SVELTEKIT • BEST PRACTICES

Images

ON THIS PAGE

Images can have a big impact on your app's performance. For best results, you should optimize them by doing the following:

- generate optimal formats like `.avif` and `.webp`
- create different sizes for different screens
- ensure that assets can be cached effectively

Doing this manually is tedious. There are a variety of techniques you can use, depending on your needs and preferences.

Vite's built-in handling

Vite will automatically process imported assets for improved performance. This includes assets referenced via the CSS `url()` function. Hashes will be added to the filenames so that they can be cached, and assets smaller than `assetsInlineLimit` will be inlined. Vite's asset handling is most often used for images, but is also useful for video, audio, etc.

```
<script>
  import logo from '$lib/assets/logo.png';
</script>

<img alt="The project logo" src={logo} />
```

provides plug and play image processing that serves smaller file formats like `avif` or `webp`, automatically sets the intrinsic `width` and `height` of the image to avoid layout shift, creates images of multiple sizes for various devices, and strips EXIF data for privacy. It will work in any Vite-based project including, but not limited to, SvelteKit projects.

> As a build plugin, `@sveltejs/enhanced-img` can only optimize files located on your machine during the build process. If you have an image located elsewhere (such as a path served from your database, CMS, or backend), please read about loading images dynamically from a CDN.
>
> WARNING: The `@sveltejs/enhanced-img` package is experimental. It uses pre-1.0 versioning and may introduce breaking changes with every minor version release.

Setup

Install:

```
npm install --save-dev @sveltejs/enhanced-img
```

Adjust `vite.config.js`:

```
import { sveltekit } from '@sveltejs/kit/vite';
import { enhancedImages } from '@sveltejs/enhanced-img';
import { defineConfig } from 'vite';

export default defineConfig({
  plugins: [
    enhancedImages(),
    sveltekit()
  ]
});
```

Building will take longer on the first build due to the computational expense of transforming images. However, the build output will be cached in `./node_modules/.cache/imagetools` so that subsequent builds will be fast.

Use in your `.svelte` components by using `<enhanced:img>` rather than `<img>` and referencing the image file with a Vite asset import path:

```
<enhanced:img src="./path/to/your/image.jpg" alt="An alt text" />
```

At build time, your `<enhanced:img>` tag will be replaced with an `<img>` wrapped by a `<picture>` providing multiple image types and sizes. It's only possible to downscale images without losing quality, which means that you should provide the highest resolution image that you need — smaller versions will be generated for the various device types that may request an image.

You should provide your image at 2x resolution for HiDPI displays (a.k.a. retina displays). `<enhanced:img>` will automatically take care of serving smaller versions to smaller devices.

If you wish to add styles to your `<enhanced:img>`, you should add a `class` and target that.

Dynamically choosing an image

You can also manually import an image asset and pass it to an `<enhanced:img>`. This is useful when you have a collection of static images and would like to dynamically choose one or iterate over them. In this case you will need to update both the `import` statement and `<img>` element as shown below to indicate you'd like process them.

```
<script>
  import MyImage from './path/to/your/image.jpg?enhanced';
</script>

<enhanced:img src={MyImage} alt="some alt text" />
```

You can also use Vite's `import.meta.glob`. Note that you will have to specify `enhanced` via a custom query:

```
```

```
      query: {
        enhanced: true
      }
    }
  )
</script>

{#each Object.entries(imageModules) as [_path, module]}
  <enhanced:img src={module.default} alt="some alt text" />
{/each}
```

Intrinsic Dimensions

`width` and `height` are optional as they can be inferred from the source image and will be automatically added when the `<enhanced:img>` tag is preprocessed. With these attributes, the browser can reserve the correct amount of space, preventing layout shift. If you'd like to use a different `width` and `height` you can style the image with CSS. Because the preprocessor adds a `width` and `height` for you, if you'd like one of the dimensions to be automatically calculated then you will need to specify that:

```
<style>
  .hero-image img {
    width: var(--size);
    height: auto;
  }
</style>
```

srcset and sizes

If you have a large image, such as a hero image taking the width of the design, you should specify `sizes` so that smaller versions are requested on smaller devices. E.g. if you have a 1280px image you may want to specify something like:

```
<enhanced:img src="./image.png" sizes="min(1280px, 100vw)"/>
```

populate the `srcset` attribute.

