Next.js Docs | Next.js

Welcome to the Next.js documentation!

Source: https://nextjs.org/docs

Generated: 2026-08-11 v0.8.10-9-g59efd08

How to Read This Book

This reference book was generated from the publicly accessible documentation at `https://nextjs.org/docs` on 2026-08-11. It is a structured, self-contained snapshot suitable for offline reference and AI ingestion.

Table of Contents

Api

All documentation pages categorized under api

- htmlLimitedBots
- Custom Next.js Cache Handler
- next.config.js: inlineCss | Next.js
- next.config.js: instrumentationClientInject | Next.js
- next.config.js: logging
- next.config.js: onDemandEntries | Next.js
- optimizePackageImports
- next.config.js: pageExtensions | Next.js
- partialPrefetching
- prefetchInlining
- productionBrowserSourceMaps
- next.config.js: proxyClientMaxBodySize
- reactCompiler
- reactMaxHeadersLength
- next.config.js: sassOptions | Next.js
- serverActions
- serverComponentsHmrCache
- staleTimes
- next.config.js: staticGeneration*
- next.config.js: supportsImmutableAssets | Next.js
- transpilePackages
- turbopack
- turbopackChunking
- Turbopack FileSystem Caching
- turbopack.ignoreIssue
- next.config.js: turbopackMemoryEviction | Next.js
- turbopackRustReactCompiler
- typedRoutes
- typescript
- useLightningcss
- next.config.js: useTypeScriptCli | Next.js
- Custom Webpack Config
- use cache: private

Guide

All documentation pages categorized under guide

- Creating an Adapter
- Implementing PPR in an Adapter
- Testing Adapters
- Adapters: Use Cases | Next.js

Overview

All documentation pages categorized under overview

- Next.js Docs
- App Router
- API Reference
- Adapters
- Components
- Configuration

Reference

All documentation pages categorized under reference

- Adapters: Supporting Immutable Static Assets | Next.js
- Adapters: Invoking Entrypoints | Next.js
- API Reference: CLI | Next.js
- Components: Image Component | Next.js
- ESLint Plugin
- next.config.js
- basePath
- next.config.js: cssChunking
- next.config.js: deploymentId
- distDir
- next.config.js: exportPathMap | Next.js
- httpAgentOptions
- next.config.js: images
- mdxRs
- next.config.js: output | Next.js
- outputHashSalt
- next.config.js: poweredByHeader

- next.config.js: reactStrictMode | Next.js
- next.config.js: redirects | Next.js
- next.config.js: rewrites | Next.js
- serverExternalPackages
- next.config.js: taint | Next.js
- trailingSlash
- turbopackLocalPostcssConfig
- next.config.js: urlImports | Next.js
- next.config.js: useOffline | Next.js
- webVitalsAttribution
- Configuration: TypeScript | Next.js
- Directives
- Directives: use cache | Next.js

Api

API Reference

`API`

Source: https://nextjs.org/docs/app/api-reference/adapters/api-reference

This page describes the Next.js adapter API, including functions to modify configuration and handle build completion.

`async modifyConfig(config, context)`

Called for any CLI command that loads the `next.config.js` file to allow modification of the configuration.

Parameters:

- `config`: The complete Next.js configuration object
- `context.phase`:...

`async onBuildComplete(context)`

Called after the build process completes with detailed information about routes and outputs.

Parameters:

- `context.routing`: Object containing Next.js routing phases and metadata -...

Parameters

Name	Type	Description	Default	Required
`config`	object	The complete Next.js configuration object		Yes
`context.phase`	string	The current build phase (see phases)		Yes
`context.nextVersion`	string	Version of Next.js being used		Yes
`context.projectDir`	string	Absolute path to the Next.js project directory		Yes
`context.routing`	object	Object containing Next.js routing phases and metadata		Yes
`context.routing.beforeMiddleware`	any	Routes executed before middleware (includes header and redirect handling)		Yes
`context.routing.beforeFiles`	any	Rewrite routes checked before filesystem route matching		Yes
`context.routing.afterFiles`	any	Rewrite routes checked after filesystem route matching		Yes
`context.routing.dynamicRoutes`	any	Dynamic route matching table		Yes
`context.routing.onMatch`	any	Routes applied after a successful match (for example immutable static asset cache headers)		Yes
`context.routing.fallback`	any	Final rewrite fallback routes		Yes
`context.routing.shouldNormalizeNextData`	boolean	Whether `/_next/data/<buildId>/...` URLs should be normalized during matching		Yes

`context.routing.rsc`	any	Route metadata used for React Server Components routing behavior	Yes
`context.outputs`	object	Detailed information about all build outputs organized by type	Yes
`context.repoRoot`	string	Absolute path to the detected repository root	Yes
`context.distDir`	string	Absolute path to the build output directory	Yes
`context.config`	object	The final Next.js configuration (with modifyConfig applied)	Yes
`context.buildId`	string	Unique identifier for the current build	Yes

Configuration

`API`

Source: https://nextjs.org/docs/app/api-reference/adapters/configuration

This page describes how to configure Next.js adapters by specifying the adapter module path via `adapterPath` in `next.config.js` or the `NEXT_ADAPTER_PATH` environment variable.

Configuration

To use an adapter, specify the path to your adapter module in `adapterPath`:

Alternatively `NEXT_ADAPTER_PATH` can be set to enable zero-config usage in deployment platforms.

next.config.js

```
/** @type {import('next').NextConfig} */
const nextConfig = {
  adapterPath: require.resolve('./my-adapter.js'),
}

module.exports = nextConfig
```

Parameters

Name	Type	Description	Default	Required
`adapterPath`	string	Path to the adapter module. Specify the path to your adapter module in `adapterPath`.		No
`NEXT_ADAPTER_PATH`	string	Environment variable that can be set to enable zero-config usage in deployment platforms.		No

Adapters: Output Types

`API`

Source: https://nextjs.org/docs/app/api-reference/adapters/output-types

Describes the `outputs` object structure exposed to Next.js adapters, including arrays for pages, API routes, app pages, app routes, prerenders, static files, and middleware, with TypeScript type...

Output Types

The `outputs` object contains arrays of build output types:

- `outputs.pages`: React pages from the `pages/` directory
- `outputs.pagesApi`: API routes from `pages/api/`
- `outputs.appPages`: React...

Pages (`outputs.pages`)

React pages from the `pages/` directory:

```typescript
{
  type: 'PAGES'
  id: string           // Route identifier
  filePath: string     // Path to the built file
  pathname: string     // URL pathname
  sourcePage: string   // Original source file…
```

API Routes (`outputs.pagesApi`)

API routes from `pages/api/`:

```typescript
{
  type: 'PAGES_API'
  id: string           // Route identifier
  filePath: string     // Path to the built file
  pathname: string     // URL pathname
  sourcePage: string   // Original relative…
```

App Pages (`outputs.appPages`)

React pages from the `app/` directory:

```typescript
{
  type: 'APP_PAGE'
  id: string           // Route identifier
  filePath: string     // Path to the built file
  pathname: string     // URL pathname. Includes .rsc suffix for RSC routes…
```

App Routes (`outputs.appRoutes`)

API and metadata routes from the `app/` directory:

```typescript
{
  type: 'APP_ROUTE'
  id: string           // Route identifier
  filePath: string     // Path to the built file
  pathname: string     // URL pathname
  sourcePage: string   // Original relative…
```

Prerenders (`outputs.prerenders`)

ISR-enabled routes and static prerenders:

```typescript
{
  type: 'PRERENDER'
  id: string           // Route identifier
  pathname: string     // URL pathname
  parentOutputId: string  // ID of the source page/route
  groupId: number        //…
```

Prerender classification

`routeType`, `response`, and `compute` are emitted together on the primary response in a prerender group. Related RSC, data, and segment outputs omit these fields. Pages Router templates with…

Static Files (`outputs.staticFiles`)

Static assets and auto-statically optimized pages:

See Supporting immutable static assets for more information about `immutableHash`.

```typescript
{
  type: 'STATIC_FILE'
  id: string // Unique identifier for this static file output
  filePath: string // Absolute filesystem path to the built file
  pathname: string // The routable URL pathname…
```

Middleware (`outputs.middleware`)

`middleware.ts` (`.js` / `.ts`) or `proxy.ts` (`.js` / `.ts`) function (if present):

```typescript
{
  type: 'MIDDLEWARE'
  id: string           // Route identifier
  filePath: string     // Path to the built file
  pathname: string      // Always '/_middleware'
  sourcePage: string    // Always…
```

Parameters

Name	Type	Description	Default	Required
`outputs.pages`	Object[]	React pages from the `pages/` directory		Yes
`outputs.pagesApi`	Object[]	API routes from `pages/api/`		Yes
`outputs.appPages`	Object[]	React pages from the `app/` directory		Yes
`outputs.appRoutes`	Object[]	API and metadata routes from `app/`		Yes
`outputs.prerenders`	Object[]	ISR-enabled routes and static prerenders		Yes
`outputs.staticFiles`	Object[]	Static assets and auto-statically optimized pages		Yes
`outputs.middleware`	Object[]	Middleware function (if present)		Yes

Adapters: Routing Information

`API`

Source: https://nextjs.org/docs/app/api-reference/adapters/routing-information

Describes the `routing` object available in `onBuildComplete`, including the route phases and common fields that make up deployment-ready routing information.

Routing Information

The `routing` object in `onBuildComplete` provides complete routing information with processed patterns ready for deployment:

routing.beforeMiddleware

Routes applied before middleware execution. These include generated header and redirect behavior.

routing.beforeFiles

Rewrite routes checked before filesystem route matching.

routing.afterFiles

Rewrite routes checked after filesystem route matching.

routing.dynamicRoutes

Dynamic matchers generated from route segments such as `[slug]` and catch-all routes.

routing.onMatch

Routes that apply after a successful match, such as immutable cache headers for hashed static assets.

routing.fallback

Final rewrite routes checked when earlier phases did not produce a match.

Common Route Fields

Each route entry can include:

- `source`: Original route pattern (optional for generated internal rules)
- `sourceRegex`: Compiled regex for matching requests
- `destination`: Internal destination...

Routing with @next/routing

`API`

Source: https://nextjs.org/docs/app/api-reference/adapters/routing-with-next-routing

Explains how to use the `@next/routing` package's `resolveRoutes()` function to reproduce Next.js route matching behavior with data from `onBuildComplete`.

Routing with @next/routing

You can use `@next/routing` to reproduce Next.js route matching behavior with data from `onBuildComplete`.

`resolveRoutes()` returns:

- `middlewareResponded`: `true` when middleware already sent a...

```javascript
import { resolveRoutes } from '@next/routing'

const pathnames = [
  ...outputs.pages,
  ...outputs.pagesApi,
  ...outputs.appPages,
  ...outputs.appRoutes,
  ...outputs.staticFiles,
].map((output)…
```

Parameters

Name	Type	Description	Default	Required
`url`	URL	The request URL as a URL object.		Yes
`buildId`	string	The build ID for the deployment.		Yes
`basePath`	string	The base path of the Next.js application.	''	No
`i18n`	object	The i18n configuration object.		No
`headers`	Headers	The request headers as a Headers object.		Yes
`requestBody`	ReadableStream	The request body as a ReadableStream.		No
`pathnames`	string[]	An array of pathnames from the build output.		Yes
`routes`	object	The routing configuration from the build.		Yes
`invokeMiddleware`	async function	An async function to invoke middleware, returning a response object.		Yes

Runtime Integration

`API`

Source: https://nextjs.org/docs/app/api-reference/adapters/runtime-integration

Describes the runtime behavior of Next.js server and cache interfaces, and how adapters interact with them, including handler context and PPR chain headers.

