Runtime Config

Nuxt provides a runtime config API to expose configuration within your application and server routes, with the ability to update it at runtime by setting environment variables.

Exposing Runtime Config

To expose config and environment variables to the rest of your app, you will need to define runtime configuration in your `nuxt.config` file, using the `runtimeConfig` option.

Example:

nuxt.config.ts

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

When adding `apiBase` to the `runtimeConfig.public`, Nuxt adds it to each page payload. We can universally access `apiBase` in both server and browser.

```
const runtimeConfig = useRuntimeConfig()

console.log(runtimeConfig.apiSecret)
console.log(runtimeConfig.public.apiBase)
```

When using Options API the public runtime config is available via `this.$config.public` .

Serialization

Your runtime config will be serialized before being passed to Nitro. This means that anything that cannot be serialized and then deserialized (such as functions, Sets, Maps, and so on), should not be set in your `nuxt.config` .

Instead of passing non-serializable objects or functions into your application from your `nuxt.config` , you can place this code in a Nuxt or Nitro plugin or middleware.

Environment Variables

The most common way to provide configuration is by using Environment Variables.

Nuxi CLI has built-in support for reading your `.env` file in development, build and generate. But when you run your built server, **your `.env` file will not be read**.

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

Runtime config values are automatically replaced by matching environment variables at runtime. There are two key requirements.

1. Your desired variables must be defined in your `nuxt.config` . This ensures that arbitrary environment variables are not exposed to your application code.
2. Only a specially-named environment variable can override a runtime config property. That is, an uppercase environment variable starting with `NUXT_` which uses `_` to separate keys and case changes.

Example

```.env
> NUXT_API_SECRET=api_secret_token
> NUXT_PUBLIC_API_BASE=https://nuxtjs.org
```

```ts
export default defineNuxtConfig({
  runtimeConfig: {
    apiSecret: '', // can be overridden by NUXT_API_SECRET environment variable
    public: {
      apiBase: '', // can be overridden by NUXT_PUBLIC_API_BASE environment variable
    }
  },
})
```

Accessing Runtime Config

Vue App

Within the Vue part of your Nuxt app, you will need to call `useRuntimeConfig()` to access the runtime config.

Note: Behavior is different between the client-side and server-side:

- On the client-side, only keys in `public` are available, and the object is both writable and reactive. The entire runtime config is available on the server-side, but it is read-only to avoid context sharing.

```vue
<script setup lang="ts">
const config = useRuntimeConfig()
console.log('Runtime config:', config)
if (process.server) {
  console.log('API secret:', config.apiSecret)
}
</script>

<template>
  <div>
    <div>Check developer console!</div>
  </div>
</template>
```

🛑 **Security note:** Be careful not to expose runtime config keys to the client-side by either rendering them or passing them to `useState`.

👉 `useRuntimeConfig` only works during `setup` or `Lifecycle Hooks`.

Plugins

If you want to use the runtime config within any (custom) plugin, you can use `useRuntimeConfig()` inside of your `defineNuxtPlugin` function.

For Example:

```
export default defineNuxtPlugin((nuxtApp) => {
  const config = useRuntimeConfig()
  console.log('API base URL:', config.public.apiBase)
});
```

Server Routes

You can access runtime config within the server routes as well using `useRuntimeConfig`.

```
export default async () => {
  const result = await $fetch('https://my.api.com/test', {
    headers: {
      Authorization: `Bearer ${useRuntimeConfig().apiSecret}`
    }
  })
  return result
}
```

Manually Typing Runtime Config

Nuxt tries to automatically generate a typescript interface from provided runtime config using unjs/untyped.

It is also possible to type your runtime config manually:

```ts
declare module 'nuxt/schema' {
  interface RuntimeConfig {
    apiSecret: string
  }
  interface PublicRuntimeConfig {
    apiBase: string
  }
}
// It is always important to ensure you import/export something when augmenting a type
export {}
```

index.d.ts

Edit on Github