next/image

▶ **Examples**

▶ **Version History**

> ***Note: This is API documentation for the Image Component and Image Optimization. For a feature overview and usage information for images in Next.js, please see Images.***

Required Props

The `<Image />` component requires the following properties.

src

Must be one of the following:

1. A statically imported image file, or
2. A path string. This can be either an absolute external URL, or an internal path depending on the loader prop or loader configuration.

When using an external URL, you must add it to domains in `next.config.js`.

width

The `width` property can represent either the *rendered* width or *original* width in pixels, depending on the `layout` and `sizes` properties.

When using `layout="intrinsic"`, `layout="fixed"`, or `layout="raw"` without `sizes`, the `width` property represents the *rendered* width in pixels, so it will affect how large the image appears.

When using `layout="responsive"`, `layout="fill"`, or `layout="raw"` with `sizes`, the `width` property represents the *original* width in pixels, so it will only affect the aspect ratio.

The `width` property is required, except for statically imported images, or those with `layout="fill"`.

height

The `height` property can represent either the *rendered* height or *original* height in pixels, depending on the `layout` and `sizes` properties.

When using `layout="intrinsic"`, `layout="fixed"`, or `layout="raw"` without `sizes`, the `height` property represents the *rendered* height in pixels, so it will affect how large the image appears.

When using `layout="responsive"`, `layout="fill"`, or `layout="raw"` with `sizes`, the `height` property represents the *original* height in pixels, so it will only affect the aspect ratio.

The `height` property is required, except for statically imported images, or those with `layout="fill"`.

Optional Props

The `<Image />` component accepts a number of additional properties beyond those which are required. This section describes the most commonly-used properties of the Image component. Find details about more rarely-used properties in the Advanced Props section.

layout

The layout behavior of the image as the viewport changes size.

`layout`	Behavior	`srcSet`	`sizes`	Has wrapper and sizer
`intrinsic` (default)	Scale *down* to fit width of container, up to image size	`1x`, `2x` (based on imageSizes)	N/A	yes
`fixed`	Sized to `width` and `height` exactly	`1x`, `2x` (based on imageSizes)	N/A	yes
`responsive`	Scale to fit width of container	`640w`, `750w`, ... `2048w`, `3840w` (based on imageSizes and deviceSizes)	`100vw`	yes
`fill`	Grow in both X and Y axes to fill container	`640w`, `750w`, ... `2048w`, `3840w` (based on imageSizes and deviceSizes)	`100vw`	yes
`raw`*	Insert the image element with no automatic layout behavior	Behaves like `responsive` if the image has the `sizes` prop, and like `fixed` if it does not	optional	no

- Demo the `intrinsic` layout (default)
  - When `intrinsic`, the image will scale the dimensions down for smaller viewports, but maintain the original dimensions for larger viewports.
- Demo the `fixed` layout
  - When `fixed`, the image dimensions will not change as the viewport changes (no responsiveness) similar to the native `img` element.
- Demo the `responsive` layout
  - When `responsive`, the image will scale the dimensions down for smaller viewports and scale up for larger viewports.
  - Ensure the parent element uses `display: block` in their stylesheet.
- Demo the `fill` layout
  - When `fill`, the image will stretch both width and height to the dimensions of the parent element, provided the parent element is relative.
  - This is usually paired with the `objectFit` property.
  - Ensure the parent element has `position: relative` in their stylesheet.
- When `raw` *, the image will be rendered as a single image element with no wrappers, sizers or other responsive behavior.
  - If your image styling will change the size of a `raw` image, you should include the `sizes` property for proper image serving. Otherwise your image will receive a fixed height and width.
  - The other layout modes are optimized for performance and should cover nearly all use cases. It is recommended to try to use those modes before using `raw`.
- Demo background image

loader

A custom function used to resolve URLs. Setting the loader as a prop on the Image component overrides the default loader defined in the `images` section of `next.config.js`.

A `loader` is a function returning a URL string for the image, given the following parameters:

- `src`
- `width`
- `quality`

Here is an example of using a custom loader with `next/image`:

```
import Image from 'next/image'

const myLoader = ({ src, width, quality }) => {
  return `https://example.com/${src}?w=${width}&q=${quality || 75}`
}

const MyImage = (props) => {
  return (
    <Image
      loader={myLoader}
      src="me.png"
      alt="Picture of the author"
      width={500}
      height={500}
    />
  )
}
```

sizes

A string that provides information about how wide the image will be at different breakpoints. Defaults to `100vw` (the full width of the screen) when using `layout="responsive"` or `layout="fill"`.