Overview

The Deployment Adapter API is a **build-time** interface. It tells your platform what was built and how to route requests. **Runtime** behavior (request handling, streaming, caching) is handled by…

Handler Context

When invoking entrypoints, adapters pass a `ctx` object to the Next.js handler. Key fields include:

- `ctx.waitUntil`: a function that accepts a promise. Use this to keep the serverless function…

PPR Chain Headers

In the prerenders output type, `pprChain.headers` contains the headers needed for the [resume…

create-next-app

`API`

Source: https://nextjs.org/docs/app/api-reference/cli/create-next-app

The create-next-app CLI allows you to create a new Next.js application using the default template or an example from a public GitHub repository. It is the easiest way to get started with Next.js.

Basic usage

```bash
pnpm create next-app [project-name] [options]
```

Reference

The following options are available:

Examples

With the default template

To create a new app using the default template, run the following command in your terminal:

On installation, you'll see the following prompts:

If you choose to `customize settings`, you'll see the…

```bash
pnpm create next-app
```

```text
What is your project named? my-app
Would you like to use the recommended Next.js defaults?
    Yes, use recommended defaults - TypeScript, ESLint, Tailwind CSS, App Router, AGENTS.md
    No, reuse…
```

```text
Would you like to use TypeScript? No / Yes
Which linter would you like to use? ESLint / Biome / None
Would you like to use React Compiler? No / Yes
Would you like to use Tailwind CSS? No / Yes
Would…
```

Linter Options

ESLint : The traditional and most popular JavaScript linter. Includes Next.js-specific rules from `@next/eslint-plugin-next`.

Biome : A fast, modern linter and formatter that combines the…

With an official Next.js example

To create a new app using an official Next.js example, use the `--example` flag. For example:

You can view a list of all available examples along with setup instructions in the [Next.js…

```bash
pnpm create next-app --example [example-name] [your-project-name]
```

With any public GitHub example

To create a new app using any public GitHub example, use the `--example` option with the GitHub repository's URL. For example:

```bash
pnpm create next-app --example "https://github.com/.../" [your-project-name]
```

Parameters

Name	Type	Description	Default	Required
`-h or --help`	boolean	Show all available options		No
`-v or --version`	boolean	Output the version number		No
`--no-*`	boolean	Negate default options. E.g. --no-ts		No
`--ts or --typescript`	boolean	Initialize as a TypeScript project	True	No
`--js or --javascript`	boolean	Initialize as a JavaScript project		No
`--tailwind`	boolean	Initialize with Tailwind CSS config	True	No
`--react-compiler`	boolean	Initialize with React Compiler enabled		No
`--eslint`	boolean	Initialize with ESLint config		No
`--biome`	boolean	Initialize with Biome config		No
`--no-linter`	boolean	Skip linter configuration		No
`--app`	boolean	Initialize as an App Router project		No
`--api`	boolean	Initialize a project with only route handlers		No
`--src-dir`	boolean	Initialize inside a src/ directory		No
`--turbopack`	boolean	Force enable Turbopack in generated package.json	True	No
`--webpack`	boolean	Force enable Webpack in generated package.json		No
`--import-alias <alias-to-configure>`	string	Specify import alias to use	@/*	No
`--empty`	boolean	Initialize an empty project		No
`--use-npm`	boolean	Explicitly tell the CLI to bootstrap the application		No

		using npm		
`--use-pnpm`	boolean	Explicitly tell the CLI to bootstrap the application using pnpm		No
`--use-yarn`	boolean	Explicitly tell the CLI to bootstrap the application using Yarn		No
`--use-bun`	boolean	Explicitly tell the CLI to bootstrap the application using Bun		No
`-e or --example [name] [github-url]`	string	An example to bootstrap the app with		No
`--example-path <path-to-example>`	string	Specify the path to the example separately		No
`--reset-preferences`	boolean	Explicitly tell the CLI to reset any stored preferences		No
`--skip-install`	boolean	Explicitly tell the CLI to skip installing packages		No
`--disable-git`	boolean	Explicitly tell the CLI to disable git initialization		No
`--agents-md`	boolean	Include AGENTS.md and CLAUDE.md to guide coding agents	True	No
`--yes`	boolean	Use previous preferences or defaults for all options		No

next CLI

API

Source: https://nextjs.org/docs/app/api-reference/cli/next

The Next.js CLI allows you to develop, build, start your application, and more. It provides commands like dev, build, start, info, telemetry, typegen, upgrade, and experimental-analyze.

Basic Usage

The Next.js CLI allows you to develop, build, start your application, and more. Basic usage:

> ***Good to know** : With `npm run`, use `--` before CLI flags so npm forwards them to `next`. This is...*

terminal

```
pnpm next [command] [options]
```

Reference

The following options are available:

Options	Description
`-h` or `--help`	Shows all available options
`-v` or `--version`	Outputs the Next.js version number

Commands

The following commands are available:

Command	Description
`dev`	Starts Next.js in development mode with Hot Module Reloading, error reporting, and more....

next dev options

`next dev` starts the application in development mode with Hot Module Reloading (HMR), error reporting, and more.

> ***Good to know** : Development builds output to `.next/dev` instead of `.next`....*

next build options

`next build` creates an optimized production build of your application. The output displays information about each route. For example:

```
Route (app)
┌ ○ /_not-found
└ ƒ /products/[id]

○…
```

terminal

```
Route (app)
┌ ○ /_not-found
└ ƒ /products/[id]

○  (Static)   prerendered as static content
ƒ  (Dynamic)  server-rendered on demand
```

next start options

`next start` starts the application in production mode. The application should be compiled with `next build` first.

The following options are available for the `next start`...

next info options

`next info` prints relevant details about the current system which can be used to report Next.js bugs when opening a GitHub issue. This information...

terminal

```
Operating System:
  Platform: darwin
  Arch: arm64
  Version: Darwin Kernel Version 23.6.0
  Available memory (MB): 65536
  Available CPU cores: 10
Binaries:
  Node: 20.12.0
  npm: 10.5.0
  Yarn:…
```

next telemetry options

Next.js collects **completely anonymous** telemetry data about general usage. Participation in this anonymous program is optional, and you can opt-out if you prefer not to share information.

The...

next typegen options

`next typegen` generates TypeScript definitions for your application's routes without performing a full build. This is useful for IDE autocomplete and CI type-checking of route usage.

Previously,...

terminal

```
# Generate route types first, then validate with TypeScript
next typegen && tsc --noEmit

# Or in CI workflows for type checking without building
next typegen && npm run type-check
```

terminal

```
next typegen
# or for a specific app
next typegen ./apps/web
```

next upgrade options

`next upgrade` upgrades your Next.js application to the latest version.

The following options are available for the `next upgrade` command:

Option	Description
`-h, --help`	...

next experimental-analyze options

`next experimental-analyze` analyzes your application's bundle output using Turbopack. This command helps you understand the size and composition of your bundles,...

terminal

```
pnpm next experimental-analyze
```

terminal

```
# Write output to .next/diagnostics/analyze
npx next experimental-analyze --output

# Copy the output for comparison with a future analysis
cp -r .next/diagnostics/analyze ./analyze-before-refactor
```

Debugging prerender errors

If you encounter prerendering errors during `next build`, you can pass the `--debug-prerender` flag to get more detailed output:

```
next build --debug-prerender
```

This enables several...

terminal

```
next build --debug-prerender
```

Building specific routes

You can build only specific routes in the App and Pages Routers using the `--debug-build-paths` option. This is useful for faster debugging when working with large applications. The...

terminal

```
# Build a specific route
next build --debug-build-paths="app/page.tsx"

# Build more than one route
next build --debug-build-paths="app/page.tsx,pages/index.tsx"

# Include route group folders in the…
```

Changing the default port

By default, Next.js uses `http://localhost:3000` during development and with `next start`. The default port can be changed with the `-p` option, like so:

```
next dev -p 4000
```

Or using the...

terminal

```
next dev -p 4000
```

terminal

```
PORT=4000 next dev
```

Using HTTPS during development

For certain use cases like webhooks or authentication, you can use HTTPS to have a secure environment on `localhost`. Next.js can generate a...

terminal

```
next dev --experimental-https
```

terminal

```
next dev --experimental-https --experimental-https-key ./certificates/localhost-key.pem --experimenta
```

Configuring a timeout for downstream proxies

When deploying Next.js behind a downstream proxy (e.g. a load-balancer like AWS ELB/ALB), it's important to configure Next's underlying HTTP server with [keep-alive...

terminal

```
next start --keepAliveTimeout 70000
```

Passing Node.js arguments

You can pass any node arguments to `next` commands. For example:

```
NODE_OPTIONS='--throw-deprecation' next
NODE_OPTIONS='-r esm'…
```

terminal

```
NODE_OPTIONS='--throw-deprecation' next
NODE_OPTIONS='-r esm' next
NODE_OPTIONS='--inspect' next
```

CPU profiling

You can capture CPU profiles to analyze performance bottlenecks in your Next.js application. The `--experimental-cpu-prof` flag enables V8's built-in CPU profiler and saves profiles to...

terminal

```
# Profile the build process
next build --experimental-cpu-prof

# Profile the dev server (profile saved on Ctrl+C or SIGTERM)
next dev --experimental-cpu-prof

# Profile the production server
next…
```

Version History

Version	Changes
`v16.1.0`	Add the `next upgrade` command
`v16.1.0`	Add the `next experimental-analyze` command
`v16.0.0`	The JS bundle size metrics have been...

Parameters

Name	Type	Description	Default	Required
`global -h, --help`	boolean	Shows all available options		No
`global -v, --version`	boolean	Outputs the Next.js version number		No
`dev -h, --help`	boolean	Show all available options.		No
`dev [directory]`	string	A directory in which to build the application. If not provided, current directory is used.		No
`dev --turbopack`	boolean	Force enable Turbopack (enabled by default). Also available as --turbo.		No
`dev --webpack`	boolean	Use Webpack instead of the default Turbopack bundler for development.		No
`dev -p, --port`	string	Specify a port number on which to start the application. Default: 3000, env: PORT	3000	No
`dev -H, --hostname`	string	Specify a hostname on which to start the application. Useful for making the application available for other devices on the network. Default: 0.0.0.0	0.0.0.0	No
`dev --experimental-https`	boolean	Starts the server with HTTPS and generates a self-signed certificate.		No
`dev --experimental-https-key`	string	Path to a HTTPS key file.		No
`dev --experimental-https-cert`	string	Path to a HTTPS certificate file.		No
`dev --experimental-https-ca`	string	Path to a HTTPS certificate authority file.		No
`dev --experimental-upload-trace`	string	Reports a subset of the debugging trace to a remote HTTP URL.		No

`dev --experimental-cpu-prof`	boolean	Enables CPU profiling using V8's inspector. Profiles are saved to .next-profiles/ on exit.		No
`build -h, --help`	boolean	Show all available options.		No
`build [directory]`	string	A directory on which to build the application. If not provided, the current directory will be used.		No
`build --turbopack`	boolean	Force enable Turbopack (enabled by default). Also available as --turbo.		No
`build --webpack`	boolean	Build using Webpack.		No
`build -d, --debug`	boolean	Enables a more verbose build output. With this flag enabled additional build output like rewrites, redirects, and headers will be shown.		No
`build --profile`	boolean	Enables production profiling for React.		No
`build --no-lint`	boolean	Disables linting. Note: linting will be removed from next build in Next 16. If you're using Next 15.5+ with a linter other than eslint, linting during build will not occur.		No
`build --no-mangling`	boolean	Disables mangling. This may affect performance and should only be used for debugging purposes.		No
`build --experimental-app-only`	boolean	Builds only App Router routes.		No
`build --experimental-build-mode`	string	Uses an experimental build mode. (choices: "compile", "generate", default: "default")	default	No
`build --debug-prerender`	boolean	Debug prerender errors in development.		No

