Migration from v7

- Default Browser Target Change
- Rolldown
  - Gradual Migration
  - Dependency Optimizer Now Uses Rolldown
  - JavaScript Transforms by Oxc
    - esbuild Fallbacks
  - JavaScript Minification by Oxc
  - CSS Minification by Lightning CSS
  - Consistent CommonJS Interop
  - Removed Module Resolution Using Format Sniffing
  - Require Calls For Externalized Modules
  - import.meta.url in UMD / IIFE
  - Removed build.rollupOptions.watch.chokidar option
  - Removed object form build.rollupOptions.output.manualChunks and deprecate function form one
  - build() Throws BundleError
  - Module Type Support and Auto Detection
  - Other Related Deprecations
- Removed Deprecated Features
- Advanced
- Migration from v6

If you are migrating from `rolldown-vite`, the technical preview release for Rolldown integrated Vite for v6 & v7, only the sections with in the title are applicable.

Default Browser Target Change

The default browser values of `build.target` and `'baseline-widely-available'` are updated to newer browser versions:

- Chrome 107 → 111
- Edge 107 → 111
- Firefox 104 → 114
- Safari 16.0 → 16.4

These browser versions align with Baseline Widely Available feature sets as of 2026-01-01. In other words, they were all released about two and a half years ago.

Rolldown

Vite 8 uses Rolldown and Oxc based tools instead of esbuild and Rollup.

Gradual Migration

The `rolldown-vite` package implements Vite 7 with Rolldown, without other Vite 8 changes. This can be used as an intermediate step to migrate to Vite 8. See the Rolldown Integration guide in the Vite 7 docs to switch to `rolldown-vite` from Vite 7.

For users migrating from `rolldown-vite` to Vite 8, you can undo the dependency changes in `package.json` and update to Vite 8:

```
{
  "devDependencies": {
    "vite": "npm:rolldown-vite@7.2.2" // [!code --]
    "vite": "^8.0.0" // [!code ++]
  }
}
```

Dependency Optimizer Now Uses Rolldown

Rolldown is now used for dependency optimization instead of esbuild. Vite still supports `optimizeDeps.esbuildOptions` for backward compatibility by converting it to `optimizeDeps.rolldownOptions` automatically. `optimizeDeps.esbuildOptions` is now deprecated and will be removed in the future and we encourage you to migrate to `optimizeDeps.rolldownOptions`.

The following options are converted automatically:

- `esbuildOptions.minify` -> `rolldownOptions.output.minify`
- `esbuildOptions.treeShaking` -> `rolldownOptions.treeshake`
- `esbuildOptions.define` -> `rolldownOptions.transform.define`
- `esbuildOptions.loader` -> `rolldownOptions.moduleTypes`
- `esbuildOptions.preserveSymlinks` -> `!rolldownOptions.resolve.symlinks`
- `esbuildOptions.resolveExtensions` -> `rolldownOptions.resolve.extensions`
- `esbuildOptions.mainFields` -> `rolldownOptions.resolve.mainFields`
- `esbuildOptions.conditions` -> `rolldownOptions.resolve.conditionNames`
- `esbuildOptions.keepNames` -> `rolldownOptions.output.keepNames`
- `esbuildOptions.platform` -> `rolldownOptions.platform`
- `esbuildOptions.plugins` -> `rolldownOptions.plugins` (partial support)

You can get the options set by the compatibility layer from the `configResolved` hook:

```
const plugin = {
  name: 'log-config',
  configResolved(config) {
    console.log('options', config.optimizeDeps.rolldownOptions)
  },
}
```

JavaScript Transforms by Oxc

Oxc is now used for JavaScript transformation instead of esbuild. Vite still supports the `esbuild` option for backward compatibility by converting it to `oxc` automatically. `esbuild` is now deprecated and will be removed in the future and we encourage you to migrate to `oxc`.