If you are using `layout="fill"`, `layout="responsive"`, or `layout="raw"` * it's important to assign `sizes` for any image that takes up less than the full viewport width.

For example, when the parent element will constrain the image to always be less than half the viewport width, use `sizes="50vw"`. Without `sizes`, the image will be sent at twice the necessary resolution, decreasing performance.

If you are using `layout="intrinsic"` or `layout="fixed"`, then `sizes` is not needed because the upper bound width is constrained already.

Learn more.

quality

The quality of the optimized image, an integer between `1` and `100` where `100` is the best quality. Defaults to `75`.

priority

When true, the image will be considered high priority and preload. Lazy loading is automatically disabled for images using `priority`.

You should use the `priority` property on any image detected as the Largest Contentful Paint (LCP) element. It may be appropriate to have multiple priority images, as different images may be the LCP element for different viewport sizes.

Should only be used when the image is visible above the fold. Defaults to `false`.

placeholder

A placeholder to use while the image is loading. Possible values are `blur` or `empty`. Defaults to `empty`.

When `blur`, the `blurDataURL` property will be used as the placeholder. If `src` is an object from a static import and the imported image is `.jpg`, `.png`, `.webp`, or `.avif`, then `blurDataURL` will be automatically populated.

For dynamic images, you must provide the `blurDataURL` property. Solutions such as Plaiceholder can help with `base64` generation.

When `empty`, there will be no placeholder while the image is loading, only empty space.

Try it out:

- Demo the `blur` placeholder
- Demo the shimmer effect with `blurDataURL` prop
- Demo the color effect with `blurDataURL` prop

Advanced Props

In some cases, you may need more advanced usage. The `<Image />` component optionally accepts the following advanced properties.

style

Allows passing CSS styles to the underlying image element.

Note that all `layout` modes other than `"raw"` * apply their own styles to the image element, and these automatic styles take precedence over the `style` prop.

objectFit

Defines how the image will fit into its parent container when using `layout="fill"`.

This value is passed to the object-fit CSS property for the `src` image.

objectPosition

Defines how the image is positioned within its parent element when using `layout="fill"`.

This value is passed to the object-position CSS property applied to the image.

onLoadingComplete

A callback function that is invoked once the image is completely loaded and the placeholder has been removed.

The `onLoadingComplete` function accepts one parameter, an object with the following properties:

- `naturalWidth`
- `naturalHeight`

loading

> ***Attention***: *This property is only meant for advanced usage. Switching an image to load with* `eager` *will normally* ***hurt performance***.
>
> *We recommend using the* `priority` *property instead, which properly loads the image eagerly for nearly all use cases.*

The loading behavior of the image. Defaults to `lazy`.

When `lazy`, defer loading the image until it reaches a calculated distance from the viewport.

When `eager`, load the image immediately.

Learn more

blurDataURL

A Data URL to be used as a placeholder image before the `src` image successfully loads. Only takes effect when combined with `placeholder="blur"`.

Must be a base64-encoded image. It will be enlarged and blurred, so a very small image (10px or less) is recommended. Including larger images as placeholders may harm your application performance.

Try it out:

- Demo the default `blurDataURL` prop
- Demo the shimmer effect with `blurDataURL` prop
- Demo the color effect with `blurDataURL` prop

You can also generate a solid color Data URL to match the image.

lazyBoundary

A string (with similar syntax to the margin property) that acts as the bounding box used to detect the intersection of the viewport with the image and trigger lazy loading. Defaults to `"200px"`.

If the image is nested in a scrollable parent element other than the root document, you will also need to assign the lazyRoot prop.

Learn more

lazyRoot

A React Ref pointing to the scrollable parent element. Defaults to `null` (the document viewport).

The Ref must point to a DOM element or a React component that forwards the Ref to the underlying DOM element.