`build --debug-build-paths`	string	Build only specific routes for debugging.		No
`build --experimental-cpu-prof`	boolean	Enables CPU profiling using V8's inspector. Profiles are saved to .next-profiles/ on exit.		No
`start -h, --help`	boolean	Show all available options.		No
`start [directory]`	string	A directory on which to start the application. If no directory is provided, the current directory will be used.		No
`start -p, --port`	string	Specify a port number on which to start the application. (default: 3000, env: PORT)	3000	No
`start -H, --hostname`	string	Specify a hostname on which to start the application (default: 0.0.0.0).	0.0.0.0	No
`start --keepAliveTimeout`	string	Specify the maximum amount of milliseconds to wait before closing the inactive connections.		No
`start --experimental-cpu-prof`	boolean	Enables CPU profiling using V8's inspector. Profiles are saved to .next-profiles/ on exit.		No
`info -h, --help`	boolean	Show all available options		No
`info --verbose`	boolean	Collects additional information for debugging.		No
`telemetry -h, --help`	boolean	Show all available options.		No
`telemetry --enable`	boolean	Enables Next.js' telemetry collection.		No
`telemetry --disable`	boolean	Disables Next.js' telemetry collection.		No
`typegen -h, --help`	boolean	Show all available options.		No

`typegen [directory]`	string	A directory on which to generate types. If not provided, the current directory will be used.		No
`upgrade -h, --help`	boolean	Show all available options.		No
`upgrade [directory]`	string	A directory with the Next.js application to upgrade. If not provided, the current directory will be used.		No
`upgrade --revision`	string	Specify a Next.js version or tag to upgrade to (e.g., latest, canary, 15.0.0). Defaults to the release channel you have currently installed.		No
`upgrade --verbose`	boolean	Show verbose output during the upgrade process.		No
`experimental-analyze -h, --help`	boolean	Show all available options.		No
`experimental-analyze [directory]`	string	A directory on which to analyze the application. If not provided, the current directory will be used.		No
`experimental-analyze --no-mangling`	boolean	Disables mangling. This may affect performance and should only be used for debugging purposes.		No
`experimental-analyze --profile`	boolean	Enables production profiling for React. This may affect performance.		No
`experimental-analyze -o, --output`	boolean	Write analysis files to disk without starting the server. Output is written to .next/diagnostics/analyze.		No
`experimental-analyze --port`	string	Specify a port number to serve the analyzer on. (default: 4000, env: PORT)	4000	No

Font Module

API

Source: https://nextjs.org/docs/app/api-reference/components/font

This page documents the Next.js Font Module (`next/font`), which automatically optimizes fonts, self-hosts font files, and provides built-in support for Google Fonts. It covers the API reference for...

Reference

The following table lists the available options for the font loader functions. The columns indicate which loader (`font/google` or `font/local`) each option applies to, the type, and whether it is...

src

The path of the font file as a string or an array of objects (with type `Array<{path: string, weight?: string, style?: string}>`) relative to the directory where the font loader function is...

weight

The font `weight` with the following possibilities:

- A string with possible values of the weights available for the specific font or a range of...

style

The font `style` with the following possibilities:

- A string value with...

subsets

The font `subsets` defined by an array of string values with the names of each subset you would like to be...

axes

Some variable fonts have extra `axes` that can be included. By default, only the font weight is included to keep the file size down. The possible values of `axes` depend on the specific font.

Used...

display

The font `display` with possible string values of...

preload

A boolean value that specifies whether the font should be preloaded or not. The default is `true` .

Used in `next/font/google` and...

fallback

The fallback font to use if the font cannot be loaded. An array of strings of fallback fonts with no default.

- Optional

Used in `next/font/google` and `next/font/local`

Examples:

- `fallback:...

adjustFontFallback

- For `next/font/google`: A boolean value that sets whether an automatic fallback font should be used to reduce Cumulative Layout Shift. The default is `true`.
- For...

variable

A string value to define the CSS variable name to be used if the style is applied with the CSS variable method.

Used in `next/font/google` and `next/font/local`

-...

declarations

An array of font face descriptor key-value pairs that define the generated `@font-face` further.

Used in `next/font/local`

-...

Examples

Google Fonts

To use a Google font, import it from `next/font/google` as a function. We recommend using variable fonts for the best performance and flexibility.

If you...

app/layout.tsx

```
import { Inter } from 'next/font/google'

// If loading a variable font, you don't need to specify the font weight
const inter = Inter({
  subsets: ['latin'],
  display: 'swap',
})

export default…
```

app/layout.tsx

```
import { Roboto } from 'next/font/google'

const roboto = Roboto({
  weight: '400',
  subsets: ['latin'],
  display: 'swap',
})

export default function RootLayout({
  children,
}: {
  children:…
```

app/layout.js

```
const roboto = Roboto({
  weight: ['400', '700'],
  style: ['normal', 'italic'],
  subsets: ['latin'],
  display: 'swap',
})
```

Specifying a subset

Google Fonts are automatically subset. This reduces the size of the font file and improves performance. You'll need to define which of these...

app/layout.tsx

```
const inter = Inter({ subsets: ['latin'] })
```

Using Multiple Fonts

You can import and use multiple fonts in your application. There are two approaches you can take.

The first approach is to create a utility function that exports a font, imports it, and applies its...

app/fonts.ts

```
import { Inter, Roboto_Mono } from 'next/font/google'

export const inter = Inter({
  subsets: ['latin'],
  display: 'swap',
})

export const roboto_mono = Roboto_Mono({
  subsets: ['latin'],…
```

app/layout.tsx

```
import { inter } from './fonts'

export default function Layout({ children }: { children: React.ReactNode }) {
  return (
    <html lang="en" className={inter.className}>
      <body>…
```

app/page.tsx

```
import { roboto_mono } from './fonts'

export default function Page() {
  return (
    <>
      <h1 className={roboto_mono.className}>My page</h1>
    </>
  )
}
```

app/layout.tsx

```
import { Inter, Roboto_Mono } from 'next/font/google'
import styles from './global.css'

const inter = Inter({
  subsets: ['latin'],
  variable: '--font-inter',
  display: 'swap',
})

const…
```

app/global.css

```
html {
  font-family: var(--font-inter);
}

h1 {
  font-family: var(--font-roboto-mono);
}
```

Local Fonts

Import `next/font/local` and specify the `src` of your local font file. We recommend using variable fonts for the best performance and flexibility.

If you...

app/layout.tsx

```
import localFont from 'next/font/local'

// Font files can be colocated inside of `app`
const myFont = localFont({
  src: './my-font.woff2',
  display: 'swap',
})

export default function…
```

js

```
const roboto = localFont({
  src: [
    {
      path: './Roboto-Regular.woff2',
      weight: '400',
      style: 'normal',
    },
    {
      path: './Roboto-Italic.woff2',
      weight: '400',…
```

With Tailwind CSS

`next/font` integrates seamlessly with Tailwind CSS using CSS variables.

In the example below, we use the `Inter` ...

app/layout.tsx

```
import { Inter, Roboto_Mono } from 'next/font/google'

const inter = Inter({
  subsets: ['latin'],
  display: 'swap',
  variable: '--font-inter',
})

const roboto_mono = Roboto_Mono({
  subsets:…
```

global.css

```
@import 'tailwindcss';

@theme inline {
  --font-sans: var(--font-inter);
  --font-mono: var(--font-roboto-mono);
}
```

tailwind.config.js

```
/** @type {import('tailwindcss').Config} */
module.exports = {
  content: [
    './pages/**/*.{js,ts,jsx,tsx}',
    './components/**/*.{js,ts,jsx,tsx}',
    './app/**/*.{js,ts,jsx,tsx}',
  ],…
```

html

```
<p class="font-sans ...">The quick brown fox ...</p>
<p class="font-mono ...">The quick brown fox ...</p>
```

Applying Styles

You can apply the font styles in three ways:

- `className`

- `style`
- CSS Variables

className

Returns a read-only CSS `className` for the loaded font to be passed to an HTML element.

jsx

```jsx
<p className={inter.className}>Hello, Next.js!</p>
```

CSS Variables

If you would like to set your styles in an external style sheet and specify additional options there, use the CSS variable method.

In addition to importing the font, also import the CSS file where...

app/page.tsx

```tsx
import { Inter } from 'next/font/google'
import styles from '../styles/component.module.css'

const inter = Inter({
  variable: '--font-inter',
})
```

app/page.tsx

```tsx
<main className={inter.variable}>
  <p className={styles.text}>Hello World</p>
</main>
```

styles/component.module.css

```css
.text {
  font-family: var(--font-inter);
  font-weight: 200;
  font-style: italic;
}
```

Using a font definitions file

Every time you call the `localFont` or Google font function, that font will be hosted as one instance in your application. Therefore, if you need to use the same font in multiple places, you should...

styles/fonts.ts

```ts
import { Inter, Lora, Source_Sans_3 } from 'next/font/google'
import localFont from 'next/font/local'

// define your variable fonts
const inter = Inter()
const lora = Lora()
// define 2 weights of…
```

app/page.tsx

```
import { inter, lora, sourceCodePro700, greatVibes } from '../styles/fonts'

export default function Page() {
  return (
    <div>
      <p className={inter.className}>Hello world using Inter…
```

tsconfig.json

```
{
  "compilerOptions": {
    "paths": {
      "@/fonts": ["./styles/fonts"]
    }
  }
}
```

app/about/page.tsx

```
import { greatVibes, sourceCodePro400 } from '@/fonts'
```

Preloading

When a font function is called on a page of your site, it is not globally available and preloaded on all routes. Rather, the font is only preloaded on the related routes based on the type of file…

Version Changes

The following table lists the version history for the Font Module.

Parameters

Name	Type	Description	Default	Required
`src`	String or Array of Objects	The path of the font file as a string or an array of objects (with type `Array<{path: string, weight?: string, style?: string}>`) relative to the directory where the font loader function is called....		Yes
`weight`	String or Array	The font weight. A string with possible values of the weights available for the specific font or a range of values if it's a variable font. An array of weight values if the font is not a variable...		Yes
`style`	String or Array	The font style. A string value with default value of `'normal'`. An array of style values if the font is not a variable google font (applies to `next/font/google` only). Optional.	'normal'	No
`subsets`	Array of Strings	The font subsets defined by an array of string values with the names of each subset you would like to be preloaded. Fonts specified via `subsets` will have a link preload tag injected into the head...		No
`axes`	Array of Strings	Some variable fonts have extra `axes` that can be included. By default, only the font weight is included to keep the file size down. The possible values of `axes` depend on the specific font. Used in...		No
`display`	String	The font display property with possible string values of `'auto'`, `'block'`, `'swap'`, `'fallback'` or `'optional'` with default value of `'swap'`.	'swap'	No
`preload`	Boolean	A boolean value that specifies whether the font should be preloaded or not. The default is `true`.	true	No
`fallback`	Array of Strings	The fallback font to use if the font cannot be loaded. An array of strings of fallback fonts with no default.		No
`adjustFontFallback`	Boolean or String	For `next/font/google`: A boolean value that sets whether an automatic fallback font should be used to reduce Cumulative Layout Shift. The default is `true`. For `next/font/local`: A string or...	true (google) / 'Arial' (local)	No

`variable`	String	A string value to define the CSS variable name to be used if the style is applied with the CSS variable method.	No
`declarations`	Array of Objects	An array of font face descriptor key-value pairs that define the generated `@font-face` further. Used in `next/font/local`.	No

Form Component

`API`

Source: https://nextjs.org/docs/app/api-reference/components/form

The <Form> component extends the HTML <form> element to provide prefetching of loading UI, client-side navigation on submission, and progressive enhancement. It is useful for forms that update URL...