The following options are converted automatically:

- `esbuild.jsxInject` -> `oxc.jsxInject`
- `esbuild.include` -> `oxc.include`
- `esbuild.exclude` -> `oxc.exclude`
- `esbuild.jsx` -> `oxc.jsx`
  - `esbuild.jsx: 'preserve'` -> `oxc.jsx: 'preserve'`
  - `esbuild.jsx: 'automatic'` -> `oxc.jsx: { runtime: 'automatic' }`
    - `esbuild.jsxImportSource` -> `oxc.jsx.importSource`
  - `esbuild.jsx: 'transform'` -> `oxc.jsx: { runtime: 'classic' }`
    - `esbuild.jsxFactory` -> `oxc.jsx.pragma`
    - `esbuild.jsxFragment` -> `oxc.jsx.pragmaFrag`
  - `esbuild.jsxDev` -> `oxc.jsx.development`
  - `esbuild.jsxSideEffects` -> `oxc.jsx.pure`

- `esbuild.define` -> `oxc.define`
- `esbuild.banner` -> custom plugin using transform hook
- `esbuild.footer` -> custom plugin using transform hook

The `esbuild.supported` option is not supported by Oxc. If you need this option, please see oxc-project/oxc#15373.

You can get the options set by the compatibility layer from the `configResolved` hook:

```
const plugin = {
  name: 'log-config',
  configResolved(config) {
    console.log('options', config.oxc)
  },
}
```

Currently, the Oxc transformer does not support lowering native decorators as we are waiting for the specification to progress, see (oxc-project/oxc#9170).

:::: details Workaround for lowering native decorators

You can use Babel or SWC to lower native decorators for the time being.

Using Babel:

::: code-group

```
$ npm install -D @rolldown/plugin-babel @babel/plugin-proposal-decorators
```

```
$ yarn add -D @rolldown/plugin-babel @babel/plugin-proposal-decorators
```

```
$ pnpm add -D @rolldown/plugin-babel @babel/plugin-proposal-decorators
```

```
$ bun add -D @rolldown/plugin-babel @babel/plugin-proposal-decorators
```

```
$ deno add -D npm:@rolldown/plugin-babel npm:@babel/plugin-proposal-decorators
```

:::

```
import babel from '@rolldown/plugin-babel'
import { defineConfig } from 'vite'
```

```
function decoratorPreset(options: Record<string, unknown>) {
  return {
    preset: () => ({
      plugins: [['@babel/plugin-proposal-decorators', options]],
    }),
    rolldown: {
      // Only run this transform if the file contains a decorator.
      filter: {
        code: '@',
      },
    },
  }
}

export default defineConfig({
  plugins: [babel({ presets: [decoratorPreset({ version: '2023-11' })] })],
})
```

Using SWC:

::: code-group

```
$ npm install -D @rollup/plugin-swc @swc/core
```

```
$ yarn add -D @rollup/plugin-swc @swc/core
```

```
$ pnpm add -D @rollup/plugin-swc @swc/core
```

```
$ bun add -D @rollup/plugin-swc @swc/core
```

```
$ deno add -D npm:@rollup/plugin-swc npm:@swc/core
```

:::

```
import swc from '@rollup/plugin-swc'
import { defineConfig, withFilter } from 'vite'

export default defineConfig({
  // ...
  plugins: [
```

```
    withFilter(
      swc({
        swc: {
          jsc: {
            parser: { decorators: true, decoratorsBeforeExport: true },
            transform: { decoratorVersion: '2023-11' },
          },
        },
      }),
      // Only run this transform if the file contains a decorator.
      { transform: { code: '@' } },
    ),
  ],
})
```

::::

esbuild Fallbacks

`esbuild` is no longer directly used by Vite and is now an optional dependency. If you are using a plugin that uses the `transformWithEsbuild` function, you need to install `esbuild` as a `devDependency`. The `transformWithEsbuild` function is deprecated and will be removed in the future. We recommend migrating to the new `transformWithOxc` function instead.

JavaScript Minification by Oxc

The Oxc Minifier is now used for JavaScript minification instead of esbuild. You can use the deprecated `build.minify: 'esbuild'` option to switch back to esbuild. This configuration option will be removed in the future and you need install `esbuild` as a `devDependency` as Vite no longer relies on esbuild directly.

If you were using the `esbuild.minify*` options to control minification behavior, you can now use `build.rolldownOptions.output.minify` instead. If you were using the `esbuild.drop` option, you can now use `build.rolldownOptions.output.minify.compress.drop*` options.

