



## Next.js Runtime

[!NOTE]

Next.js Runtime v5 is available for early access! Please note that this repository is for the current version (v4) of the Next.js Runtime, and the early access version is not yet available in this repository. [Learn more.](#)

Next.js is supported natively on Netlify, and in most cases you will not need to install or configure anything. This repo includes the packages used to support Next.js on Netlify.

## Deploying

If you build on Netlify, the Next.js Runtime will work with no additional configuration. However if you are building and deploying locally using the Netlify CLI, you must deploy using `netlify deploy --build`. Running the build and deploy commands separately will not work, because the Next.js Runtime will not generate the required configuration.

## Using `next/image`

If you use `next/image`, your images will be automatically optimized at runtime, ensuring that they are served at the best size and format. The image will be processed on the first request which means it may take longer to load, but the generated image is then cached and served as a static file to future visitors. By default, Next.js will deliver WebP images if the browser supports it. WebP is a modern image format with wide browser support that will usually generate smaller files than PNG or JPG. Additionally, you can enable AVIF format, which is often even smaller in filesize than WebP. The drawback is that with particularly large images, AVIF images may take too long to generate, and the function times-out. You can configure the supported image formats in your `next.config.js` file.

---

## Enabling Edge Images

It is possible to run image content negotiation on the edge. This allows images to be processed on the first request, and then, in future loads, served from cache on the edge.

In order to deliver the correct format to a visitor's browser, this uses a Netlify Edge Function. In some cases your site may not support Edge Functions, in which case it will instead fall back to delivering the original file format.

To turn on Edge image handling for Next/Image, set the environment variable `NEXT_FORCE_EDGE_IMAGES` to **true**

## Returning custom response headers on images handled by ipx

Should you wish to return custom response headers on images handled by the `netlify-ipx` package, you can add them within your project's `netlify.toml` by targeting the `/_next/image/*` route:

```
1 [[headers]]
2   for = "/_next/image/*"
3
4   [headers.values]
5     Strict-Transport-Security = "max-age=31536000"
6     X-Test = 'foobar'
```

## Disabling included image loader

If you wish to disable the use of the image loader which is bundled into the runtime by default, set the `DISABLE_IPX` environment variable to **true**.

This should only be done if the site is not using `next/image` or is using a different loader (such as Cloudinary or Imgix).

See the Next.js documentation for image loader options.

## Next.js Middleware on Netlify

Next.js Middleware works out of the box on Netlify. By default, middleware runs using Netlify Edge Functions. For legacy support for running Middleware at the origin, set the environment variable `NEXT_DISABLE_NETLIFY_EDGE` to **true**. Be aware that this will result in slower performance, as all pages that match middleware must use SSR.

For more details on Next.js Middleware with Netlify, see the middleware docs.

---

## Limitations

Due to how the site configuration is handled when it's run using Netlify Edge Functions, data such as `locale` and `defaultLocale` will be missing on the `req.nextUrl` object when running `netlify dev`.

However, this data is available on `req.nextUrl` in a production environment.

## Monorepos

If you are using a monorepo you will need to change `publish` to point to the full path to the built `.next` directory, which may be in a subdirectory. If you have changed your `distDir` then it will need to match that.

If you are using Nx, then you will need to point `publish` to the folder inside `dist`, e.g. `dist/apps/myapp/.next`.

## Incremental Static Regeneration (ISR)

The Next.js Runtime fully supports ISR on Netlify. For more details see the ISR docs.

Note that Netlify has a minimum TTL of 60 seconds for revalidation.

## Disable Static 404 on Dynamic Routes with `fallback:false`

Currently when hitting a non-prerendered path with `fallback=false` it will default to a 404 page. You can now change this default setting by using the environment variable `LEGACY_FALLBACK_FALSE=true`. With the environment variable set, those non-prerendered paths will now be routed through using the ISR Handler and will allow you to add redirects for those non-prerendered paths.

## Use with `next export`

If you are using `next export` to generate a static site, you do not need most of the functionality of this Next.js Runtime and you can remove it. Alternatively you can set the environment variable `NETLIFY_NEXT_PLUGIN_SKIP` to `true` and the Next.js Runtime will handle caching but won't generate any functions for SSR support. See [demos/next-export](#) for an example.

---

## Asset optimization

Netlify asset optimization should not be used with Next.js sites. Assets are already optimized by Next.js at build time, and doing further optimization can break your site. Ensure that it is not enabled at **Site settings > Build & deploy > Post processing > Asset optimization**.

## Generated functions

The Next.js Runtime works by generating three Netlify functions that handle requests that haven't been pre-rendered. These are `___netlify-handler` (for SSR and API routes), `___netlify-odb-handler` (for ISR and fallback routes), and `_ipx` (for images). You can see the requests for these in the function logs. For ISR and fallback routes you will not see any requests that are served from the edge cache, just actual rendering requests. These are all internal functions, so you won't find them in your site's own functions directory.

The Next.js Runtime will also generate a Netlify Edge Function called 'ipx' to handle image content negotiation, and if Edge runtime or middleware is enabled it will also generate edge functions for middleware and edge routes.

## Manually installing the Next.js Runtime

The Next.js Runtime installs automatically for new Next.js sites on Netlify. You can also install it manually in the following ways:

### From the UI (Recommended):

You can go to the UI and choose the site to install the Next.js Runtime on. This method is recommended because you will benefit from auto-upgrades to important fixes and feature updates.

### From npm:

```
1 npm install -D @netlify/plugin-nextjs
```

...then add the following to your `netlify.toml` file:

```
1 [[plugins]]
2   package = "@netlify/plugin-nextjs"
```

This method is recommended if you wish to pin the Next.js Runtime to a specific version.

---

## Manually upgrading from an older version of the Next.js Runtime

If you previously set these values, they're no longer needed and should be removed:

- `distDir` in your `next.config.js`
- `node_bundler = "esbuild"` in `netlify.toml`
- `external_node_modules` in `netlify.toml`
- The environment variable `NEXT_USE_NETLIFY_EDGE` can be removed as this is now the default

The `serverless` and `experimental-serverless-trace` targets are deprecated in Next.js 12, and all builds with this Next.js Runtime will now use the default `server` target. If you previously set the target in your `next.config.js`, you should remove it.

If you currently use redirects or rewrites on your site, see the Rewrites and Redirects guide for information on changes to how they are handled in this version. In particular, note that `_redirects` and `_headers` files must be placed in `public`, not in the root of the site.

## Using with pnpm

If your site uses pnpm to manage dependencies, currently you must enable public hoisting. The simplest way to do this is to create a `.npmrc` file in the root of your project with the content:

```
1 public-hoist-pattern[]=*
```

## Running the tests

To run the tests, ensure that the dependencies are installed as follows:

```
1 npm install
```

Then run the tests:

```
1 npm test
```

## End-to-end tests

In order to run the end-to-end (E2E) tests, you'll need to be logged in to Netlify. You can do this using the Netlify CLI with the command:

```
1 netlify login
```

---

Alternatively, you can set an environment variable `NETLIFY_AUTH_TOKEN` to a valid Netlify personal access token. This can be obtained from the Netlify UI.

Then run the E2E tests if logged in:

```
1 npm run test:next
```

Or if using an access token:

```
1 NETLIFY_AUTH_TOKEN=your-token-here npm run test:next
```

*Note: The E2E tests will be deployed to a Netlify owned site. To deploy to your own site then set the environment variable `NETLIFY_SITE_ID` to your site ID.*

## Feedback

If you think you have found a bug in Next.js on Netlify, please open an issue. If you have comments or feature requests, see the discussion board