Modules

Module Compatibility

Nuxt 3 has a basic backward compatibility layer for Nuxt 2 modules using `@nuxt/kit` auto wrappers. But there are usually steps to follow to make modules compatible with Nuxt 3 and sometimes, using Nuxt Bridge is required for cross-version compatibility.

We have prepared a Dedicated Guide for authoring Nuxt 3 ready modules using `@nuxt/kit`. Currently best migration path is to follow it and rewrite your modules. Rest of this guide includes preparation steps if you prefer to avoid a full rewrite yet making modules compatible with Nuxt 3.

You can check for a list of Nuxt 3 ready modules from Nuxt Modules.

Plugin Compatibility

Nuxt 3 plugins are **not** fully backward compatible with Nuxt 2.

Vue Compatibility

Plugins or components using the Composition API need exclusive Vue 2 or Vue 3 support.

By using vue-demi they should be compatible with both Nuxt 2 and 3.

Module Migration

When Nuxt 3 users add your module, you will not have access to the module container (`this.*`) so you will need to use utilities from `@nuxt/kit` to access the container functionality.

Test with `@nuxt/bridge`

Migrating to `@nuxt/bridge` is the first and most important step for supporting Nuxt 3.

If you have a fixture or example in your module, add `@nuxt/bridge` package to its config (see example)

Migrate from CommonJS to ESM

Nuxt 3 natively supports TypeScript and ECMAScript Modules. Please check Native ES Modules for more info and upgrading.

Ensure Plugins Default Export

If you inject a Nuxt plugin that does not have `export default` (such as global Vue plugins), ensure you add `export default () => { }` to the end of it.

Before After

Before

```
// ~/plugins/vuelidate.js
import Vue from 'vue'
import Vuelidate from 'vuelidate'

Vue.use(Vuelidate)
```

Avoid Runtime Modules

With Nuxt 3, Nuxt is now a build-time-only dependency, which means that modules shouldn't attempt to hook into the Nuxt runtime.

Your module should work even if it's only added to `buildModules` (instead of `modules`). For example:

- Avoid updating `process.env` within a Nuxt module and reading by a Nuxt plugin; use `runtimeConfig` instead.
- (*) Avoid depending on runtime hooks like `vue-renderer:*` for production
- (*) Avoid adding `serverMiddleware` by importing them inside the module. Instead, add them by referencing a file path so that they are independent of the module's context

(*) Unless it is for `nuxt dev` purpose only and guarded with `if (nuxt.options.dev) { }`.

🔎 Continue reading about Nuxt 3 modules in the Modules guide.

Use TypeScript (Optional)

While it is not essential, most of the Nuxt ecosystem is shifting to use TypeScript, so it is highly recommended to consider migration.

💡 You can start migration by renaming `.js` files, to `.ts`. TypeScript is designed to be progressive!

💡 You can use TypeScript syntax for Nuxt 2 and 3 modules and plugins without any extra dependencies.

Edit on Github

Enterprise Design Kit NuxtLabs

Nuxt Studio