Configuration

By default, Nuxt is configured to cover most use cases. The `nuxt.config.ts` file can override or extend this default configuration.

Nuxt Configuration

The `nuxt.config.ts` file is located at the root of a Nuxt project and can override or extend the application's behavior.

A minimal configuration file exports the `defineNuxtConfig` function containing an object with your configuration. The `defineNuxtConfig` helper is globally available without import.

nuxt.config.ts
```ts
export default defineNuxtConfig({
  // My Nuxt config
})
```

This file will often be mentioned in the documentation, for example to add custom scripts, register modules or change rendering modes.

Every configuration option is described in the Configuration Reference.

You don't have to use TypeScript to build an application with Nuxt. However, it is strongly recommended to use the `.ts` extension for the `nuxt.config` file. This way you can benefit from hints in your IDE to avoid typos and mistakes while editing your configuration.

Environment overrides

You can configure fully typed, per-environment overrides in your nuxt.config

```
nuxt.config.ts
export default defineNuxtConfig({
  $production: {
    routeRules: {
      '/**': { isr: true }
    }
  },
  $development: {
    //
  }
})
```

If you're authoring layers, you can also use the `$meta` key to provide metadata that you or the consumers of your layer might use.

Environment Variables and Private Tokens

The `runtimeConfig` API exposes values like environment variables to the rest of your application. By default, these keys are only available server-side. The keys within `runtimeConfig.public` are also available client-side.

Those values should be defined in `nuxt.config` and can be overridden using environment variables.

nuxt.config.ts   .env

```
nuxt.config.ts
export default defineNuxtConfig({
  runtimeConfig: {
    // The private keys which are only available server-side
    apiSecret: '123',
    // Keys within public are also exposed client-side
    public: {
      apiBase: '/api'
    }
  }
})
```

These variables are exposed to the rest of your application using the `useRuntimeConfig` composable.

```
pages/index.vue
<script setup lang="ts">
```

```
const runtimeConfig = useRuntimeConfig()
</script>
```

👉 Read more in Docs > Guide > Going Further > Runtime Config.

App Configuration

The `app.config.ts` file, located in the source directory (by default the root of the project), is used to expose public variables that can be determined at build time. Contrary to the `runtimeConfig` option, these can not be overridden using environment variables.

A minimal configuration file exports the `defineAppConfig` function containing an object with your configuration. The `defineAppConfig` helper is globally available without import.

app.config.ts

```
export default defineAppConfig({
  title: 'Hello Nuxt',
  theme: {
    dark: true,
    colors: {
      primary: '#ff0000'
    }
  }
})
```

These variables are exposed to the rest of your application using the `useAppConfig` composable.

pages/index.vue

```
<script setup lang="ts">
const appConfig = useAppConfig()
</script>
```

👉 Read more in Docs > Guide > Directory Structure > App Config.

`runtimeConfig` VS `app.config`

As stated above, `runtimeConfig` and `app.config` are both used to expose variables to the rest of your application. To determine whether you should use one or the other, here are some guidelines:

- `runtimeConfig` : Private or public tokens that need to be specified after build using environment variables.
- `app.config` : Public tokens that are determined at build time, website configuration such as theme variant, title and any project config that are not sensitive.

Feature	`runtimeConfig`	`app.config`
Client Side	Hydrated	Bundled
Environment Variables	✅ Yes	❌ No
Reactive	✅ Yes	✅ Yes
Types support	✅ Partial	✅ Yes
Configuration per Request	❌ No	✅ Yes
Hot Module Replacement	❌ No	✅ Yes
Non primitive JS types	❌ No	✅ Yes

External Configuration Files

Nuxt uses `nuxt.config.ts` file as the single source of trust for configurations and skips reading external configuration files. During the course of building your project, you may have a need to configure those. The following table highlights common configurations and, where applicable, how they can be configured with Nuxt.

Name	Config File	How To Configure
Nitro	~~`nitro.config.ts`~~	Use `nitro` key in `nuxt.config`
PostCSS	~~`postcss.config.js`~~	Use `postcss` key in `nuxt.config`

Name	Config File	How To Configure
Vite	~~`vite.config.ts`~~	Use `vite` key in `nuxt.config`
webpack	~~`webpack.config.ts`~~	Use `webpack` key in `nuxt.config`

Here is a list of other common config files:

Name	Config File	How To Configure
TypeScript	`tsconfig.json`	More Info
ESLint	`.eslintrc.js`	More Info
Prettier	`.prettierrc.json`	More Info
Stylelint	`.stylelintrc.json`	More Info
TailwindCSS	`tailwind.config.js`	More Info
Vitest	`vitest.config.ts`	More Info

Vue Configuration

With Vite

If you need to pass options to `@vitejs/plugin-vue` or `@vitejs/plugin-vue-jsx`, you can do this in your `nuxt.config` file.

- `vite.vue` for `@vitejs/plugin-vue`. Check available options here.
- `vite.vueJsx` for `@vitejs/plugin-vue-jsx`. Check available options here.

nuxt.config.ts
```
export default defineNuxtConfig({
  vite: {
    vue: {
      customElement: true
    },
    vueJsx: {
      mergeProps: true
    }
  }
})
```

👉 Read more in Docs > Guide > Directory Structure > Nuxt Config #vue.

With webpack

If you use webpack and need to configure `vue-loader`, you can do this using `webpack.loaders.vue` key inside your `nuxt.config` file. The available options are defined here.

nuxt.config.ts
```
export default defineNuxtConfig({
  webpack: {
    loaders: {
      vue: {
        hotReload: true,
      }
    }
  }
})
```

👉 Read more in Docs > Guide > Directory Structure > Nuxt Config #loaders.

Enabling Experimental Vue Features

You may need to enable experimental features in Vue, such as `defineModel` or `propsDestructure`. Nuxt provides an easy way to do that in `nuxt.config.ts`, no matter which builder you are using:

nuxt.config.ts
```
export default defineNuxtConfig({
  vue: {
    defineModel: true,
    propsDestructure: true
  }
})
```

👉 Read more in Docs > Guide > Directory Structure > Nuxt Config #vue 1.

Edit on Github

Enterprise Design Kit NuxtLabs Nuxt Studio