Server Options

- server.host
- server.allowedHosts
- server.port
- server.strictPort
- server.https
- server.open
- server.proxy
- server.cors
- server.headers
- server.hmr
- server.ws
- server.forwardConsole
- server.warmup
- server.watch
- server.middlewareMode
- server.fs.strict
- server.fs.allow
- server.fs.deny
- server.origin
- server.sourcemapIgnoreList

Unless noted, the options in this section are only applied to dev.

server.host

- **Type:** `string | boolean`
- **Default:** `'localhost'`

Specify which IP addresses the server should listen on. Set this to `0.0.0.0` or `true` to listen on all addresses, including LAN and public addresses.

This can be set via the CLI using `--host 0.0.0.0` or `--host`.

::: tip NOTE

There are cases when other servers might respond instead of Vite.

The first case is when `localhost` is used. Node.js's `dns.setDefaultResultOrder` changes how DNS-resolved addresses are ordered, and browsers may use a different resolved address than the one Vite is listening to. Vite prints the resolved address when it differs.

The second case is when wildcard hosts (e.g. `0.0.0.0`) are used. This is because servers listening on non-wildcard hosts take priority over those listening on wildcard hosts.

:::

::: tip Accessing the server on WSL2 from your LAN

When running Vite on WSL2, it is not sufficient to set `host: true` to access the server from your LAN. See the WSL document for more details.

:::

server.allowedHosts

- **Type:** `string[] | true`
- **Default:** `[]`

The hostnames that Vite is allowed to respond to. `localhost` and domains under `.localhost` and all IP addresses are allowed by default. When using HTTPS, this check is skipped.

If a string starts with `.`, it will allow that hostname without the `.` and all subdomains under the hostname. For example, `.example.com` will allow `example.com`, `foo.example.com`, and `foo.bar.example.com`. If set to `true`, the server is allowed to respond to requests for any hosts.

::: details What hosts are safe to be added?

Hosts that you have control over which IP addresses they resolve to are safe to add to the list of allowed hosts.

For example, if you own a domain `vite.dev`, you can add `vite.dev` and `.vite.dev` to the list. If you don't own that domain and you cannot trust the owner of that domain, you should not add it.

Especially, you should never add Top-Level Domains like `.com` to the list. This is because anyone can purchase a domain like `example.com` and control the IP address it resolves to.

:::

::: danger

Setting `server.allowedHosts` to `true` allows any website to send requests to your dev server through DNS rebinding attacks, allowing them to download your source code and content. We recommend always using an explicit list of allowed hosts. See GHSA-vg6x-rcgg-rjx6 for more details.

:::

::: details Configure via environment variable You can set the environment variable `__VITE_ADDITIONAL_SERVER_ALLOWED_HOSTS` to add additional allowed hosts. Use commas to separate multiple hosts (e.g., `host1.example.com,host2.example.com`). :::

server.port

- **Type:** `number`
- **Default:** `5173`

Specify server port. Note if the port is already being used, Vite will automatically try the next available port so this may not be the actual port the server ends up listening on.

server.strictPort

- **Type:** `boolean`

Set to `true` to exit if port is already in use, instead of automatically trying the next available port.

server.https

- **Type:** `https.ServerOptions`

Enable TLS + HTTP/2. The value is an options object passed to `https.createServer()`.

A valid certificate is needed. For a basic setup, you can add @vitejs/plugin-basic-ssl to the project plugins, which will automatically create and cache a self-signed certificate. But we recommend creating your own certificates.

server.open

- **Type:** `boolean | string`

Automatically open the app in the browser on server start. When the value is a string, it will be used as the URL's pathname. If you want to open the server in a specific browser you like, you can set the env `process.env.BROWSER` (e.g. `firefox`). You can also set `process.env.BROWSER_ARGS` to pass additional arguments (e.g. `--incognito`).

`BROWSER` and `BROWSER_ARGS` are also special environment variables you can set in the `.env` file to configure it. See the `open` package for more details.

Example:

```
export default defineConfig({
  server: {
    open: '/docs/index.html',
  },
})
```

server.proxy

- **Type:** `Record<string, string | ProxyOptions>`

Configure custom proxy rules for the dev server. Expects an object of `{ key: options }` pairs. Any requests whose request path starts with that key will be proxied to the specified target. If the key starts with `^`, it will be interpreted as a `RegExp`. The `configure` option can be used to access the proxy instance. If a request matches any of the configured proxy rules, the request won't be transformed by Vite.