Overview

The `<Form>` component extends the HTML `<form>` element to provide **prefetching** of loading UI, **client-side navigation** on submission, and **progressive enhancement**. It's useful for forms...

`/app/ui/search.tsx`

```
import Form from 'next/form'

export default function Page() {
  return (
    <Form action="/search">
      {/* On submission, the input value will be appended to
          the URL, e.g.…
```

Reference

The behavior of the `<Form>` component depends on whether the `action` prop is passed a `string` or `function`.

- When `action` is a **string**, the `<Form>` behaves like a native HTML form that...

action (string) Props

When `action` is a string, the `<Form>` component supports the following props:

- `action` : The URL or path to navigate to when the form is submitted. An empty string `""` will navigate to the...

action (function) Props

When `action` is a function, the `<Form>` component supports the following prop:

- `action` : The Server Action to be called when the form is submitted. See the [React...

Caveats

- `formAction` : Can be used in a `<button>` or `<input type="submit">` fields to override the `action` prop. Next.js will perform a client-side navigation, however, this approach doesn't support...

Examples

Search form that leads to a search result page

You can create a search form that navigates to a search results page by passing the path as an `action`:

When the user updates the query input field and submits the form, the form data will be...

/app/page.tsx

```
import Form from 'next/form'

export default function Page() {
  return (
    <Form action="/search">
      <input name="query" />
      <button type="submit">Submit</button>
    </Form>
  )
}
```

/app/search/page.tsx

```
import { getSearchResults } from '@/lib/search'

export default async function SearchPage({
  searchParams,
}: {
  searchParams: Promise<{ [key: string]: string | string[] | undefined }>
}) {…
```

/app/search/loading.tsx

```
export default function Loading() {
  return <div>Loading...</div>
}
```

/app/ui/search-button.tsx

```
'use client'
import { useFormStatus } from 'react-dom'

export default function SearchButton() {
  const status = useFormStatus()
  return (
    <button type="submit">{status.pending ?…
```

/app/page.tsx

```
import Form from 'next/form'
import { SearchButton } from '@/ui/search-button'

export default function Page() {
  return (
    <Form action="/search">
      <input name="query" />…
```

Mutations with Server Actions

You can perform mutations by passing a function to the `action` prop.

After a mutation, it's common to redirect to the new resource. You can use the `redirect` ...

/app/posts/create/page.tsx

```
import Form from 'next/form'
import { createPost } from '@/posts/actions'

export default function Page() {
  return (
    <Form action={createPost}>
      <input name="title" />
      {/* ... */}…
```

/app/posts/actions.ts

```
'use server'
import { redirect } from 'next/navigation'

export async function createPost(formData: FormData) {
  // Create a new post
  // ...

  // Redirect to the new post…
```

/app/posts/[id]/page.tsx

```
import { getPost } from '@/posts/data'

export default async function PostPage({
  params,
}: {
  params: Promise<{ id: string }>
}) {
  const { id } = await params
  const data = await getPost(id)…
```

Parameters

Name	Type	Description	Default	Required
`action`	string | function	The URL or path to navigate to when the form is submitted (string) or the Server Action to be called when the form is submitted (function). When a string, an empty string `""` navigates to the same...		Yes
`replace`	boolean	Replaces the current history state instead of pushing a new one to the browser's history stack. Only applies when `action` is a string. Default is `false`.	false	No
`scroll`	boolean	Controls the scroll behavior during navigation. Defaults to `true`, meaning it will scroll to the top of the new route and maintain scroll position for backwards/forwards navigation. Only applies...	true	No
`prefetch`	boolean	Controls whether the path should be prefetched when the form becomes visible in the user's viewport. Only applies when `action` is a string. Defaults to `true`.	true	No

Link Component

`API`

Source: https://nextjs.org/docs/app/api-reference/components/link

This page documents the Next.js `<Link>` component, a React component that extends the HTML `<a>` element to provide prefetching and client-side navigation between routes. It covers the component's...

Link Component

`<Link>` is a React component that extends the HTML `<a>` element to provide prefetching and client-side navigation between routes. It...

app/page.tsx

```
import Link from 'next/link'

export default function Page() {
  return <Link href="/dashboard">Dashboard</Link>
}
```

Reference

The following props can be passed to the `<Link>` component:

Prop	Example	Type	Required
`href`	`href="/dashboard"`	String or Object	Yes...

href (required)

The path or URL to navigate to.

app/page.tsx

```
import Link from 'next/link'

// Navigate to /about?name=test
export default function Page() {
  return (
    <Link
      href={{
        pathname: '/about',
        query: { name: 'test' },…
```

replace

Defaults to `false`. When `true`, `next/link` will replace the current history state instead of adding a new URL into the [browser's...

app/page.tsx

```
import Link from 'next/link'

export default function Page() {
  return (
    <Link href="/dashboard" replace>
      Dashboard
    </Link>
  )
}
```

scroll

Defaults to `true`. The default scrolling behavior of `<Link>` in Next.js **is to maintain scroll position**, similar to how browsers handle back and forwards navigation. When you navigate to a...

app/page.tsx

```
import Link from 'next/link'

export default function Page() {
  return (
    <Link href="/dashboard" scroll={false}>
      Dashboard
    </Link>
  )
}
```

prefetch

Prefetching happens when a `<Link />` component enters the user's viewport (initially or through scroll). Next.js prefetches and loads the linked route (denoted by the `href`) and its data in the...

app/page.tsx

```
import Link from 'next/link'

export default function Page() {
  return (
    <Link href="/dashboard" prefetch={false}>
      Dashboard
    </Link>
  )
}
```

onNavigate

An event handler called during client-side navigation. The handler receives an event object that includes a `preventDefault()` method, allowing you to cancel the navigation if needed.

> ***Good to...*

app/page.tsx

```
import Link from 'next/link'

export default function Page() {
  return (
    <Link
      href="/dashboard"
      onNavigate={(e) => {
        // Only executes during SPA navigation…
```

transitionTypes

A list of transition types to apply to the navigation. These types are passed to `React.addTransitionType` inside the navigation transition,...

app/page.tsx

```
import Link from 'next/link'

export default function Page() {
  return (
    <Link href="/about" transitionTypes={['slide-in']}>
      About
    </Link>
  )
}
```

Examples

The following examples demonstrate how to use the `<Link>` component in different scenarios.

Linking to dynamic route segments

When linking to dynamic segments, you can use [template literals and...

app/blog/post-list.tsx

```
import Link from 'next/link'

interface Post {
  id: number
  title: string
  slug: string
}

export default function PostList({ posts }: { posts: Post[] }) {
  return (
    <ul>…
```

Checking active links

You can use `usePathname()` to check if a link is active. For example, to add a class to the active link, you can check if the current `pathname`...

app/ui/nav-links.tsx

```
'use client'

import { usePathname } from 'next/navigation'
import Link from 'next/link'

export function Links() {
  const pathname = usePathname()

  return (
    <nav>
      <Link…
```

Scrolling to an `id`

If you'd like to scroll to a specific `id` on navigation, you can append your URL with a `#` hash link or just pass a hash link to the `href` prop. This is possible since `<Link>` renders to an `<a>`...

jsx

```
<Link href="/dashboard#settings">Settings</Link>

// Output
<a href="/dashboard#settings">Settings</a>
```

Replace the URL instead of push

The default behavior of the `Link` component is to `push` a new URL into the `history` stack. You can use the `replace` prop to prevent adding a new entry, as in the following example:

app/page.js

```
import Link from 'next/link'

export default function Page() {
  return (
    <Link href="/about" replace>
      About us
    </Link>
  )
}
```

Disable scrolling to the top of the page

The default scrolling behavior of `<Link>` in Next.js **is to maintain scroll position**, similar to how browsers handle back and forwards navigation. When you navigate to a new...

app/page.tsx

```
import Link from 'next/link'

export default function Page() {
  return (
    <Link href="/#hashid" scroll={false}>
      Disables scrolling to the top
    </Link>
  )
}
```

js

```
// useRouter
import { useRouter } from 'next/navigation'

const router = useRouter()

router.push('/dashboard', { scroll: false })
```

Scroll offset with sticky headers

Because Next.js skips sticky and fixed positioned elements when finding the scroll target, content may end up behind a sticky header after navigation. For example, if your layout has a sticky...

app/layout.tsx

```
import './globals.css'

export default function RootLayout({
  children,
}: {
  children: React.ReactNode
}) {
  return (
    <html lang="en">
      <body>
        <header className="sticky top-0…
```

app/globals.css

```
html {
  scroll-padding-top: 64px; /* Match the height of your sticky header */
}
```

Prefetching links in Proxy

It's common to use Proxy for authentication or other purposes that involve rewriting the user to a different page. In order for the `<Link />` ...

proxy.ts

```
import { NextResponse } from 'next/server'

export function proxy(request: Request) {
  const nextUrl = request.nextUrl
  if (nextUrl.pathname === '/dashboard') {
    if (request.cookies.authToken)…
```

app/page.tsx

```
'use client'

import Link from 'next/link'
import useIsAuthed from './hooks/useIsAuthed' // Your auth hook

export default function Page() {
  const isAuthed = useIsAuthed()
  const path = isAuthed…
```

Blocking navigation

You can use the `onNavigate` prop to block navigation when certain conditions are met, such as when a form has unsaved changes. When you need to block navigation across multiple components in your…

app/contexts/navigation-blocker.tsx

```
'use client'

import { createContext, useState, useContext } from 'react'

interface NavigationBlockerContextType {
  isBlocked: boolean
  setIsBlocked: (isBlocked: boolean) => void
}

export…
```

app/components/form.tsx

```
'use client'

import { useNavigationBlocker } from '../contexts/navigation-blocker'

export default function Form() {
  const { setIsBlocked } = useNavigationBlocker()

  return (
    <form…
```

app/components/custom-link.tsx

```
'use client'

import Link from 'next/link'
import { useNavigationBlocker } from '../contexts/navigation-blocker'

interface CustomLinkProps extends React.ComponentProps<typeof Link> {
  children:…
```

app/components/nav.tsx

```
'use client'

import { CustomLink as Link } from './custom-link'

export default function Nav() {
  return (
    <nav>
      <Link href="/">Home</Link>
      <Link href="/about">About</Link>…
```

app/layout.tsx

```
import { NavigationBlockerProvider } from './contexts/navigation-blocker'

export default function RootLayout({
  children,
}: {
  children: React.ReactNode
}) {
  return (
    <html lang="en">…
```

app/page.tsx

```
import Nav from './components/nav'
import Form from './components/form'

export default function Page() {
  return (
    <div>
      <Nav />
      <main>
        <h1>Welcome to the Dashboard</h1>…
```

Version history

Version	Changes
`v16.2.0`	Add `transitionTypes` prop.
`v15.4.0`	Add `auto` as an alias to the default `prefetch` behavior.
`v15.3.0`	Add `onNavigate` API
…	

Parameters

Name	Type	Description	Default	Required
`href`	String or Object	The path or URL to navigate to.		Yes
`replace`	Boolean	When `true`, `next/link` will replace the current history state instead of adding a new URL into the browser's history stack.	false	No
`scroll`	Boolean	The default scrolling behavior of `<Link>` in Next.js is to maintain scroll position. When `scroll={false}`, Next.js will not attempt to scroll to the first Page element.	true	No
`prefetch`	Boolean or null	Prefetch behavior depends on whether the route is static or dynamic. When `true`, the full route is prefetched. When `false`, prefetching never happens. When `null` or `'auto'`, the default behavior...	null	No
`onNavigate`	Function	An event handler called during client-side navigation. The handler receives an event object that includes a `preventDefault()` method, allowing you to cancel the navigation if needed.		No
`transitionTypes`	string[]	A list of transition types to apply to the navigation. These types are passed to `React.addTransitionType` inside the navigation transition, enabling `<ViewTransition>` components to apply different...		No

Script Component

`API`

Source: https://nextjs.org/docs/app/api-reference/components/script

API reference for the Next.js Script component (next/script), covering its props, loading strategies, event handlers, and version history.

