SVELTEKIT • BUILD AND DEPLOY

Vercel

ON THIS PAGE

To deploy to Vercel, use `adapter-vercel`.

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

Usage

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

svelte.config.js

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

export default {
  kit: {
    adapter: adapter({
      // see below for options that can be set here
    })
  }
};
```

Deployment configuration

To control how your routes are deployed to Vercel as functions, you can specify deployment configuration, either through the option shown above or with `export const`

about/+page.ts

```ts
import type { Config } from '@sveltejs/adapter-vercel';

export const config: Config = {
  runtime: 'edge'
};
```

...and others as Serverless Functions (note that by specifying `config` inside a layout, it applies to all child pages):

admin/+layout.ts

```ts
import type { Config } from '@sveltejs/adapter-vercel';

export const config: Config = {
  runtime: 'nodejs18.x'
};
```

The following options apply to all functions:

- `runtime` : `'edge'` , `'nodejs18.x'` or `'nodejs20.x'` . By default, the adapter will select the `'nodejs<version>.x'` corresponding to the Node version your project is configured to use on the Vercel dashboard
- `regions` : an array of edge network regions (defaulting to `["iad1"]` for serverless functions) or `'all'` if `runtime` is `edge` (its default). Note that multiple regions for serverless functions are only supported on Enterprise plans
- `split` : if `true` , causes a route to be deployed as an individual function. If `split` is set to `true` at the adapter level, all routes will be deployed as individual functions

Additionally, the following option applies to edge functions:

- `external` : an array of dependencies that esbuild should treat as external when bundling functions. This should only be used to exclude optional dependencies that will not run outside Node

can be decreased to `128` Mb or increased in 64Mb increments up to `3008` Mb on Pro or Enterprise accounts

`maxDuration` : maximum execution duration of the function. Defaults to `10` seconds for Hobby accounts, `15` for Pro and `900` for Enterprise

`isr` : configuration Incremental Static Regeneration, described below

If your functions need to access data in a specific region, it's recommended that they be deployed in the same region (or close to it) for optimal performance.

Image Optimization

You may set the `images` config to control how Vercel builds your images. See the image configuration reference for full details. As an example, you may set:

svelte.config.js

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

export default {
  kit: {
    adapter({
      images: {
        sizes: [640, 828, 1200, 1920, 3840],
        formats: ['image/avif', 'image/webp'],
        minimumCacheTTL: 300,
        domains: ['example-app.vercel.app'],
      }
    })
  }
};
```

Vercel supports Incremental Static Regeneration (ISR), which provides the performance and cost advantages of prerendered content with the flexibility of dynamically rendered content.

To add ISR to a route, include the `isr` property in your `config` object:

```
import { BYPASS_TOKEN } from '$env/static/private';

export const config = {
  isr: {
    // Expiration time (in seconds) before the cached asset will be re-generated by invok
    // Setting the value to `false` means it will never expire.
    expiration: 60,

    // Random token that can be provided in the URL to bypass the cached version of the a
    // with a __prerender_bypass=<token> cookie.
    //
    // Making a `GET` or `HEAD` request with `x-prerender-revalidate: <token>` will force
    bypassToken: BYPASS_TOKEN,

    // List of valid query parameters. Other parameters (such as utm tracking codes) will
    // ensuring that they do not result in content being regenerated unnecessarily
    allowQuery: ['search']
  }
};
```

The `expiration` property is required; all others are optional.

> Pages that are prerendered will ignore ISR configuration.

Environment variables

Vercel makes a set of deployment-specific environment variables available. Like other environment variables, these are accessible from `$env/static/private` and `$env/dynamic/private` (sometimes — more on that later), and inaccessible from their

```ts
import { VERCEL_COMMIT_REF } from '$env/static/private';
import type { LayoutServerLoad } from './$types';

export const load: LayoutServerLoad = () => {
  return {
    deploymentGitBranch: VERCEL_COMMIT_REF
  };
};
```

+layout.svelte JS TS

```svelte
<script lang="ts">
  import type { LayoutServerData } from './$types';

  let { data }: { data: LayoutServerData } = $props();
</script>

<p>This staging environment was deployed from {data.deploymentGitBranch}.</p>
```

Since all of these variables are unchanged between build time and run time when building on Vercel, we recommend using `$env/static/private` — which will statically replace the variables, enabling optimisations like dead code elimination — rather than `$env/dynamic/private`.

Skew protection

When a new version of your app is deployed, assets belonging to the previous version may no longer be accessible. If a user is actively using your app when this happens, it can cause errors when they navigate — this is known as *version skew*. SvelteKit mitigates this by detecting errors resulting from version skew and causing a hard reload to get the latest version of the app, but this will cause any client-side state to be lost. (You can also proactively mitigate it by observing the `updated` store value, which tells clients when a new version has been deployed.)

Skew protection is a Vercel feature that routes client requests to their original deployment.

Docs

they reload the page, they will get the newest deployment. (The `updated` store is exempted from this behaviour, and so will continue to report new deployments.) To enable it, visit the Advanced section of your project settings on Vercel.

Cookie-based skew protection comes with one caveat: if a user has multiple versions of your app open in multiple tabs, requests from older versions will be routed to the newer one, meaning they will fall back to SvelteKit's built-in skew protection.

Notes

Vercel functions

If you have Vercel functions contained in the `api` directory at the project's root, any requests for `/api/*` will *not* be handled by SvelteKit. You should implement these as API routes in your SvelteKit app instead, unless you need to use a non-JavaScript language in which case you will need to ensure that you don't have any `/api/*` routes in your SvelteKit app.

Node version

Projects created before a certain date may default to using an older Node version than what SvelteKit currently requires. You can change the Node version in your project settings.

Troubleshooting

Accessing the file system

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

You *can* use it in serverless functions, but it won't work as expected, since files are not

functions (this may change in future).

Alternatively, you can prerender the routes in question.

Edit this page on GitHub

PREVIOUS
Netlify

NEXT
Writing adapters