The smallest picture generated automatically will have a width of 540px. If you'd like smaller images or would otherwise like to specify custom widths, you can do that with the `w` query parameter:

```
<enhanced:img
  src="./image.png?w=1280;640;400"
  sizes="(min-width:1920px) 1280px, (min-width:1080px) 640px, (min-width:768px) 400px"
/>
```

If `sizes` is not provided, then a HiDPI/Retina image and a standard resolution image will be generated. The image you provide should be 2x the resolution you wish to display so that the browser can display that image on devices with a high device pixel ratio.

Per-image transforms

By default, enhanced images will be transformed to more efficient formats. However, you may wish to apply other transforms such as a blur, quality, flatten, or rotate operation. You can run per-image transforms by appending a query string:

```
<enhanced:img src="./path/to/your/image.jpg?blur=15" alt="An alt text" />
```

See the imagetools repo for the full list of directives.

Loading images dynamically from a CDN

In some cases, the images may not be accessible at build time — e.g. they may live inside a content management system or elsewhere.

Using a content delivery network (CDN) can allow you to optimize these images dynamically, and provides more flexibility with regards to sizes, but it may involve some

Building HTML to target CDNs allows using an `<img>` tag since the CDN can serve the appropriate format based on the `User-Agent` header, whereas build-time optimizations must produce `<picture>` tags with multiple sources. Finally, some CDNs may generate images lazily, which could have a negative performance impact for sites with low traffic and frequently changing images.

CDNs can generally be used without any need for a library. However, there are a number of libraries with Svelte support that make it easier. `@unpic/svelte` is a CDN-agnostic library with support for a large number of providers. You may also find that specific CDNs like Cloudinary have Svelte support. Finally, some content management systems (CMS) which support Svelte (such as Contentful, Storyblok, and Contentstack) have built-in support for image handling.

Best practices

For each image type, use the appropriate solution from those discussed above. You can mix and match all three solutions in one project. For example, you may use Vite's built-in handling to provide images for `<meta>` tags, display images on your homepage with `@sveltejs/enhanced-img`, and display user-submitted content with a dynamic approach.

Consider serving all images via CDN regardless of the image optimization types you use. CDNs reduce latency by distributing copies of static assets globally.

Your original images should have a good quality/resolution and should have 2x the width it will be displayed at to serve HiDPI devices. Image processing can size images down to save bandwidth when serving smaller screens, but it would be a waste of bandwidth to invent pixels to size images up.

For images which are much larger than the width of a mobile device (roughly 400px), such as a hero image taking the width of the page design, specify `sizes` so that smaller images can be served on smaller devices.

For important images, such as the largest contentful paint (LCP) image, set

while the page is loading affecting your cumulative layout shift (CLS). `width` and `height` help the browser to reserve space while the image is still loading, so `@sveltejs/enhanced-img` will add a `width` and `height` for you.

Always provide a good `alt` text. The Svelte compiler will warn you if you don't do this.

Do not use `em` or `rem` in `sizes` and change the default size of these measures. When used in `sizes` or `@media` queries, `em` and `rem` are both defined to mean the user's default `font-size`. For a `sizes` declaration like `sizes="(min-width: 768px) min(100vw, 108rem), 64rem"`, the actual `em` or `rem` that controls how the image is laid out on the page can be different if changed by CSS. For example, do not do something like `html { font-size: 62.5%; }` as the slot reserved by the browser preloader will now end up being larger than the actual slot of the CSS object model once it has been created.

Edit this page on GitHub

PREVIOUS
Performance

NEXT
Accessibility