SVELTEKIT • BUILD AND DEPLOY

Netlify

ON THIS PAGE

To deploy to Netlify, use `adapter-netlify`.

This adapter will be installed by default when you use `adapter-auto`, but adding it to your project allows you to specify Netlify-specific options.

Usage

Install with `npm i -D @sveltejs/adapter-netlify`, then add the adapter to your `svelte.config.js`:

svelte.config.js

```
import adapter from '@sveltejs/adapter-netlify';

export default {
  kit: {
    // default options are shown
    adapter: adapter({
      // if true, will create a Netlify Edge Function rather
      // than using standard Node-based functions
      edge: false,

      // if true, will split your app into multiple functions
      // instead of creating a single one for the entire app.
      // if `edge` is true, this option cannot be used
      split: false
    })
  }
};
```

Then, make sure you have a netlify.toml file in the project root. This will determine where to write static assets based on the `build.publish` settings, as per this sample configuration:

```
[build]
  command = "npm run build"
  publish = "build"
```

If the `netlify.toml` file or the `build.publish` value is missing, a default value of `"build"` will be used. Note that if you have set the publish directory in the Netlify UI to something else then you will need to set it in `netlify.toml` too, or use the default value of `"build"`.

Node version

New projects will use the current Node LTS version by default. However, if you're upgrading a project you created a while ago it may be stuck on an older version. See the Netlify docs for details on manually specifying a current Node version.

Netlify Edge Functions

SvelteKit supports Netlify Edge Functions. If you pass the option `edge: true` to the `adapter` function, server-side rendering will happen in a Deno-based edge function that's deployed close to the site visitor. If set to `false` (the default), the site will deploy to Node-based Netlify Functions.

svelte.config.js

```
import adapter from '@sveltejs/adapter-netlify';

export default {
  kit: {
    adapter: adapter({
      // will create a Netlify Edge Function using Deno-based
      // rather than using standard Node-based functions
```

```
      edge: true
    })
  }
};
```

Netlify alternatives to SvelteKit functionality

You may build your app using functionality provided directly by SvelteKit without relying on any Netlify functionality. Using the SvelteKit versions of these features will allow them to be used in dev mode, tested with integration tests, and to work with other adapters should you ever decide to switch away from Netlify. However, in some scenarios you may find it beneficial to use the Netlify versions of these features. One example would be if you're migrating an app that's already hosted on Netlify to SvelteKit.

Redirect rules

During compilation, redirect rules are automatically appended to your `_redirects` file. (If it doesn't exist yet, it will be created.) That means:

- `[[redirects]]` in `netlify.toml` will never match as `_redirects` has a higher priority. So always put your rules in the `_redirects` file.
- `_redirects` shouldn't have any custom "catch all" rules such as `/* /foobar/:splat`. Otherwise the automatically appended rule will never be applied as Netlify is only processing the first matching rule.

Netlify Forms

1. Create your Netlify HTML form as described here, e.g. as `/routes/contact/+page.svelte`. (Don't forget to add the hidden `form-name` input element!)

2. Netlify's build bot parses your HTML files at deploy time, which means your form must be prerendered as HTML. You can either add `export const prerender = true` to your `contact.svelte` to prerender just that page or set the `kit.prerender.force: true` option to prerender all pages.

3. If your Netlify form has a custom success message like `<form netlify ... action="/success">` then ensure the corresponding `/routes/success/+page.svelte` exists and is prerendered.

Netlify Functions

With this adapter, SvelteKit endpoints are hosted as Netlify Functions. Netlify function handlers have additional context, including Netlify Identity information. You can access this context via the `event.platform.context` field inside your hooks and `+page.server` or `+layout.server` endpoints. These are serverless functions when the `edge` property is `false` in the adapter config or edge functions when it is `true`.

+page.server.js

```
export const load = async (event) => {
  const context = event.platform.context;
  console.log(context); // shows up in your functions log in the Netlify app
};
```

Additionally, you can add your own Netlify functions by creating a directory for them and adding the configuration to your `netlify.toml` file. For example:

```
[build]
  command = "npm run build"
  publish = "build"

[functions]
  directory = "functions"
```

Troubleshooting

Accessing the file system

You can't use `fs` in edge deployments.

You *can* use it in serverless deployments, but it won't work as expected, since files are not copied from your project into your deployment. Instead, use the `read` function from `$app/server` to access your files. `read` does not work inside edge deployments (this may change in future).

Alternatively, you can prerender the routes in question.

Edit this page on GitHub

PREVIOUS
Cloudflare Workers

NEXT
Vercel