Image Component and Image Optimization

▼ **Examples**

- Image Component

The Next.js Image component, `next/image`, is an extension of the HTML `<img>` element, evolved for the modern web. It includes a variety of built-in performance optimizations to help you achieve good Core Web Vitals. These scores are an important measurement of user experience on your website, and are factored into Google's search rankings.

Some of the optimizations built into the Image component include:

- **Improved Performance:** Always serve correctly sized image for each device, using modern image formats
- **Visual Stability:** Prevent Cumulative Layout Shift automatically
- **Faster Page Loads:** Images are only loaded when they enter the viewport, with optional blur-up placeholders
- **Asset Flexibility:** On-demand image resizing, even for images stored on remote servers

Using the Image Component

To add an image to your application, import the `next/image` component:

```
import Image from 'next/image'
```

Now, you can define the `src` for your image (either local or remote).

Local Images

To use a local image, `import` your `.jpg`, `.png`, or `.webp` files:

```
import profilePic from '../public/me.png'
```

Dynamic `await import()` or `require()` are *not* supported. The `import` must be static so it can be analyzed at build time.

Next.js will automatically determine the `width` and `height` of your image based on the imported file. These values are used to prevent Cumulative Layout Shift while your image is loading.

```
import Image from 'next/image'
import profilePic from '../public/me.png'

function Home() {
  return (
    <>
      <h1>My Homepage</h1>
      <Image
        src={profilePic}
        alt="Picture of the author"
        // width={500} automatically provided
        // height={500} automatically provided
        // blurDataURL="data:..." automatically provided
```

```
        // placeholder="blur" // Optional blur-up while loading
      />
      <p>Welcome to my homepage!</p>
    </>
  )
}
```

Remote Images

To use a remote image, the `src` property should be a URL string, which can be relative or absolute. Because Next.js does not have access to remote files during the build process, you'll need to provide the `width`, `height` and optional `blurDataURL` props manually:

```
import Image from 'next/image'

export default function Home() {
  return (
    <>
      <h1>My Homepage</h1>
      <Image
        src="/me.png"
        alt="Picture of the author"
        width={500}
        height={500}
      />
      <p>Welcome to my homepage!</p>
    </>
  )
}
```

> *Learn more about the sizing requirements in `next/image`.*

Domains

Sometimes you may want to access a remote image, but still use the built-in Next.js Image Optimization API. To do this, leave the `loader` at its default setting and enter an absolute URL for the Image `src`.

To protect your application from malicious users, you must define a list of remote domains that you intend to access this way. This is configured in your `next.config.js` file, as shown below:

```
module.exports = {
  images: {
    domains: ['example.com', 'example2.com'],
  },
}
```

Loaders

Note that in the example earlier, a partial URL (`"/me.png"`) is provided for a remote image. This is possible because of the `next/image` loader architecture.

A loader is a function that generates the URLs for your image. It appends a root domain to your provided `src`, and generates multiple URLs to request the image at different sizes. These multiple URLs are used in the automatic srcset generation, so that visitors to your site will be served an image that is the right size for their viewport.

The default loader for Next.js applications uses the built-in Image Optimization API, which optimizes images from anywhere on the web, and then serves them directly from the Next.js web server. If you would like to serve your images directly from a CDN or image server, you can use one of the built-in loaders or write your own with a few lines of JavaScript.

Loaders can be defined per-image, or at the application level.

Priority

You should add the `priority` property to the image that will be the Largest Contentful Paint (LCP) element for each page. Doing so allows Next.js to specially prioritize the image for loading (e.g. through preload tags or priority hints), leading to a meaningful boost in LCP.

The LCP element is typically the largest image or text block visible within the viewport of the page. When you run `next dev`, you'll see a console warning if the LCP element is an `<Image>` without the `priority` property.

Once you've identified the LCP image, you can add the property like this:

```
import Image from 'next/image'

export default function Home() {
  return (
    <>
      <h1>My Homepage</h1>
      <Image
        src="/me.png"
        alt="Picture of the author"
        width={500}
        height={500}
        priority
      />
      <p>Welcome to my homepage!</p>
    </>
  )
}
```

See more about priority in the `next/image` component documentation.

Image Sizing

One of the ways that images most commonly hurt performance is through *layout shift*, where the image pushes other elements around on the page as it loads in. This performance problem is so annoying to users that it has its own Core Web Vital, called Cumulative Layout Shift. The way to avoid image-based layout shifts is to always size your images. This allows the browser to reserve precisely enough space for the image before it loads.

Because `next/image` is designed to guarantee good performance results, it cannot be used in a way that will contribute to layout shift, and **must** be sized in one of three ways:

1. Automatically, using a static import

2. Explicitly, by including a `width` and `height` property
3. Implicitly, by using `layout="fill"` which causes the image to expand to fill its parent element.

> ***What if I don't know the size of my images?***
>
> *If you are accessing images from a source without knowledge of the images' sizes, there are several things you can do:*
>
> ***Use `layout='fill'`***
>
> *The `fill` layout mode allows your image to be sized by its parent element. Consider using CSS to give the image's parent element space on the page, then using the `objectFit property` with `fill`, `contain`, or `cover`, along with the `objectPosition property` to define how the image should occupy that space.*
>
> ***Normalize your images***
>
> *If you're serving images from a source that you control, consider modifying your image pipeline to normalize the images to a specific size.*
>
> ***Modify your API calls***
>
> *If your application is retrieving image URLs using an API call (such as to a CMS), you may be able to modify the API call to return the image dimensions along with the URL.*

If none of the suggested methods works for sizing your images, the `next/image` component is designed to work well on a page alongside standard `<img>` elements.

Styling

Styling the Image component is not that different from styling a normal `<img>` element, but there are a few guidelines to keep in mind:

Pick the correct layout mode

The image component has several different layout modes that define how it is sized on the page. If the styling of your image isn't turning out the way you want, consider experimenting with other layout modes.

Target the image with className, not based on DOM structure

For most layout modes, the Image component will have a DOM structure of one `<img>` tag wrapped by exactly one `<span>`. For some modes, it may also have a sibling `<span>` for spacing. These additional `<span>` elements are critical to allow the component to prevent layout shifts.

The recommended way to style the inner `<img>` is to set the `className` prop on the Image component to the value of an imported CSS Module. The value of `className` will be automatically applied to the underlying `<img>` element.

Alternatively, you can import a global stylesheet and manually set the `className` prop to the same name used in the global stylesheet.

You cannot use styled-jsx because it's scoped to the current component.

When using `layout='fill'`, the parent element must have `position: relative`

This is necessary for the proper rendering of the image element in that layout mode.

When using `layout='responsive'`, the parent element must have `display: block`

This is the default for `<div>` elements but should be specified otherwise.

Properties

View all properties available to the `next/image` component.

Styling Examples

For examples of the Image component used with the various fill modes, see the Image component example app.

Configuration

The `next/image` component and Next.js Image Optimization API can be configured in the `next.config.js` file. These configurations allow you to enable remote domains, define custom image breakpoints, change caching behavior and more.

Read the full image configuration documentation for more information.

Related

For more information on what to do next, we recommend the following sections:

next/image See all available properties for the Image component