Property mangling and its related options (`mangleProps`, `reserveProps`, `mangleQuoted`, `mangleCache`) are not supported by Oxc. If you need these options, please see oxc-project/oxc#15375.

esbuild and Oxc Minifier make slightly different assumptions about source code. In case you suspect the minifier is causing breakage in your code, you can compare these assumptions here:

- esbuild minify assumptions

- Oxc Minifier assumptions

Please report any issues you find related to minification in your JavaScript apps.

CSS Minification by Lightning CSS

Lightning CSS is now used for CSS minification by default. You can use the `build.cssMinify: 'esbuild'` option to switch back to esbuild. Note that you need to install `esbuild` as a `devDependency`.

Lightning CSS supports better syntax lowering and your CSS bundle size might increase slightly.

Consistent CommonJS Interop

The `default` import from a CommonJS (CJS) module is now handled in a consistent way.

If it matches one of the following conditions, the `default` import is the `module.exports` value of the importee CJS module. Otherwise, the `default` import is the `module.exports.default` value of the importee CJS module:

- The importer is `.mjs` or `.mts`.
- The closest `package.json` for the importer has a `type` field set to `module`.
- The `module.exports.__esModule` value of the importee CJS module is not set to `true`.

::: details The previous behavior

In development, if it matches one of the following conditions, the `default` import is the `module.exports` value of the importee CJS module. Otherwise, the `default` import is the `module.exports.default` value of the importee CJS module:

- *The importer is included in the dependency optimization* and `.mjs` or `.mts`.
- *The importer is included in the dependency optimization* and the closest `package.json` for the importer has a `type` field set to `module`.
- The `module.exports.__esModule` value of the importee CJS module is not set to `true`.

In build, the conditions were:

- The `module.exports.__esModule` value of the importee CJS module is not set to `true`.
- *`default` property of `module.exports` does not exist.*

(assuming `build.commonjsOptions.defaultIsModuleExports` is not changed from the default `'auto'`)

:::

See Rolldown's docs about this problem for more details: Ambiguous `default` import from CJS modules - Bundling CJS | Rolldown.

This change may break some existing code importing CJS modules. You can use the deprecated `legacy.inconsistentCjsInterop: true` option to temporarily restore the previous behavior. If you find a package that is affected by this change, please report it to the package author or send them a pull request. Make sure to link to the Rolldown documentation above so that the author can understand the context.

Removed Module Resolution Using Format Sniffing

When both `browser` and `module` fields are present in `package.json`, Vite used to resolve the field based on the content of the file and it used to pick the ESM file for browsers. This was introduced because some packages were using the `module` field to point to ESM files for Node.js and some other packages were using the `browser` field to point to UMD files for browsers. Given that the modern `exports` field solved this problem and is now adopted by many packages, Vite no longer uses this heuristic and always respects the order of the `resolve.mainFields` option. If you were relying on this behavior, you can use the `resolve.alias` option to map the field to the desired file or apply a patch with your package manager (e.g. `patch-package`, `pnpm patch`).

Require Calls For Externalized Modules

`require` calls for externalized modules are now preserved as `require` calls and not converted to `import` statements. This is to preserve the semantics of `require` calls. If you want to convert them to `import` statements, you can use Rolldown's built-in `esmExternalRequirePlugin`, which is re-exported from `vite`.

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

export default defineConfig({
  // ...
  plugins: [
    esmExternalRequirePlugin({
      external: ['react', 'vue', /^node:/],
    }),
  ],
})
```

See Rolldown's docs for more details: `require` external modules - Bundling CJS | Rolldown.

`import.meta.url` in UMD / IIFE

`import.meta.url` is no longer polyfilled in UMD / IIFE output formats. It will be replaced with `undefined` by default. If you prefer the previous behavior, you can use the `define` option with `build.rolldownOptions.output.intro` option. See Rolldown's docs for more details: Well-known `import.meta` properties - Non ESM Output Formats | Rolldown.

Removed

`build.rollupOptions.watch.chokidar` option

The `build.rollupOptions.watch.chokidar` option was removed. Please migrate to the `build.rolldownOptions.watch.watcher` option.

Removed object form `build.rollupOptions.output.manualChunks` and deprecate function form one

The object form `output.manualChunks` option is not supported anymore. The function form `output.manualChunks` is deprecated. Rolldown has the more flexible `codeSplitting` option. See Rolldown's docs for more details about `codeSplitting`: Manual Code Splitting - Rolldown.

`build()` Throws `BundleError`

This change only affects JS API users.

`build()` now throws a `BundleError` instead of the raw error thrown in the plugin. `BundleError` is typed as `Error & { errors?: RolldownError[] }` and it wraps the individual errors in an `errors` array. If you need the individual errors, you need to access `.errors`:

