# Authoring Nuxt Layers

Nuxt layers are a powerful feature that you can use to share and reuse partial Nuxt applications within a monorepo, or from a git repository or npm package. The layers structure is almost identical to a standard Nuxt application, which makes them easy to author and maintain. (Read More)

A minimal Nuxt layer directory should contain a `nuxt.config.ts` file to indicate it is a layer.

base/nuxt.config.ts

```ts
export default defineNuxtConfig({})
```

Additionally, certain other files in the layer directory will be auto-scanned and used by Nuxt for the project extending this layer.

- `components/*` - Extend the default components
- `composables/*` - Extend the default composables
- `pages/*` - Extend the default pages
- `server/*` - Extend the default server endpoints & middleware
- `nuxt.config.ts` - Extend the default nuxt config
- `app.config.ts` - Extend the default app config

## Basic Example

`nuxt.config.ts` `app.vue` `base/nuxt.config.ts` `base/components/BaseComponent.vue`

nuxt.config.ts

```ts
export default defineNuxtConfig({
  extends: [
    './base'
  ]
})
```

If you're interested in deepening your understanding about layers, consider examining [a fully fleshed out `nuxt.config.ts` file on the Docus platform](https://github.com/nuxt-themes/docus/blob/main/nuxt.config.ts).

# Starter Template

To get started you can initialize a layer with the nuxt/starter/layer template. This will create a basic structure you can build upon. Execute this command within the terminal to get started:

```
> npx nuxi init --template layer nuxt-layer
```

Follow up on the README instructions for the next steps.

Check nuxt-themes/starter for a more opinionated starter for authoring Nuxt themes. It can be initialized with:

```
> npx nuxi init --template gh:nuxt-themes/starter my-theme
```

# Publishing Layers

You can publish and share layers by either using a remote source or an npm package.

## Git Repository

You can use a git repository to share your Nuxt layer. Some examples:

```
nuxt.config.ts
export default defineNuxtConfig({
  extends: [
    'github:username/repoName',        // GitHub Remote Source
    'github:username/repoName/base',   // GitHub Remote Source within /base directory
    'github:username/repoName#dev',    // GitHub Remote Source from dev branch
    'github:username/repoName#v1.0.0', // GitHub Remote Source from v1.0.0 tag
    'gitlab:username/repoName',        // GitLab Remote Source example
    'bitbucket:username/repoName',     // Bitbucket Remote Source example
  ]
})
```

> If you want to extend a private remote source, you need to add the environment variable `GIGET_AUTH=<token>` to provide a token.

> Currently, with git remote sources, if a layer has npm dependencies, you will need to manually install them in the target project. We are working on this to auto-install layer dependencies with git sources.

## npm Package

You can publish Nuxt layers as an npm package that contains the files and dependencies you want to extend. This allows you to share your config with others, use it in multiple projects or use it privately.

To extend from an npm package, you need to make sure that the module is published to npm and installed in the user's project as a devDependency. Then you can use the module name to extend the current nuxt config:

nuxt.config.ts

```ts
export default defineNuxtConfig({
  extends: [
    // Node Module with scope
    '@scope/moduleName',
    // or just the module name
    'moduleName'
  ]
})
```

To publish a layer directory as an npm package, you want to make sure that the `package.json` has the correct properties filled out. This will make sure that the files are included when the package is published.

package.json

```json
{
  "name": "my-theme",
  "version": "1.0.0",
  "type": "module",
  "main": "./nuxt.config.ts",
  "dependencies": {},
  "devDependencies": {
    "nuxt": "^3.0.0"
  }
}
```

Make sure any dependency imported in the layer is **explicitly added** to the `dependencies`. The `nuxt` dependency, and anything only used for testing the layer before publishing, should remain in the `devDependencies` field.

Now you can proceed to publish the module to npm, either publicly or privately.

When publishing the layer as a private npm package, you need to make sure you log in, to authenticate with npm to download the node module.

# Tips

## Relative Paths and Aliases

When importing using aliases (such as `~/` and `@/`) in a layer components and composables, note that aliases are resolved relative to the user's project paths. As a workaround, you can **use relative paths** to import them. We are working on a better solution for named layer aliases.

Also when using relative paths in `nuxt.config` file of a layer, (with exception of nested `extends`) they are resolved relative to user's project instead of the layer. As a workaround, use full resolved paths in `nuxt.config`:

nuxt.config.ts

```ts
import { fileURLToPath } from 'url'
import { dirname, join } from 'path'

const currentDir = dirname(fileURLToPath(import.meta.url))

export default defineNuxtConfig({
  css: [
    join(currentDir, './assets/main.css')
  ]
})
```

# Multi-Layer Support for Nuxt Modules

You can use the internal array `nuxt.options._layers` to support custom multi-layer handling for your modules.

Example:

modules/my-module.ts

```ts
export default defineNuxtModule({
  setup(_options, nuxt) {
    for (const layer of nuxt.options._layers) {
      // You can check for a custom directory existence to extend for each layer
      console.log('Custom extension for', layer.cwd, layer.config)
    }
  }
})
```

**Notes:**

- Earlier items in the `_layers` array have higher priority and override later ones
- The user's project is the first item in the `_layers` array

# Going Deeper

Configuration loading and extends support is handled by unjs/c12, merged using unjs/defu and remote git sources are supported using unjs/giget. Check the docs and source code to learn more.

We are working to bring more improvements for layers support. Please refer to nuxt/nuxt#13367.