Script Component

This API reference will help you understand how to use props available for the Script Component. For features and usage, please see the Optimizing Scripts page.

```
app/dashboard/page.tsx
```

```
import Script from 'next/script'

export default function Dashboard() {
  return (
    <>
      <Script src="https://example.com/script.js" />
    </>
  )
}
```

Props

Here's a summary of the props available for the Script Component:

Prop	Example	Type	Required
`src`	`src="http://example.com/script"`	String	Required...

Required Props

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

src

A path string specifying the URL of an external script. This can be either an absolute external URL or an internal path. The `src` property is required unless an inline script is used.

Optional Props

The `<Script />` component accepts a number of additional properties beyond those which are required.

strategy

The loading strategy of the script. There are four different strategies that can be used:

- `beforeInteractive`: Load before any Next.js code and before any page hydration occurs. -...

beforeInteractive

Scripts that load with the `beforeInteractive` strategy are injected into the initial HTML from the server, downloaded before any Next.js module, and executed in the order they are placed.

Scripts...

```
app/layout.tsx
```

```
import Script from 'next/script'

export default function RootLayout({
  children,
}: {
  children: React.ReactNode
}) {
  return (
    <html lang="en">
      <body>
        {children}…
```

afterInteractive

Scripts that use the `afterInteractive` strategy are injected into the HTML client-side and will load after some (or all) hydration occurs on the page. **This is the default strategy** of the Script...

app/page.js

```
import Script from 'next/script'

export default function Page() {
  return (
    <>
      <Script src="https://example.com/script.js" strategy="afterInteractive" />
    </>
  )
}
```

lazyOnload

Scripts that use the `lazyOnload` strategy are injected into the HTML client-side during browser idle time and will load after all resources on the page have been fetched. This strategy should be...

app/page.js

```
import Script from 'next/script'

export default function Page() {
  return (
    <>
      <Script src="https://example.com/script.js" strategy="lazyOnload" />
    </>
  )
}
```

worker

> ***Warning:*** *The `worker` strategy is not yet stable and does not yet work with the App Router. Use with caution.*

Scripts that use the `worker` strategy are off-loaded to a web worker in order to...

next.config.js

```
module.exports = {
  experimental: {
    nextScriptWorkers: true,
  },
}
```

pages/home.tsx

```
import Script from 'next/script'

export default function Home() {
  return (
    <>
      <Script src="https://example.com/script.js" strategy="worker" />
    </>
  )
}
```

onLoad

> ***Warning:*** `onLoad` *does not yet work with Server Components and can only be used in Client Components. Further,* `onLoad` *can't be used with* `beforeInteractive` *– consider using* `onReady` *...*

app/page.tsx

```
'use client'

import Script from 'next/script'

export default function Page() {
  return (
    <>
      <Script…
```

onReady

> ***Warning:*** `onReady` *does not yet work with Server Components and can only be used in Client Components.*

Some third-party scripts require users to run JavaScript code after the script has...

app/page.tsx

```
'use client'

import { useRef } from 'react'
import Script from 'next/script'

export default function Page() {
  const mapRef = useRef()

  return (
    <>
      <div ref={mapRef}></div>…
```

onError

> ***Warning:*** `onError` *does not yet work with Server Components and can only be used in Client Components.* `onError` *cannot be used with the* `beforeInteractive` *loading strategy.*

Sometimes it is...

app/page.tsx

```
'use client'

import Script from 'next/script'

export default function Page() {
  return (
    <>
      <Script
        src="https://example.com/script.js"
        onError={(e: Error) => {…
```

Version History

Version	Changes
`v13.0.0`	`beforeInteractive` and `afterInteractive` is modified to support `app`.
`v12.2.4`	`onReady` prop added.
`v12.2.2`	Allow `next/script`...

Parameters

Name	Type	Description	Default	Required
`src`	String	A path string specifying the URL of an external script. This can be either an absolute external URL or an internal path. The `src` property is required unless an inline script is used.		Yes
`strategy`	String	The loading strategy of the script. One of `beforeInteractive`, `afterInteractive`, `lazyOnload`, or `worker`.	afterInteractive	No
`onLoad`	Function	Executes JavaScript code after the script has finished loading. Only works with `afterInteractive` or `lazyOnload` strategies. Cannot be used with Server Components or `beforeInteractive`.		No
`onReady`	Function	Executes JavaScript code after the script's load event when it first loads and after every subsequent component re-mount. Cannot be used with Server Components.		No
`onError`	Function	Handles errors when a script fails to load. Cannot be used with Server Components or the `beforeInteractive` loading strategy.		No

adapterPath

`API`

Source: https://nextjs.org/docs/app/api-reference/config/next-config-js/adapterPath

Documentation for the Next.js adapterPath configuration option, which allows deployment platforms or build systems to integrate with the Next.js build process via adapters.

Overview

Next.js provides a built-in adapters API. It allows deployment platforms or build systems to integrate with the Next.js build process. For a full reference implementation, see the…

Configuration

To use an adapter, specify the path to your adapter module in `adapterPath`:

Alternatively `NEXT_ADAPTER_PATH` can be set to enable zero-config usage in deployment platforms.

```
js
```

```
/** @type {import('next').NextConfig} */
const nextConfig = {
  adapterPath: require.resolve('./my-adapter.js'),
}

module.exports = nextConfig
```

Adapters

For full adapter implementation details, use the dedicated Adapters section:

- Configuration
- [Creating an...

Creating an Adapter

See Creating an Adapter.

API Reference

See API Reference.

Testing Adapters

See Testing Adapters.

Routing with `@next/routing`

See Routing with `@next/routing`.

Implementing PPR in an Adapter

See Implementing PPR in an Adapter.

Runtime Integration

See Runtime Integration.

Invoking Entrypoints

See Invoking Entrypoints.

Output Types

See Output Types.

Routing Information

See Routing Information.

Use Cases

See Use Cases.

allowedDevOrigins

API

Source: https://nextjs.org/docs/app/api-reference/config/next-config-js/allowedDevOrigins

Describes the `allowedDevOrigins` config option in Next.js, which allows additional origins to request the dev server during development.

allowedDevOrigins

Next.js blocks cross-origin requests to dev-only assets and endpoints during development by default to prevent unauthorized access.

To configure a Next.js application to allow requests from origins...

next.config.js

```
module.exports = {
  allowedDevOrigins: ['local-origin.dev', '*.local-origin.dev'],
}
```

Parameters

Name	Type	Description	Default	Required
allowedDevOrigins	string[]	Sets additional origins that can request the dev server in development mode. Accepts exact hostnames and wildcard subdomains (e.g., 'local-origin.dev', '*.local-origin.dev').	[]	No

appDir

API

Source: https://nextjs.org/docs/app/api-reference/config/next-config-js/appDir

Documents the `appDir` configuration option in next.config.js, which enables the App Router. It is a legacy API, no longer needed as of Next.js 13.4.

appDir

This is a legacy API and no longer recommended. It's still supported for backward compatibility.

Good to know: This option is no longer needed as of Next.js 13.4. The App Router is now stable.

The...

assetPrefix

API

Source: https://nextjs.org/docs/app/api-reference/config/next-config-js/assetPrefix

Explains how to configure the `assetPrefix` option in `next.config.js` to serve static assets from a CDN.

Set up a CDN

Attention: Deploying to Vercel automatically configures a global CDN for your Next.js project. You do not need to manually setup an Asset Prefix.

Good to know: Next.js 9.5+ added support for a...

next.config.mjs

```
// @ts-check
import { PHASE_DEVELOPMENT_SERVER } from 'next/constants'

export default (phase) => {
  const isDev = phase === PHASE_DEVELOPMENT_SERVER
  /**
   * @type {import('next').NextConfig}…
```

text

```
/_next/static/chunks/4b9b41aaa062cbbfeff4add70f256968c51ece5d.4d708494b3aed70c04f0.js
```

text

```
https://cdn.mydomain.com/_next/static/chunks/4b9b41aaa062cbbfeff4add70f256968c51ece5d.4d708494b3aed70
```

Parameters

Name	Type	Description	Default	Required
assetPrefix	string	The URL prefix to use for static assets when served from a CDN.	undefined	No

authInterrupts

API

Source: https://nextjs.org/docs/app/api-reference/config/next-config-js/authInterrupts

Reference for the experimental authInterrupts option in next.config.js, which enables the forbidden and unauthorized APIs in a Next.js application.

authInterrupts

This feature is currently available in the canary channel and subject to change. Try it out by upgrading Next.js, and share your feedback on...

next.config.ts

```
import type { NextConfig } from 'next'

const nextConfig: NextConfig = {
  experimental: {
    authInterrupts: true,
  },
}

export default nextConfig
```

Parameters

Name	Type	Description	Default	Required
`experimental.authInterrupts`	boolean	Enables the use of the forbidden and unauthorized APIs in your application. While these functions are experimental, this option must be enabled to use them.		No

cacheComponents

API

Source: https://nextjs.org/docs/app/api-reference/config/next-config-js/cacheComponents

This page documents the `cacheComponents` configuration flag in Next.js, which enables component and function-level caching using the `use cache` directive, and implements Partial Prerendering as the...

Usage

Cache Components enables component and function-level caching using the `use cache` directive. Data fetching is dynamic by default, and you choose what...

typescript

```
import type { NextConfig } from 'next'

const nextConfig: NextConfig = {
  cacheComponents: true,
}

export default nextConfig
```

Navigation with Activity

When `cacheComponents` is enabled, Next.js uses React's `<Activity>` component to preserve component state during client-side navigation.

Rather than...

Version History

Version	Change
16.0.0	`cacheComponents` introduced. This flag controls the `ppr`, `useCache`, and `dynamicIO` flags as a single, unified configuration.

Learn more

[### Caching

Learn how to cache data and UI in Next.js](/docs/app/getting-started/caching)[### ISR with Cache Components

Learn how to prerender a subset of dynamic routes, serve App Shells for the...

next.config.js: cacheHandlers | Next.js

API

Source: https://nextjs.org/docs/app/api-reference/config/next-config-js/cacheHandlers

This page documents the `cacheHandlers` configuration in next.config.js, which lets you define custom cache storage implementations for `'use cache'` and `'use cache: remote'`. It covers handler...

cacheHandlers

The `cacheHandlers` configuration allows you to define custom cache storage implementations for `'use cache'` and [`'use cache:...

When to use custom cache handlers

Most applications don't need custom cache handlers. The default in-memory cache works well in the typical use case.

Custom cache handlers are for advanced scenarios where you need to either...