Note that if you are using non-relative `base`, you must prefix each key with that `base`.

Extends `http-proxy-3`. Additional options are here.

In some cases, you might also want to configure the underlying dev server (e.g. to add custom middlewares to the internal connect app). In order to do that, you need to write your own plugin

and use configureServer function.

Example:

```
export default defineConfig({
  server: {
    proxy: {
      // string shorthand:
      // http://localhost:5173/foo
      //   -> http://localhost:4567/foo
      '/foo': 'http://localhost:4567',
      // with options:
      // http://localhost:5173/api/bar
      //   -> http://jsonplaceholder.typicode.com/bar
      '/api': {
        target: 'http://jsonplaceholder.typicode.com',
        changeOrigin: true,
        rewrite: (path) => path.replace(/^\/api/, ''),
      },
      // with RegExp:
      // http://localhost:5173/fallback/
      //   -> http://jsonplaceholder.typicode.com/
      '^/fallback/.*': {
        target: 'http://jsonplaceholder.typicode.com',
        changeOrigin: true,
        rewrite: (path) => path.replace(/^\/fallback/, ''),
      },
      // Using the proxy instance
      '/api': {
        target: 'http://jsonplaceholder.typicode.com',
        changeOrigin: true,
        configure: (proxy, options) => {
          // proxy will be an instance of 'http-proxy-3'
        },
      },
      // Proxying websockets or socket.io:
      // ws://localhost:5173/socket.io
      //   -> ws://localhost:5174/socket.io
      // Exercise caution using `rewriteWsOrigin` as it can leave the
      // proxying open to CSRF attacks.
```

```
      '/socket.io': {
        target: 'ws://localhost:5174',
        ws: true,
        rewriteWsOrigin: true,
      },
    },
  },
})
```

::: warning Origin check for WebSockets

Vite does not check the origin of WebSocket requests before proxying. The proxy target is expected to check the `Origin` header or other checks. Note that the `rewriteWsOrigin` option will rewrite the origin to the target origin and will cause the origin check to be bypassed.

:::

server.cors

- **Type:** `boolean | CorsOptions`
- **Default:** `{ origin: /^https?:\/\/(?:(?:[^:]+\.)?localhost|127\.0\.0\.1|\[::1\])(?::\d+)?$/ }` (allows localhost, `127.0.0.1` and `::1`)

Configure CORS for the dev server. Pass an options object to fine tune the behavior or `true` to allow any origin.

::: danger

Setting `server.cors` to `true` allows any website to send requests to your dev server and download your source code and content. We recommend always using an explicit list of allowed origins.

:::

server.headers

- **Type:** `OutgoingHttpHeaders`

Specify server response headers.

server.hmr

- **Type:** `boolean | { overlay?: boolean }`

Disable or configure HMR behavior.

Set `server.hmr.overlay` to `false` to disable the server error overlay.

::: warning Deprecated Options

The WebSocket-related options (`protocol`, `host`, `port`, `path`, `clientPort`, `timeout`, `server`) are deprecated. Use `server.ws` instead. These options are automatically synced, so existing configurations will continue to work.

:::

server.ws

- **Type:** `false | { protocol?: string, host?: string, port?: number, path?: string, timeout?: number, clientPort?: number, server?: Server }`

Configure WebSocket connection options. Set to `false` to disable the WebSocket connection entirely.

- `protocol` - WebSocket protocol (`ws` or `wss`)
- `host` - WebSocket server host
- `port` - WebSocket server port
- `path` - WebSocket path
- `clientPort` - Override the port on the client side, allowing you to serve the websocket on a different port than the client code looks for it on
- `timeout` - Connection timeout in milliseconds (default: 30000)
- `server` - Use a custom HTTP server for WebSocket connections

When `server.ws.server` is defined, Vite will process the WebSocket connection requests through the provided server. If not in middleware mode, Vite will attempt to process WebSocket connection requests through the existing server. This can be helpful when using self-signed certificates or when you want to expose Vite over a network on a single port.

```
export default defineConfig({
  server: {
    ws: {
      protocol: 'wss',
```

```
      host: 'localhost',
      port: 3001,
    },
  },
})
```

Check out `vite-setup-catalogue` for some examples.

