` in your config.

```
import { defineConfig } from 'vite'
// ---cut---
export default defineConfig({
  optimizeDeps: {
    include: ['linked-dep'],
  },
})
```

When making changes to the linked dep, restart the dev server with the `--force` command line option for the changes to take effect.

Customizing the Behavior

The default dependency discovery heuristics may not always be desirable. In cases where you want to explicitly include/exclude dependencies from the list, use the `optimizeDeps` config options.

A typical use case for `optimizeDeps.include` or `optimizeDeps.exclude` is when you have an import that is not directly discoverable in the source code. For example, maybe the import is created as a result of a plugin transform. This means Vite won't be able to discover the import on the initial scan - it can only discover it after the file is requested by the browser and transformed. This will cause the server to immediately re-bundle after server start.

Both `include` and `exclude` can be used to deal with this. If the dependency is large (with many internal modules) or is CommonJS, then you should include it; If the dependency is small and is already valid ESM, you can exclude it and let the browser load it directly.

You can further customize Rolldown too with the `optimizeDeps.rolldownOptions` option. For example, adding a Rolldown plugin to handle special files in dependencies or changing the build `target`.

Caching

File System Cache

Vite caches the pre-bundled dependencies in `node_modules/.vite`. It determines whether it needs to re-run the pre-bundling step based on a few sources:

- Package manager lockfile content, e.g. `package-lock.json`, `yarn.lock`, `pnpm-lock.yaml`, `bun.lock`, `aube-lock.yaml` or `nub.lock`.
- Patches folder modification time.
- Relevant fields in your `vite.config.js`, if present.
- `NODE_ENV` value.

The pre-bundling step will only need to be re-run when one of the above has changed.

If for some reason you want to force Vite to re-bundle deps, you can either start the dev server with the `--force` command line option, or manually delete the `node_modules/.vite` cache directory.

Browser Cache

Resolved dependency requests are strongly cached with HTTP headers `max-age=31536000,immutable` to improve page reload performance during dev. Once cached, these requests will never hit the dev server again. They are auto invalidated by the appended version query if a different version is installed (as reflected in your package manager lockfile). If you want to debug your dependencies by making local edits, you can:

1. Temporarily disable cache via the Network tab of your browser devtools.
2. Restart Vite dev server with the `--force` flag to re-bundle the deps.
3. Reload the page.

□□□ #2
□ □□□ □□□ 2026-10-31 13:14:13 UTC
□ □□□ □□□ 2026-10-31 13:14:14 UTC