Usage

To configure custom cache handlers:

- Define your cache handler in a separate file, see examples for implementation details.
- Reference the file path in your Next config file

next.config.ts

```
import type { NextConfig } from 'next'

const nextConfig: NextConfig = {
  cacheHandlers: {
    default: require.resolve('./cache-handlers/default-handler.js'),
    remote:…
```

Handler types

- `default`: Used by the `'use cache'` directive
- `remote`: Used by the `'use cache: remote'` directive

If you don't configure `cacheHandlers`, Next.js uses an in-memory LRU (Least Recently…

API Reference

A cache handler must implement the `CacheHandler` interface with the following methods:

get()

Retrieve a cache entry for the given cache key.

Parameter	Type	Description
`cacheKey`	`string`	The unique key for the cache entry.
`softTags`	`string[]`	…

typescript

```
get(cacheKey: string, softTags: string[]): Promise<CacheEntry | undefined>
```

javascript

```
const cacheHandler = {
  async get(cacheKey, softTags) {
    const entry = cache.get(cacheKey)
    if (!entry) return undefined

    // Check if expired
    const now = Date.now()
    if (now >…
```

set()

Store a cache entry for the given cache key.

Parameter	Type	Description
`cacheKey`	`string`	The unique key to store the entry under.
`pendingEntry`	…	

typescript

```
set(cacheKey: string, pendingEntry: Promise<CacheEntry>): Promise<void>
```

```javascript
const cacheHandler = {
  async set(cacheKey, pendingEntry) {
    // Wait for the entry to be ready
    const entry = await pendingEntry

    // Store in your cache system
    cache.set(cacheKey,…
```

refreshTags()

Called periodically before starting a new request to sync with external tag services.

This is useful if you're coordinating cache invalidation across multiple instances or services. For in-memory…

```typescript
refreshTags(): Promise<void>
```

```javascript
const cacheHandler = {
  async refreshTags() {
    // For in-memory cache, no action needed
    // For distributed cache, sync tag state from external service
  },
}
```

getExpiration()

Get the maximum revalidation timestamp for a set of tags.

Parameter	Type	Description
`tags`	`string[]`	Array of tags to check expiration for.

Returns:

- `0` if…

```typescript
getExpiration(tags: string[]): Promise<number>
```

```javascript
```

```
const cacheHandler = {
  async getExpiration(tags) {
    // Return 0 if not tracking tag revalidation
    return 0

    // Or return the most recent revalidation timestamp
    // return…
```

updateTags()

Called when tags are revalidated or expired.

Parameter	Type	Description
`tags`	`string[]`	Array of tags to update.
`durations`	`{ expire?: number }`	…

typescript

```
updateTags(tags: string[], durations?: { expire?: number }): Promise<void>
```

javascript

```
const cacheHandler = {
  async updateTags(tags, durations) {
    // Invalidate all cache entries with matching tags
    for (const [key, entry] of cache.entries()) {
      if (entry.tags.some((tag)…
```

CacheEntry Type

The `CacheEntry` object has the following structure:

| Property | Type | Description | | --- |…

typescript

```
interface CacheEntry {
  value: ReadableStream<Uint8Array>
  tags: string[]
  stale: number
  timestamp: number
  expire: number
  revalidate: number
}
```

Examples

Basic in-memory cache handler

Here's a minimal implementation using a `Map` for storage. This example demonstrates the core concepts, but for a production-ready implementation with LRU eviction, error handling, and tag…

cache-handlers/memory-handler.js

```
const cache = new Map()
const pendingSets = new Map()

module.exports = {
  async get(cacheKey, softTags) {
    // Wait for any pending set operation to complete
    const pendingPromise =…
```

External storage pattern

For durable storage like Redis or a database, you'll need to serialize the cache entries. Here's a simple Redis example:

cache-handlers/redis-handler.js

```
const { createClient } = require('redis')

const client = createClient({ url: process.env.REDIS_URL })
client.connect()

module.exports = {
  async get(cacheKey, softTags) {
    // Retrieve from…
```

Distributed Tag Coordination

When running multiple Next.js instances, tag invalidation must be coordinated across instances. The default in-memory handler only tracks tags locally, so calling `revalidateTag()` on one instance…

cache-handlers/distributed-tags.js

```
const { createClient } = require('redis')

const client = createClient({ url: process.env.REDIS_URL })
client.connect()

// Local cache of tag timestamps, synced via refreshTags
const…
```

Soft Tags

Soft tags are implicit tags that Next.js automatically generates based on the route path. For example, the route `/blog/hello` generates soft tags for `/`, `/blog`, `/blog/hello`, and their…

Handling Streams

The `CacheEntry.value` is a `ReadableStream<Uint8Array>`. When implementing a cache handler that stores entries externally, keep in…

Error Handling

Cache operations should be implemented defensively:

- `set()` **failure**: the response is still served to the user because `set()` is called asynchronously after the response stream is already…

Platform Support

Deployment Option	Supported
Node.js server	Yes
Docker container	Yes...

Version History

Version	Changes
`v16.0.0`	`cacheHandlers` introduced.

Related

View related API references.

- use cache — Learn how to use the "use cache" directive to cache data in your Next.js application.
- [use cache:...

Parameters

Name	Type	Description	Default	Required
`cacheKey`	string	The unique key for the cache entry.		Yes
`softTags`	string[]	Implicit tags derived from the route path. See Soft Tags for how to use them.		Yes
`cacheKey`	string	The unique key to store the entry under.		Yes
`pendingEntry`	Promise<CacheEntry>	A promise that resolves to the cache entry.		Yes
`tags`	string[]	Array of tags to check expiration for.		Yes
`tags`	string[]	Array of tags to update.		Yes
`durations`	{ expire?: number }	Optional expiration duration in seconds.		No
`value`	ReadableStream<Uint8Array>	The cached data as a stream.		Yes
`tags`	string[]	Cache tags (excluding soft tags).		Yes
`stale`	number	Duration in seconds for client-side staleness.		Yes
`timestamp`	number	When the entry was created (timestamp in milliseconds).		Yes
`expire`	number	How long the entry is allowed to be used (in seconds).		Yes
`revalidate`	number	How long until the entry should be revalidated (in seconds).		Yes

next.config.js: cacheLife | Next.js

`API`

Source: https://nextjs.org/docs/app/api-reference/config/next-config-js/cacheLife

Reference for the cacheLife option in next.config.js, which lets you define custom cache profiles for use with the cacheLife function and the use cache directive.

cacheLife

The `cacheLife` option allows you to define **custom cache profiles** when using the `cacheLife` function inside components or functions, and within the...

Usage

To define a profile, enable the `cacheComponents` flag and add the cache profile in the `cacheLife` object in the `next.config.js`...

next.config.ts

```
import type { NextConfig } from 'next'

const nextConfig: NextConfig = {
  cacheComponents: true,
  cacheLife: {
    blog: {
      stale: 3600, // 1 hour
      revalidate: 900, // 15 minutes…
```

app/actions.ts

```
import { cacheLife } from 'next/cache'

export async function getCachedData() {
  'use cache'
  cacheLife('blog')
  const data = await fetch('/api/data')
  return data
}
```

Reference

The configuration object has key values with the following format:

Property	Value	Description	Requirement
`stale`	`number`	Duration the...	

Related

View related API references.

- use cache — Learn how to use the "use cache" directive to cache data in your Next.js application. -...

Parameters

Name	Type	Description	Default	Required
`stale`	number	Duration the client should cache a value without checking the server.		No
`revalidate`	number	Frequency at which the cache should refresh on the server; stale values may be served while revalidating.		No
`expire`	number	Maximum duration for which a value can remain stale before switching to dynamic. Must be longer than `revalidate`.		No

compress

`API`

Source: https://nextjs.org/docs/app/api-reference/config/next-config-js/compress

This page describes the `compress` option in `next.config.js`, which controls gzip compression for rendered content and static files when using `next start` or a custom server.

compress

By default, Next.js uses `gzip` to compress rendered content and static files when using `next start` or a custom server. This is an optimization for applications that do not have compression...

Disabling compression

To disable **compression**, set the `compress` config option to `false`:

We **do not recommend disabling compression** unless you have compression configured on your server, as compression reduces...

next.config.js

```
module.exports = {
  compress: false,
}
```

Parameters

Name	Type	Description	Default	Required
`compress`	boolean	Enables or disables gzip compression for rendered content and static files when using `next start` or a custom server. Defaults to `true` unless compression is already configured via a custom server.	true	No

crossOrigin

API

Source: https://nextjs.org/docs/app/api-reference/config/next-config-js/crossOrigin

Use the `crossOrigin` option to add a `crossOrigin` attribute in all `<script>` tags generated by the `next/script` component, and define how cross-origin requests should be handled.

crossOrigin

Use the `crossOrigin` option to add a `crossOrigin` attribute in all `<script>` tags generated by the...

```javascript
module.exports = {
  crossOrigin: 'anonymous',
}
```

Options

- `'anonymous'`: Adds `crossOrigin="anonymous"` attribute.
- `'use-credentials'`: Adds...

Parameters

Name	Type	Description	Default	Required
`crossOrigin`	string	Adds a crossOrigin attribute to all script tags generated by next/script. Can be 'anonymous' or 'use-credentials'.		No

devIndicators

API

Source: https://nextjs.org/docs/app/api-reference/config/next-config-js/devIndicators

Configuration for the on-screen development indicator in Next.js, allowing you to set its position or hide it entirely.

devIndicators

`devIndicators` allows you to configure the on-screen indicator that gives context about the current route you're viewing during development. Open `next.config.ts` and set `position` to choose where...

```next.config.ts
```

```
import type { NextConfig } from 'next'

const nextConfig: NextConfig = {
  devIndicators: {
    position: 'bottom-right', // 'bottom-left' | 'bottom-right' | 'top-left' | 'top-right'
  },
}

export…
```

next.config.ts

```
const nextConfig: NextConfig = {
  devIndicators: false,
}

export default nextConfig
```

Troubleshooting

Indicator not marking a route as static

If you expect a route to be static and the indicator has marked it as dynamic, it's likely the route has opted out of prerendering. You can confirm if a route is...

Build Output

```
Route (app)
┌ ○ /_not-found
└ ƒ /products/[id]

○  (Static)   prerendered as static content
ƒ  (Dynamic)  server-rendered on demand
```

Version History

Version	Changes
`v16.0.0`	`appIsrStatus`, `buildActivity`, and `buildActivityPosition` options have been removed.
`v15.2.0`	Improved on-screen indicator with new...

Parameters

Name	Type	Description	Default	Required
`devIndicators`	object | false	Configuration for the on-screen development indicator. Set to `false` to hide it entirely.		No
`position`	'bottom-left' | 'bottom-right' | 'top-left' | 'top-right'	Position of the indicator on the screen.	'bottom-left'	No

env

API

Source: https://nextjs.org/docs/app/api-reference/config/next-config-js/env

Describes the legacy `env` config option in next.config.js for adding environment variables to the JavaScript bundle, noting it's no longer recommended.

env

This is a legacy API and no longer recommended. It's still supported for backward compatibility.

> *Since the release of Next.js 9.4 we now have a more intuitive...*

next.config.js

```
module.exports = {
  env: {
    customKey: 'my-value',
  },
}
```

jsx

```
function Page() {
  return <h1>The value of customKey is: {process.env.customKey}</h1>
}

export default Page
```

jsx

```
return <h1>The value of customKey is: {process.env.customKey}</h1>
```

jsx

```
return <h1>The value of customKey is: {'my-value'}</h1>
```

expireTime

API

Source: https://nextjs.org/docs/app/api-reference/config/next-config-js/expireTime

Describes the `expireTime` configuration option in Next.js, which sets a custom stale-while-revalidate expire time for CDNs in the Cache-Control header for ISR pages.

expireTime

You can specify a custom `stale-while-revalidate` expire time for CDNs to consume in the `Cache-Control` header for ISR enabled pages.