::: tip NOTE

With the default configuration, reverse proxies in front of Vite are expected to support proxying WebSocket. If the Vite HMR client fails to connect WebSocket, the client will fall back to connecting the WebSocket directly to the Vite HMR server bypassing the reverse proxies:

```
Direct websocket connection fallback. Check out https://vite.dev/config/server-options.html#server-ws to remove the previous connection error.
```

The error that appears in the Browser when the fallback happens can be ignored. To avoid the error by directly bypassing reverse proxies, you could either:

- configure the reverse proxy to proxy WebSocket too
- set `server.strictPort = true` and set `server.ws.clientPort` to the same value with `server.port`
- set `server.ws.port` to a different value from `server.port`

:::

server.forwardConsole

- **Type:** `boolean | { unhandledErrors?: boolean, logLevels?: ('error' | 'warn' | 'info' | 'log' | 'debug')[] }`
- **Default:** auto (`true` when an AI coding agent is detected based on `std-env`, otherwise `false`)

Forward browser runtime events to the Vite server console during development.

- `true` enables forwarding unhandled errors and `console.error` / `console.warn` logs.
- `unhandledErrors` controls forwarding uncaught exceptions and unhandled promise rejections.
- `logLevels` controls which `console.*` calls are forwarded.

For example:

```
export default defineConfig({
  server: {
    forwardConsole: {
      unhandledErrors: true,
      logLevels: ['warn', 'error'],
    },
  },
})
```

When unhandled errors are forwarded, they are logged in the server terminal with enhanced formatting, for example:

```
1:18:38 AM [vite] (client) [Unhandled error] Error: this is test error
 > testError src/main.ts:20:8
     18|
     19| function testError() {
     20|   throw new Error('this is test error')
       |         ^
     21| }
     22|
 > HTMLButtonElement.<anonymous> src/main.ts:6:2
```

server.warmup

- **Type:** `{ clientFiles?: string[], ssrFiles?: string[] }`
- **Related:** Warm Up Frequently Used Files

Warm up files to transform and cache the results in advance. This improves the initial page load during server starts and prevents transform waterfalls.

`clientFiles` are files that are used in the client only, while `ssrFiles` are files that are used in SSR only. They accept an array of file paths or `tinyglobby` patterns relative to the `root`.

Make sure to only add files that are frequently used to not overload the Vite dev server on startup.

```
export default defineConfig({
  server: {
    warmup: {
```

```
      clientFiles: ['./src/components/*.vue', './src/utils/big-utils.js'],
      ssrFiles: ['./src/server/modules/*.js'],
    },
  },
})
```

server.watch

- **Type:** `object | null`

File system watcher options to pass on to chokidar.

When bundled-dev mode is enabled, Rolldown watch options (for example, `usePolling`, `pollInterval`, `useDebounce`, `debounceDuration`, `include`, `exclude`) are also accepted. The chokidar-only options are still used by the chokidar watcher, which keeps watching files outside the module graph, such as config file dependencies and env files.

The Vite server watcher watches the `root` and skips the `.git/`, `node_modules/`, `test-results/`, and Vite's `cacheDir` and `build.outDir` directories by default. When updating a watched file, Vite will apply HMR and update the page only if needed.

If set to `null`, no files will be watched. `server.watcher` will provide a compatible event emitter, but calling `add` or `unwatch` will have no effect.

::: warning Watching files in `node_modules`

It's currently not possible to watch files and packages in `node_modules`. For further progress and workarounds, you can follow issue #8619.

:::

::: warning Using Vite on Windows Subsystem for Linux (WSL) 2

When running Vite on WSL2, file system watching does not work when a file is edited by Windows applications (non-WSL2 process). This is due to a WSL2 limitation. This also applies to running on Docker with a WSL2 backend.

To fix it, you could either:

- **Recommended**: Use WSL2 applications to edit your files.
  - It is also recommended to move the project folder outside of a Windows filesystem. Accessing Windows filesystem from WSL2 is slow. Removing that overhead will

improve performance.

- Set `{ usePolling: true }`.
  - Note that `usePolling` leads to high CPU utilization.

:::

server.middlewareMode

- **Type:** `boolean | { server: http.Server }`
- **Default:** `false`

Create Vite server in middleware mode.

If proxy is setup for WebSocket, the `server` should be provided to bind the proxy correctly.