Example pointing to a DOM element

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

const lazyRoot = React.useRef(null)

const Example = () => (
  <div ref={lazyRoot} style={{ overflowX: 'scroll', width: '500px' }}>
    <Image lazyRoot={lazyRoot} src="/one.jpg" width="500" height="500" />
    <Image lazyRoot={lazyRoot} src="/two.jpg" width="500" height="500" />
  </div>
)
```

Example pointing to a React component

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

const Container = React.forwardRef((props, ref) => {
  return (
    <div ref={ref} style={{ overflowX: 'scroll', width: '500px' }}>
      {props.children}
    </div>
  )
})

const Example = () => {
  const lazyRoot = React.useRef(null)

  return (
    <Container ref={lazyRoot}>
      <Image lazyRoot={lazyRoot} src="/one.jpg" width="500" height="500" />
      <Image lazyRoot={lazyRoot} src="/two.jpg" width="500" height="500" />
    </Container>
  )
}
```

Learn more

unoptimized

When true, the source image will be served as-is instead of changing quality, size, or format. Defaults to `false`.

Other Props

Other properties on the `<Image />` component will be passed to the underlying `img` element with the exception of the following:

- `srcSet`. Use Device Sizes instead.
- `ref`. Use `onLoadingComplete` instead.
- `decoding`. It is always `"async"`.

Configuration Options

Domains

To protect your application from malicious users, you must define a list of image provider domains that you want to be served from the Next.js Image Optimization API. This is configured in with the `domains` property in your `next.config.js` file, as shown below:

```
module.exports = {
  images: {
    domains: ['assets.acme.com'],
  },
}
```

Loader Configuration

If you want to use a cloud provider to optimize images instead of using the Next.js built-in Image Optimization API, you can configure the `loader` and `path` prefix in your `next.config.js` file. This allows you to use relative URLs for the Image `src` and automatically generate the correct absolute URL for your provider.

```
module.exports = {
  images: {
    loader: 'imgix',
    path: 'https://example.com/myaccount/',
  },
}
```

Built-in Loaders

The following Image Optimization cloud providers are included:

- Default: Works automatically with `next dev`, `next start`, or a custom server
- Vercel: Works automatically when you deploy on Vercel, no configuration necessary. Learn more
- Imgix: `loader: 'imgix'`
- Cloudinary: `loader: 'cloudinary'`
- Akamai: `loader: 'akamai'`
- Custom: `loader: 'custom'` use a custom cloud provider by implementing the `loader` prop on the `next/image` component

If you need a different provider, you can use the `loader` prop with `next/image`.

> *Images can not be optimized at build time using `next export`, only on-demand. To use `next/image` with `next export`, you will need to use a different loader than the default. Read more in the discussion.*

> *The `next/image` component's default loader uses `squoosh` because it is quick to install and suitable for a development environment. When using `next start` in your production environment, it is strongly recommended that you install `sharp` by running `yarn add sharp` in your project directory. This is not necessary for Vercel deployments, as `sharp` is installed automatically.*

Advanced

The following configuration is for advanced use cases and is usually not necessary. If you choose to configure the properties below, you will override any changes to the Next.js defaults in future updates.

Device Sizes

If you know the expected device widths of your users, you can specify a list of device width breakpoints using the `deviceSizes` property in `next.config.js`. These widths are used when the `next/image` component uses `layout="responsive"` or `layout="fill"` to ensure the correct image is served for user's device.

If no configuration is provided, the default below is used.

```
module.exports = {
  images: {
    deviceSizes: [640, 750, 828, 1080, 1200, 1920, 2048, 3840],
  },
}
```

Image Sizes

You can specify a list of image widths using the `images.imageSizes` property in your `next.config.js` file. These widths are concatenated with the array of device sizes to form the full array of sizes used to generate image srcsets.

The reason there are two separate lists is that imageSizes is only used for images which provide a `sizes` prop, which indicates that the image is less than the full width of the screen. **Therefore, the sizes in imageSizes should all be smaller than the smallest size in deviceSizes.**

If no configuration is provided, the default below is used.

```
module.exports = {
  images: {
    imageSizes: [16, 32, 48, 64, 96, 128, 256, 384],
  },
}
```

Acceptable Formats

The default Image Optimization API will automatically detect the browser's supported image formats via the request's `Accept` header.