Open `next.config.js` and add the `expireTime` config:

Now...

next.config.js

```
module.exports = {
  // one hour in seconds
  expireTime: 3600,
}
```

Parameters

Name	Type	Description	Default	Required
expireTime	number	Custom stale-while-revalidate expire time in seconds for CDNs to consume in the Cache-Control header for ISR enabled pages.		No

generateBuildId

API

Source: https://nextjs.org/docs/app/api-reference/config/next-config-js/generateBuildId

Explains how to use the generateBuildId option in next.config.js to generate a consistent build ID for your Next.js application.

generateBuildId

Next.js generates an ID during `next build` to identify which version of your application is being served. The same build should be used and boot up multiple containers.

If you are rebuilding for...

next.config.js

```
module.exports = {
  generateBuildId: async () => {
    // This could be anything, using the latest git hash
    return process.env.GIT_HASH
  },
}
```

generateEtags

API

Source: https://nextjs.org/docs/app/api-reference/config/next-config-js/generateEtags

This page describes the `generateEtags` configuration option in `next.config.js`, which controls whether Next.js generates ETags for every page. It shows how to disable ETag generation for HTML pages.

generateEtags

Next.js will generate etags for every page by default. You may want to disable etag generation for HTML pages depending on your cache strategy.

Open...

next.config.js

```
module.exports = {
  generateEtags: false,
}
```

Parameters

Name	Type	Description	Default	Required
generateEtags	boolean	When set to false, disables ETag generation for HTML pages. Defaults to true.	true	No

next.config.js: headers

API

Source: https://nextjs.org/docs/app/api-reference/config/next-config-js/headers

This page explains how to set custom HTTP headers on responses in Next.js using the `headers` key in `next.config.js`, covering path matching, regex, header/cookie/query matching, basePath/i18n...

headers

Headers allow you to set custom HTTP headers on the response to an incoming request on a given path.

To set custom HTTP headers you can use the `headers` key in `next.config.js`:

`headers` can be...

next.config.js

```
module.exports = {
  headers() {
    return [
      {
        source: '/about',
        headers: [
          {
            key: 'x-custom-header',
            value: 'my custom header value',…
```

Header Overriding Behavior

If two headers match the same path and set the same header key, the last header key will override the first. Using the below headers, the path `/hello` will result in the header `x-hello` being...

next.config.js

```
module.exports = {
  headers() {
    return [
      {
        source: '/:path*',
        headers: [
          {
            key: 'x-hello',
            value: 'there',
          },
        ],…
```

Path Matching

Path matches are allowed, for example `/blog/:slug` will match `/blog/first-post` (no nested paths):

The pattern `/blog/:slug` matches `/blog/first-post` and `/blog/post-1` but not a nested path...

next.config.js

```
module.exports = {
  headers() {
    return [
      {
        source: '/blog/:slug',
        headers: [
          {
            key: 'x-slug',
            value: ':slug', // Matched parameters can be…
```

Wildcard Path Matching

To match a wildcard path you can use `*` after a parameter, for example `/blog/:slug*` will match `/blog/a/b/c/d/hello-world`:

next.config.js

```
module.exports = {
  headers() {
    return [
      {
        source: '/blog/:slug*',
        headers: [
          {
            key: 'x-slug',
            value: ':slug*', // Matched parameters can…
```

Regex Path Matching

To match a regex path you can wrap the regex in parenthesis after a parameter, for example `/blog/:slug(\d{1,})` will match `/blog/123` but not `/blog/abc`:

The following characters `(`, `)`, `{`,...

next.config.js

```
module.exports = {
  headers() {
    return [
      {
        source: '/blog/:post(\\d{1,})',
        headers: [
          {
            key: 'x-post',
            value: ':post',
          },…
```

next.config.js

```
module.exports = {
  headers() {
    return [
      {
        // this will match `/english(default)/something` being requested
        source: '/english\\(default\\)/:slug',
        headers: […
```

Header, Cookie, and Query Matching

To only apply a header when header, cookie, or query values also match the `has` field or don't match the `missing` field can be used. Both the `source` and all `has` items must match and all...

next.config.js

```
module.exports = {
  headers() {
    return [
      // if the header `x-add-header` is present,
      // the `x-another-header` header will be applied
      {
        source: '/:path*',
        has:…
```

Headers with basePath support

When leveraging `basePath` support with headers each `source` is automatically prefixed with the `basePath` unless you add `basePath: false`...

next.config.js

```
module.exports = {
  basePath: '/docs',

  headers() {
    return [
      {
        source: '/with-basePath', // becomes /docs/with-basePath
        headers: [
          {
            key:…
```

Headers with i18n support

When leveraging `i18n` support with headers each `source` is automatically prefixed to handle the configured `locales` unless you add `locale: false` to the...

next.config.js

```
module.exports = {
  i18n: {
    locales: ['en', 'fr', 'de'],
    defaultLocale: 'en',
  },

  headers() {
    return [
      {
        source: '/with-locale', // automatically handles all locales…
```

Cache-Control

Next.js sets the `Cache-Control` header of `public, max-age=31536000, immutable` for truly immutable assets. It cannot be overridden. These immutable files contain a SHA-hash in the file name, so...

Options

CORS

Cross-Origin Resource Sharing (CORS) is a security feature that allows you to control which sites can access your resources. You can set the...

```
headers() {
    return [
      {
        source: "/api/:path*",
        headers: [
          {
            key: "Access-Control-Allow-Origin",
            value: "*", // Set your origin
          },…
```

X-DNS-Prefetch-Control

This header controls DNS prefetching, allowing browsers to proactively perform domain name resolution on external links,...

```
{
  key: 'X-DNS-Prefetch-Control',
  value: 'on'
}
```

Strict-Transport-Security

This header informs browsers it should only be accessed using HTTPS, instead of using HTTP. Using the configuration...

```
{
  key: 'Strict-Transport-Security',
  value: 'max-age=63072000; includeSubDomains; preload'
}
```

X-Frame-Options

This header indicates whether the site should be allowed to be displayed within an `iframe`. This can prevent against...

```
{
  key: 'X-Frame-Options',
  value: 'SAMEORIGIN'
}
```

Permissions-Policy

This header allows you to control which features and APIs can be used in the browser. It was previously named...

```
{
  key: 'Permissions-Policy',
  value: 'camera=(), microphone=(), geolocation=(), browsing-topics=()'
}
```

X-Content-Type-Options

This header prevents the browser from attempting to guess the type of content if the `Content-Type` header is not...

```
{
  key: 'X-Content-Type-Options',
  value: 'nosniff'
}
```

Referrer-Policy

This header controls how much information the browser includes when navigating from the current website (origin) to another.

```
{
  key: 'Referrer-Policy',
  value: 'origin-when-cross-origin'
}
```

Content-Security-Policy

Learn more about adding a Content Security Policy to your application.

Version History

Version	Changes
`v13.3.0`	`missing` added.
`v10.2.0`	`has` added.
`v9.5.0`	Headers added.

Parameters

Name	Type	Description	Default	Required
`source`	string	The incoming request path pattern.		Yes
`headers`	Array<{ key: string, value: string }>	An array of response header objects, with `key` and `value` properties.		Yes
`basePath`	false | undefined	If false the basePath won't be included when matching, can be used for external rewrites only.	undefined	No
`locale`	false | undefined	Whether the locale should not be included when matching.	undefined	No
`has`	Array<{ type: 'header' | 'cookie' | 'host' | 'query', key: string, value?: string | undefined }>	An array of has objects with the `type`, `key` and `value` properties. All must match for the header to be applied.		No
`missing`	Array<{ type: 'header' | 'cookie' | 'host' | 'query', key: string, value?: string | undefined }>	An array of missing objects with the `type`, `key` and `value` properties. All must not match for the header to be applied.		No

htmlLimitedBots

API

Source: https://nextjs.org/docs/app/api-reference/config/next-config-js/htmlLimitedBots

The `htmlLimitedBots` config allows you to specify a list of user agents that should receive blocking metadata instead of streaming metadata in Next.js.

htmlLimitedBots

The `htmlLimitedBots` config allows you to specify a list of user agents that should receive blocking metadata instead of [streaming...

```typescript
import type { NextConfig } from 'next'

const config: NextConfig = {
  htmlLimitedBots: /MySpecialBot|MyAnotherSpecialBot|SimpleCrawler/,
}

export default config
```

Default list

Next.js includes a default list of HTML limited bots, including:

- Google crawlers (e.g. Mediapartners-Google, AdsBot-Google, Google-PageRenderer)
- Bingbot
- Twitterbot
- Slackbot

See the full...

```typescript
const config: NextConfig = {
  htmlLimitedBots: /MySpecialBot|MyAnotherSpecialBot|SimpleCrawler/,
}

export default config
```

Disabling

To fully disable streaming metadata:

```typescript
import type { NextConfig } from 'next'

const config: NextConfig = {
  htmlLimitedBots: /.*/,
}

export default config
```

Version History

Version	Changes
15.2.0	`htmlLimitedBots` option introduced.

Custom Next.js Cache Handler

`API`

Source: https://nextjs.org/docs/app/api-reference/config/next-config-js/incrementalCacheHandlerPath

Explains how to configure a custom cache handler in next.config.js to persist cached pages and data to durable storage or share cache across instances, including method signatures and image...

Custom Next.js Cache Handler

You can configure the Next.js cache location if you want to persist cached pages and data to durable storage, or share the cache across multiple containers or instances of your Next.js...

```js
module.exports = {
  cacheHandler: require.resolve('./cache-handler.js'),
  cacheMaxMemorySize: 0, // disable default in-memory caching
}
```

API Reference

The cache handler can implement the following methods: `get`, `set`, `revalidateTag`, and `resetRequestCache`.

get()

The `ctx` parameter contains a `kind` property that indicates the type of cache entry being retrieved. Possible values include `'APP_PAGE'`, `'APP_ROUTE'`, `'PAGES'`, `'FETCH'`, and...

set()

The `data` object contains a `kind` property that indicates the type of cache entry. For image optimization, `kind` will be `'IMAGE'` and the data will include properties like `buffer`, `etag`,...

revalidateTag()

Returns `Promise<void>`. Learn more about revalidating data or the `revalidateTag()`...

resetRequestCache()

This method resets the temporary in-memory cache for a single request before the next request.

Returns `void`.

Good to know:

- `revalidatePath` is a convenience layer on top of cache tags....

Image Optimization Caching

The `cacheHandler` can also be used for caching optimized images from `next/image`. To enable this, set `images.customCacheHandler` to `true` in your `next.config.js`:

> ***Good to know*** *: This...*

```js
module.exports = {
  cacheHandler: require.resolve('./cache-handler.js'),
  images: {
    customCacheHandler: true,
  },
}
```

Platform Support

Deployment Option	Supported
Node.js server	Yes
Docker container	Yes...

Version History

Version	Changes
`v16.2.0`	`cacheHandler` support for image optimization caching.
`v14.1.0`	Renamed to `cacheHandler` and became stable.
`v13.4.0`	...

Parameters

Name	Type	Description	Default	Required
`key`	string	The key to the cached value (used in get).		Yes
`ctx`	object	Context including the cache entry kind (used in get).		Yes
`key`	string	The key to store the data under (used in set).		Yes
`data`	Data or null	The data to be cached (used in set).		Yes
`ctx`	{ tags: [] }	The cache tags provided (used in set).		Yes
`tag`	string or string[]	The cache tags to revalidate (used in revalidateTag).		Yes

next.config.js: inlineCss | Next.js

`API`

Source: https://nextjs.org/docs/app/api-reference/config/next-config-js/inlineCss

Experimental Next.js configuration option that inlines CSS into the <head> by generating <style> tags instead of <link> tags, including trade-offs, benefits, and known limitations.

inlineCss

This feature is currently experimental and subject to change, it's not recommended for production. Try it out and share your feedback on GitHub.