- **Related:** appType, SSR - Setting Up the Dev Server
- **Example:**

```
import express from 'express'
import { createServer as createViteServer } from 'vite'

async function createServer() {
  const app = express()

  // Create Vite server in middleware mode
  const vite = await createViteServer({
    server: { middlewareMode: true },
    // don't include Vite's default HTML handling middlewares
    appType: 'custom',
  })
  // Use vite's connect instance as middleware
  app.use(vite.middlewares)

  app.use('*', async (req, res) => {
    // Since `appType` is ``'custom'``, should serve response here.
    // Note: if `appType` is ``'spa'`` or ``'mpa'``, Vite includes middlewares
    // to handle HTML requests and 404s so user middlewares should be added
    // before Vite's middlewares to take effect instead
  })
```

```
}

createServer()
```

server.fs.strict

- **Type:** `boolean`
- **Default:** `true` (enabled by default since Vite 2.7)

Restrict serving files outside of workspace root.

server.fs.allow

- **Type:** `string[]`

Restrict files that could be served via `/@fs/`. When `server.fs.strict` is set to `true`, accessing files outside this directory list that aren't imported from an allowed file will result in a 403.

Both directories and files can be provided.

Vite will search for the root of the potential workspace and use it as default. A valid workspace met the following conditions, otherwise will fall back to the project root.

- contains `workspaces` field in `package.json`
- contains one of the following file
  - `lerna.json`
  - `pnpm-workspace.yaml`

Accepts a path to specify the custom workspace root. Could be a absolute path or a path relative to project root. For example:

```
export default defineConfig({
  server: {
    fs: {
      // Allow serving files from one level up to the project root
      allow: ['..'],
    },
  },
})
```

When `server.fs.allow` is specified, the auto workspace root detection will be disabled. To extend the original behavior, a utility `searchForWorkspaceRoot` is exposed:

```
import { defineConfig, searchForWorkspaceRoot } from 'vite'

export default defineConfig({
  server: {
    fs: {
      allow: [
        // search up for workspace root
        searchForWorkspaceRoot(process.cwd()),
        // your custom rules
        '/path/to/custom/allow_directory',
        '/path/to/custom/allow_file.demo',
      ],
    },
  },
})
```

server.fs.deny

- **Type:** `string[]`
- **Default:** `['.env', '.env.*', '*.{crt,pem,key,p12,pfx,cer,der}', '.npmrc', '.yarnrc.yml', '**/.git/**']`

Blocklist for sensitive files being restricted to be served by Vite dev server. This will have higher priority than `server.fs.allow`. picomatch patterns are supported.

::: tip NOTE

This blocklist does not apply to the public directory. All files in the public directory are served without any filtering, since they are copied directly to the output directory during build.

:::

::: tip NOTE

The deny filter is applied against the module id and the id with query parameters stripped. Since a plugin can read files from any files in its load hook (including resolving symlinks to denied paths), Vite cannot guarantee that a denied file is inaccessible through an alternative path. If you have an alternative path, include it in the deny list as well.

:::

server.origin

- **Type:** `string`

Defines the origin of the generated asset URLs during development.

```
export default defineConfig({
  server: {
    origin: 'http://127.0.0.1:8080',
  },
})
```

server.sourcemapIgnoreList

- **Type:** `false | (sourcePath: string, sourcemapPath: string) => boolean`
- **Default:** `(sourcePath) => /(?:^|[\\/])node_modules(?:[\\/]|$)/.test(sourcePath)`

Whether or not to ignore source files in the server sourcemap, used to populate the `x_google_ignoreList` source map extension.

`server.sourcemapIgnoreList` is the equivalent of `build.rolldownOptions.output.sourcemapIgnoreList` for the dev server. A difference between the two config options is that the Rolldown function is called with a relative path for `sourcePath` while `server.sourcemapIgnoreList` is called with an absolute path. During dev, most modules have the map and the source in the same folder, so the relative path for `sourcePath` is the file name itself. In these cases, absolute paths make it convenient to be used instead.

By default, it excludes all paths that contain `node_modules` as a path segment. You can pass `false` to disable this behavior, or, for full control, a function that takes the source path and sourcemap path and returns whether to ignore the source path.

```
export default defineConfig({
  server: {
    // This is the default value, and will add all files that have
    // node_modules as a path segment to the ignore list.
    sourcemapIgnoreList(sourcePath, sourcemapPath) {
      return /(?:^|[\\/])node_modules(?:[\\/]|$)/.test(sourcePath)
```

```
    },
  },
})
```

::: tip Note `server.sourcemapIgnoreList` and `build.rolldownOptions.output.sourcemapIgnoreList` need to be set independently. `server.sourcemapIgnoreList` is a server only config and doesn't get its default value from the defined Rolldown options. :::

□□□ #2
□ □□□ □□□ 2026-10-31 13:13:35 UTC
□ □□□ □□□ 2026-10-31 13:13:36 UTC