If the `Accept` head matches more than one of the configured formats, the first match in the array is used. Therefore, the array order matters. If there is no match (or the source image is animated), the Image Optimization API will fallback to the original image's format.

If no configuration is provided, the default below is used.

```
module.exports = {
  images: {
    formats: ['image/webp'],
  },
}
```

You can enable AVIF support with the following configuration.

```
module.exports = {
  images: {
    formats: ['image/avif', 'image/webp'],
  },
}
```

> *Note: AVIF generally takes 20% longer to encode but it compresses 20% smaller compared to WebP. This means that the first time an image is requested, it will typically be slower and then subsequent requests that are cached will be faster.*

Caching Behavior

The following describes the caching algorithm for the default loader. For all other loaders, please refer to your cloud provider's documentation.

Images are optimized dynamically upon request and stored in the `<distDir>/cache/images` directory. The optimized image file will be served for subsequent requests until the expiration is reached. When a request is made that matches a cached but expired file, the expired image is served stale immediately. Then the image is optimized again in the background (also called revalidation) and saved to the cache with the new expiration date.

The expiration (or rather Max Age) is defined by either the `minimumCacheTTL` configuration or the upstream server's `Cache-Control` header, whichever is larger. Specifically, the `max-age` value of the `Cache-Control` header is used. If both `s-maxage` and `max-age` are found, then `s-maxage` is preferred.

- You can configure `minimumCacheTTL` to increase the cache duration when the upstream image does not include `Cache-Control` header or the value is very low.
- You can configure `deviceSizes` and `imageSizes` to reduce the total number of possible generated images.
- You can configure formats to disable multiple formats in favor of a single image format.

Minimum Cache TTL

You can configure the Time to Live (TTL) in seconds for cached optimized images. In many cases, it's better to use a Static Image Import which will automatically hash the file contents and cache the image forever with a `Cache-Control` header of `immutable`.

```
module.exports = {
  images: {
    minimumCacheTTL: 60,
  },
}
```

If you need to add a `Cache-Control` header for the browser (not recommended), you can configure `headers` on the upstream image e.g. `/some-asset.jpg` not `/_next/image` itself.

Disable Static Imports

The default behavior allows you to import static files such as `import icon from './icon.png` and then pass that to the `src` property.

In some cases, you may wish to disable this feature if it conflicts with other plugins that expect the import to behave differently.

You can disable static image imports inside your `next.config.js` :

```
module.exports = {
  images: {
    disableStaticImages: true,
  },
}
```

Dangerously Allow SVG

The default loader does not optimize SVG images for a few reasons. First, SVG is a vector format meaning it can be resized losslessly. Second, SVG has many of the same features as HTML/CSS, which can lead to vulnerabilities without proper Content Security Policy (CSP) headers.

If you need to serve SVG images with the default Image Optimization API, you can set `dangerouslyAllowSVG` and `contentSecurityPolicy` inside your `next.config.js` :

```
module.exports = {
  images: {
    dangerouslyAllowSVG: true,
    contentSecurityPolicy: "default-src 'self'; script-src 'none'; sandbox;",
  },
}
```

Experimental "raw" layout mode

The image component currently supports an additional `layout="raw"` mode, which renders the image without wrappers or styling. This layout mode is currently an experimental feature, while user feedback is gathered. As there is the possibility of breaking changes to the `layout="raw"` interface, the feature is locked behind an experimental feature flag. If you would like to use the `raw` layout mode, you must add the following to your `next.config.js` :

```
module.exports = {
  experimental: {
    images: {
      layoutRaw: true,
    },
  },
}
```

> *Note on CLS with `layout="raw"` : It is possible to cause layout shift with the image component in `raw` mode. If you include a `sizes` property, the image component will not pass `height` and `width` attributes to the image, to allow you to apply your own responsive sizing. An aspect-ratio style property is automatically applied to prevent layout shift, but this won't apply on older browsers.*

Animated Images

The default loader will automatically bypass Image Optimization for animated images and serve the image as-is.

Auto-detection for animated files is best-effort and supports GIF, APNG, and WebP. If you want to explicitly bypass Image Optimization for a given animated image, use the unoptimized prop.

Related

For an overview of the Image component features and usage guidelines, see:

Images Learn how to display and optimize images with the Image component.