output

During a build, Next.js will automatically trace each page and its dependencies to determine all of the files that are needed for deploying a production version of your application.

This feature helps reduce the size of deployments drastically. Previously, when deploying with Docker you would need to have all files from your package's `dependencies` installed to run `next start`. Starting with Next.js 12, you can leverage Output File Tracing in the `.next/` directory to only include the necessary files.

Furthermore, this removes the need for the deprecated `serverless` target which can cause various issues and also creates unnecessary duplication.

How it Works

During `next build`, Next.js will use `@vercel/nft` ↗ to statically analyze `import`, `require`, and `fs` usage to determine all files that a page might load.

Next.js' production server is also traced for its needed files and output at `.next/next-server.js.nft.json` which can be leveraged in production.

To leverage the `.nft.json` files emitted to the `.next` output directory, you can read the list of files in each trace that are relative to the `.nft.json` file and then copy them to your deployment location.

Automatically Copying Traced Files

Next.js can automatically create a `standalone` folder that copies only the necessary files for a production deployment including select files in `node_modules`.

To leverage this automatic copying you can enable it in your `next.config.js`:

next.config.js

```
module.exports = {
  output: 'standalone',
};
```

This will create a folder at `.next/standalone` which can then be deployed on its own without installing `node_modules`.

Additionally, a minimal `server.js` file is also output which can be used instead of `next start`. This minimal server does not copy the `public` or `.next/static` folders by default as these should ideally be handled by a CDN instead, although these folders can be copied to the `standalone/public` and `standalone/.next/static` folders manually, after which `server.js` file will serve these automatically.

Note: `next.config.js` is read during `next build` and serialized into the `server.js` output file. If the legacy `serverRuntimeConfig` or `publicRuntimeConfig` options are being used, the values will be specific to values at build time.

Note: If your project uses Image Optimization with the default `loader`, you must install `sharp` as a dependency:

Terminal

```
npm i sharp
```

Terminal

```
yarn add sharp
```

Terminal

```
pnpm add sharp
```

Caveats

- While tracing in monorepo setups, the project directory is used for tracing by default. For `next build packages/web-app`, `packages/web-app` would be the tracing root and any files outside of that folder

will not be included. To include files outside of this folder you can set `experimental.outputFileTracingRoot` in your `next.config.js`.

packages/web-app/next.config.js

```
module.exports = {
  experimental: {
    // this includes files from the monorepo base two directories up
    outputFileTracingRoot: path.join(__dirname, '../../'),
  },
};
```

- There are some cases in which Next.js might fail to include required files, or might incorrectly include unused files. In those cases, you can leverage `experimental.outputFileTracingExcludes` and `experimental.outputFileTracingIncludes` respectively in `next.config.js`. Each config accepts an object with minimatch globs ↗ for the key to match specific pages and a value of an array with globs relative to the project's root to either include or exclude in the trace.

next.config.js

```
module.exports = {
  experimental: {
    outputFileTracingExcludes: {
      '/api/hello': ['./un-necessary-folder/**/*'],
    },
    outputFileTracingIncludes: {
      '/api/another': ['./necessary-folder/**/*'],
    },
  },
};
```

- Currently, Next.js does not do anything with the emitted `.nft.json` files. The files must be read by your deployment platform, for example Vercel ↗, to create a minimal deployment. In a future release, a new command is planned to utilize these `.nft.json` files.

Experimental `turbotrace`

Tracing dependencies can be slow because it requires very complex computations and analysis. We created `turbotrace` in Rust as a faster and smarter alternative to the JavaScript implementation.

To enable it, you can add the following configuration to your `next.config.js`:

next.config.js

```
module.exports = {
  experimental: {
    turbotrace: {
      // control the log level of the turbotrace, default is `error`
      logLevel?:
      | 'bug'
      | 'fatal'
      | 'error'
      | 'warning'
      | 'hint'
      | 'note'
      | 'suggestions'
      | 'info',
      // control if the log of turbotrace should contain the details of the analysis, def
      logDetail?: boolean
      // show all log messages without limit
      // turbotrace only show 1 log message for each categories by default
      logAll?: boolean
      // control the context directory of the turbotrace
      // files outside of the context directory will not be traced
      // set the `experimental.outputFileTracingRoot` has the same effect
      // if the `experimental.outputFileTracingRoot` and this option are both set, the `e
      contextDirectory?: string
      // if there is `process.cwd()` expression in your code, you can set this option to
      // for example the require(process.cwd() + '/package.json') will be traced as requi
      processCwd?: string
      // control the maximum memory usage of the `turbotrace`, in `MB`, default is `6000`
      memoryLimit?: number
    },
  },
}
```