# Auto-imports

## Composables and Helper Functions

Nuxt auto-imports helper functions, composables and Vue APIs to use across your application without explicitly importing them. Based on the directory structure, every Nuxt application can also use auto-imports for its own components, composables and plugins. Components, composables or plugins can use these functions.

Contrary to a classic global declaration, Nuxt preserves typings and IDEs completions and hints, and only includes what is actually used in your production code.

> 💡 In the documentation, every function that is not explicitly imported is auto-imported by Nuxt and can be used as-is in your code. You can find a reference for auto-imported composables and utilities in the API section.

In the server directory, we auto import exported functions and variables from `server/utils/` .

You can also auto-import functions exported from custom folders or third-party packages by configuring the `imports` section of your `nuxt.config` file.

## Built-in Auto-imports

### Nuxt Auto-imports

Nuxt auto-imports functions and composables to perform data fetching, get access to the app context and runtime config, manage state or define components and plugins.

```
<script setup lang="ts">
/* useAsyncData() and $fetch() are auto-imported */
const { data, refresh, pending } = await useAsyncData('/api/hello', () => $fetch('/api/hello'))
</script>
```

## Vue Auto-imports

Vue 3 exposes Reactivity APIs like `ref` or `computed`, as well as lifecycle hooks and helpers that are auto-imported by Nuxt.

```
<script setup lang="ts">
/* ref() and computed() are auto-imported */
const count = ref(1)
const double = computed(() => count.value * 2)
</script>
```

## Using Vue and Nuxt composables

When you are using the built-in Composition API composables provided by Vue and Nuxt, be aware that many of them rely on being called in the right *context*.

During a component lifecycle, Vue tracks the temporary instance of the current component (and similarly, Nuxt tracks a temporary instance of `nuxtApp`) via a global variable, and then unsets it in same tick. This is essential when server rendering, both to avoid cross-request state pollution (leaking a shared reference between two users) and to avoid leakage between different components.

That means that (with very few exceptions) you cannot use them outside a Nuxt plugin, Nuxt route middleware or Vue setup function. On top of that, you must use them synchronously - that is, you cannot use `await` before calling a composable, except within `<script setup>` blocks, within the setup function of a component declared with `defineNuxtComponent`, in `defineNuxtPlugin` or in `defineNuxtRouteMiddleware`, where we perform a transform to keep the synchronous context even after the `await`.

If you get an error message like `Nuxt instance is unavailable` then it probably means you are calling a Nuxt composable in the wrong place in the Vue or Nuxt lifecycle.

See the full explanation in this comment.

> Documentation for this section is not yet complete. You can contribute to the documentation.

### Example

**Example:** Breaking code:

composables/example.ts

```
// trying to access runtime config outside a composable
const config = useRuntimeConfig()

export const useMyComposable = () => {
  // accessing runtime config here
}
```

**Example:** Fixing the error:

composables/example.ts

```
export const useMyComposable = () => {
  // Because your composable is called in the right place in the lifecycle,
  // useRuntimeConfig will also work
  const config = useRuntimeConfig()

  // ...
}
```

## Directory-based Auto-imports

Nuxt directly auto-imports files created in defined directories:

- `components/` for Vue components.
- `composables/` for Vue composables.
- `utils/` for helper functions and other utilities.

## Explicit Imports

Nuxt exposes every auto-import with the `#imports` alias that can be used to make the import explicit if needed:

```
<script setup lang="ts">
import { ref, computed } from '#imports'

const count = ref(1)
const double = computed(() => count.value * 2)
</script>
```

## Disabling Auto-imports

If you want to disable auto-importing composables and utilities, you can set `imports.autoImport` to `false` in the `nuxt.config` file.

nuxt.config.ts

```
export default defineNuxtConfig({
  imports: {
    autoImport: false
  }
})
```

This will disable auto-imports completely but it's still possible to use explicit imports from `#imports` .

# Auto-imported Components

Nuxt also automatically imports components from your `~/components` directory, although this is configured separately from auto-importing composables and utility functions.

> 👉 Read more in Docs > Guide > Directory Structure > Components.

To disable auto-importing components from your own `~/components` directory, you can set `components.dirs` to an empty array (though note that this will not affect components added by modules).

nuxt.config.ts

```
export default defineNuxtConfig({
  components: {
    dirs: []
  }
```

```
})
```

# Auto-import from third-party packages

Nuxt also allows auto-importing from third-party packages.

> If you are using the Nuxt module for that package, it is likely that the module has already configured auto-imports for that package.

For example, you could enable the auto-import of the `useI18n` composable from the `vue-i18n` package like this:

nuxt.config.ts

```ts
export default defineNuxtConfig({
  imports: {
    presets: [
      {
        from: 'vue-i18n',
        imports: ['useI18n']
      }
    ]
  }
})
```