```
try {
  await build()
} catch (e) {
```

```
  if (e.errors) {
    for (const error of e.errors) {
      console.log(error.code) // error code
    }
  }
}
```

Module Type Support and Auto Detection

This change only affects plugin authors.

Rolldown has experimental support for Module types, similar to esbuild's `loader` option. Due to this, Rolldown automatically sets a module type based on the extension of the resolved id. If you are converting content from other module types to JavaScript in `load` or `transform` hooks, you may need to add `moduleType: 'js'` to the returned value:

```
import fs from 'node:fs'

const plugin = {
  name: 'txt-loader',
  load: {
    filter: { id: /\.txt$/ },
    handler(id) {
      const content = fs.readFileSync(id, 'utf-8')
      return {
        code: `export default ${JSON.stringify(content)}`,
        moduleType: 'js', // [!code ++]
      }
    },
  },
}
```

Other Related Deprecations

The following options are deprecated and will be removed in the future:

- `build.rollupOptions`: renamed to `build.rolldownOptions`
- `worker.rollupOptions`: renamed to `worker.rolldownOptions`
- `build.commonjsOptions`: it is now no-op

- `build.dynamicImportVarsOptions.warnOnError`: it is now no-op
- `resolve.alias[].customResolver`: Use a custom plugin with `resolveId` hook and `enforce: 'pre'` instead

Removed Deprecated Features

- Passing a URL to `import.meta.hot.accept` is no longer supported. Please pass an id instead. (#21382)

Advanced

These breaking changes are expected to only affect a minority of use cases:

- Extglobs are not supported yet (rolldown-vite#365)
- TypeScript legacy namespace is only supported partially. See Oxc Transformer's related documentation for more details.
- `define` does not share reference for objects: When you pass an object as a value to `define`, each variable will have a separate copy of the object. See Oxc Transformer's related documentation for more details.
- `bundle` object changes (`bundle` is an object passed in `generateBundle` / `writeBundle` hooks, returned by `build` function):
  - Assigning to `bundle[foo]` is not supported. This is discouraged by Rollup as well. Please use `this.emitFile()` instead.
  - the reference is not shared across the hooks (rolldown-vite#410)
  - `structuredClone(bundle)` errors with `DataCloneError: #<Object> could not be cloned`. This is not supported anymore. Please clone it with `structuredClone({ ...bundle })`. (rolldown-vite#128)
- All parallel hooks in Rollup work as sequential hooks. See Rolldown's documentation for more details.
- `"use strict";` is not injected sometimes. See Rolldown's documentation for more details.
- Transforming to ES5 and below with plugin-legacy is not supported (rolldown-vite#452)
- Passing the same browser with multiple versions of it to `build.target` option now errors: esbuild selects the latest version of it, which was probably not what you intended.
- Missing support by Rolldown: The following features are not supported by Rolldown and are no longer supported by Vite.
  - `build.rollupOptions.output.format: 'system'` (rolldown#2387)

    - `build.rollupOptions.output.format: 'amd'` (rolldown#2528)
    - `shouldTransformCachedModule` hook (rolldown#4389)
    - `resolveImportMeta` hook (rolldown#1010)
    - `renderDynamicImport` hook (rolldown#4532)
    - `resolveFileUrl` hook
- `parseAst` / `parseAstAsync` functions are now deprecated in favor of `parseSync` / `parse` functions which have more features.
- comments are removed before the `renderChunk` hook instead of after the `renderChunk` hook
- comments other than the ones listed here are moved, while Rollup only removes comments if the adjacent code is removed

Migration from v6

Check the Migration from v6 Guide in the Vite v7 docs first to see the needed changes to port your app to Vite 7, and then proceed with the changes on this page.

□□□ #2
□ □□□ □□□ 2026-10-31 13:14:28 UTC
□ □□□ □□□ 2026-10-31 13:14:29 UTC