Usage

Experimental support for inlining CSS in the ``. When this flag is enabled, all places where we normally generate a `<link>` tag will instead have a generated `<style>` tag.

next.config.ts

```ts
import type { NextConfig } from 'next'

const nextConfig: NextConfig = {
  experimental: {
    inlineCss: true,
  },
}

export default nextConfig
```

Trade-Offs

- **Enable** if you use atomic CSS (like Tailwind) and want to optimize first-load performance for new visitors
- **Skip** if returning visitors are common and you want them to benefit from cached…

When Inline CSS Helps

Normally, the browser must download HTML, parse it, discover CSS `<link>` tags, then request stylesheets before it can render. Inlining [eliminates this request...

When External CSS is Better

Inlined styles cannot be cached separately from HTML. Every page load re-downloads the same CSS.

This trade-off matters most with:

- **Returning visitors**: Users who visit your site repeatedly...

Good to know

This feature is currently experimental and has some known limitations:

- CSS inlining is applied globally and cannot be configured on a per-page basis
- Styles are duplicated during initial page...

next.config.js: instrumentationClientInject | Next.js

`API`

Source: https://nextjs.org/docs/app/api-reference/config/next-config-js/instrumentationClientInject

This page documents the `instrumentationClientInject` option in `next.config.js`, which allows plugins to inject client-side instrumentation modules that run before the user's...

instrumentationClientInject

`instrumentationClientInject` is a list of modules that are imported on the client for their side effects before the user's...

withMyInstrumentation.js

```
module.exports = function withMyInstrumentation(nextConfig = {}) {
  return {
    ...nextConfig,
    instrumentationClientInject: [
      ...(nextConfig.instrumentationClientInject ?? []),…
```

next.config.js

```
/** @type {import('next').NextConfig} */
module.exports = {
  instrumentationClientInject: [
    'my-analytics-package',
    './lib/sentry-client.js',
  ],
}
```

Execution order

Modules run on the client in this order:

- Each entry in `instrumentationClientInject`, in array order.

- The project's `instrumentation-client.{js,ts}` file, if present.
- React hydration.

Router navigation hook

Each injected module may optionally export an `onRouterTransitionStart` function with the same signature as the one documented for the [`instrumentation-client` file...

lib/sentry-client.js

```
// Side-effectful setup runs at load time.
setupSentry()

export function onRouterTransitionStart(url, navigationType) {
  recordNavigationBreadcrumb(url, navigationType)
}
```

Version history

Version	Changes
`v16.3.0`	`instrumentationClientInject` introduced

next.config.js: logging

`API`

Source: https://nextjs.org/docs/app/api-reference/config/next-config-js/logging

Configuration options for logging in Next.js, including fetch logging, server functions, incoming requests, browser console logs, and disabling logging.

Options

Fetching

You can configure the logging level and whether the full URL is logged to the console when running Next.js in development mode.

Any `fetch` requests that are restored from the [Server Components HMR...

javascript

```
module.exports = {
  logging: {
    fetches: {
      fullUrl: true,
    },
  },
}
```

javascript

```
module.exports = {
  logging: {
    fetches: {
      hmrRefreshes: true,
    },
  },
}
```

Server Functions

Server Function invocations are logged by default during development. You can disable this by setting `logging.serverFunctions` to `false`.

When...

javascript

```
module.exports = {
  logging: {
    serverFunctions: false,
  },
}
```

terminal

```
POST /
  └─ ƒ myAction(arg1, arg2) in 5ms app/actions.ts
```

Incoming Requests

By default all the incoming requests will be logged in the console during development. You can use the `incomingRequests` option to decide which requests to ignore. Since this is only logged in...

javascript

```
module.exports = {
  logging: {
    incomingRequests: {
      ignore: [/\api\/v1\/health/],
    },
  },
}
```

javascript

```
module.exports = {
  logging: {
    incomingRequests: false,
  },
}
```

Browser Console Logs

You can forward browser console logs (such as `console.log`, `console.warn`, `console.error`) to the terminal during development. This is useful for debugging client-side code without needing to...

```javascript
module.exports = {
  logging: {
    browserToTerminal: true,
  },
}
```

Source Location

When enabled, browser logs include source location information (file path and line number) by default. For example:

```tsx
'use client'

export default function Home() {
  return (
    <button
      type="button"
      onClick={() => {
        console.log('Hello World')
      }}
    >
      Click me
    </button>
  )
}
```

```terminal
[browser] Hello World (app/page.tsx:8:17)
```

Disabling Logging

In addition, you can disable the development logging by setting `logging` to `false`.

```javascript
module.exports = {
  logging: false,
}
```

Version History

Version	Changes
`v16.2.0`	`browserToTerminal` added (moved from `experimental.browserDebugInfoInTerminal`)
`v15.4.0`	`experimental.browserDebugInfoInTerminal` introduced...

Parameters

Name	Type	Description	Default	Required
`logging`	boolean | object	Main logging configuration. Can be set to `false` to disable all development logging.		No
`logging.fetches.fullUrl`	boolean	Whether to log the full URL for fetch requests in development.		No
`logging.fetches.hmrRefreshes`	boolean	Whether to log fetch requests restored from the Server Components HMR cache.		No
`logging.serverFunctions`	boolean	Whether to log Server Function invocations in development. Defaults to `true`.		No
`logging.incomingRequests`	object | boolean	Configuration for logging incoming requests. Can be an object with an `ignore` array or `false` to disable.		No
`logging.incomingRequests.ignore`	array	Array of regular expressions to match requests to ignore from logging.		No
`logging.browserToTerminal`	boolean | string	Forward browser console logs to terminal. Can be `true`, `false`, `'warn'`, or `'error'`. Defaults to `'warn'`.		No

next.config.js: onDemandEntries | Next.js

API

Source: https://nextjs.org/docs/app/api-reference/config/next-config-js/onDemandEntries

Explains the onDemandEntries configuration option in next.config.js, which controls how the development server keeps built pages in memory.

onDemandEntries

Next.js exposes some options that give you some control over how the server will dispose or keep in memory built pages in development.

To change the defaults, open `next.config.js` and add the...

```
next.config.js
```

```
module.exports = {
  onDemandEntries: {
    // period (in ms) where the server will keep pages in the buffer
    maxInactiveAge: 25 * 1000,
    // number of pages that should be kept simultaneously…
```

Parameters

Name	Type	Description	Default	Required
`maxInactiveAge`	number	Period (in ms) where the server will keep pages in the buffer.		No
`pagesBufferLength`	number	Number of pages that should be kept simultaneously without being disposed.		No

optimizePackageImports

API

Source: https://nextjs.org/docs/app/api-reference/config/next-config-js/optimizePackageImports

Describes the experimental `optimizePackageImports` option in next.config.js, which optimizes package imports by only loading the modules actually used, and lists the libraries optimized by default.

optimizePackageImports

This feature is currently experimental and subject to change, it's not recommended for production. Try it out and share your feedback on GitHub.

Some packages can export hundreds or thousands of...

next.config.js

```
module.exports = {
  experimental: {
    optimizePackageImports: ['package-name'],
  },
}
```

Parameters

Name	Type	Description	Default	Required
`experimental.optimizePackageImports`	string[]	An array of package names to optimize. Adding a package to this option will only load the modules you are actually using, while still giving you the convenience of writing import statements with many...		No

next.config.js: pageExtensions | Next.js

API

Source: https://nextjs.org/docs/app/api-reference/config/next-config-js/pageExtensions

Reference for the pageExtensions option in next.config.js, which customizes which file extensions Next.js treats as pages—for example, to support Markdown and MDX.

pageExtensions

By default, Next.js accepts files with the following extensions: `.tsx`, `.ts`, `.jsx`, `.js`. This can be modified to allow other extensions like markdown (`.md`, `.mdx`).

next.config.js

```
const withMDX = require('@next/mdx')()

/** @type {import('next').NextConfig} */
const nextConfig = {
  pageExtensions: ['js', 'jsx', 'ts', 'tsx', 'md', 'mdx'],
}

module.exports =…
```

Parameters

Name	Type	Description	Default	Required
`pageExtensions`	string[]	Modifies the file extensions that Next.js accepts for pages. Defaults to .tsx, .ts, .jsx, .js; can include markdown extensions like .md and .mdx.	['tsx', 'ts', 'jsx', 'js']	No

partialPrefetching

API

Source: https://nextjs.org/docs/app/api-reference/config/next-config-js/partialPrefetching

Configuration option for Next.js that enables Partial Prefetching at the app level, allowing the framework to prefetch reusable App Shells per route instead of per link.

Usage

`partialPrefetching` requires `cacheComponents`. Without it, `next dev` and `next build` throw at config validation.

```typescript
import type { NextConfig } from 'next'

const nextConfig: NextConfig = {
  cacheComponents: true,
  partialPrefetching: true,
}

export default nextConfig
```

Reference

Value	Description
`true`	Enables Partial Prefetching across the app.
`false`	Default. No change to prefetch behavior.

How prefetches resolve

Before Partial Prefetching, Next.js prefetched per visible link: a page with N links to N routes produced ~N route prefetches as those links entered the viewport.

With `partialPrefetching: true`,...

Per-segment overrides

A segment that exports an explicit `prefetch` value overrides the app-level default for that route.

Version History

Version	Change
16.3.0	`partialPrefetching` introduced. Requires `cacheComponents` to be enabled.

Related

View related API references and guides.

- cacheComponents - Learn how to enable the cacheComponents flag in Next.js.
- [prefetch - API...

Parameters

Name	Type	Description	Default	Required
`partialPrefetching`	boolean	Enables Partial Prefetching at the app level. Requires `cacheComponents` to be enabled.	false	No

prefetchInlining

API

Source: https://nextjs.org/docs/app/api-reference/config/next-config-js/prefetchInlining

Describe the experimental `prefetchInlining` configuration option in Next.js, which controls whether App Router prefetch responses are bundled into a single response, and how to disable or customize...

prefetchInlining

When the App Router prefetches a route, it can bundle small segment responses into a single response instead of requesting each one separately. This reduces the number of prefetch requests at the...

Usage

To turn off prefetch inlining, set `experimental.prefetchInlining` to `false`:

To override the thresholds instead of disabling inlining, pass an object. Any value you omit keeps its default:

next.config.ts

```ts
import type { NextConfig } from 'next'

const nextConfig: NextConfig = {
  experimental: {
    prefetchInlining: false,
  },
}

export default nextConfig
```

next.config.ts

```ts
import type { NextConfig } from 'next'

const nextConfig: NextConfig = {
  experimental: {
    prefetchInlining: {
      maxSize: 2048,
      maxBundleSize: 10240,
    },
  },
}

export default…
```

Reference

Value	Description
`true`	Inlines prefetch responses with the default thresholds. This is the default.
`false`	Disables prefetch inlining. Each segment is prefetched as...

Version History

Version	Change
16.3.0	`experimental.prefetchInlining` enabled by default.
16.2.0	`experimental.prefetchInlining` added.

Related

View related API references and guides.

Link Component — Enable fast client-side navigation with the built-in `next/link`...

Parameters

Name	Type	Description	Default	Required
`prefetchInlining`	boolean | { maxSize?: number; maxBundleSize?: number }	Controls whether prefetch responses are inlined. `true` (default) inlines with default thresholds, `false` disables inlining, or an object customizes the thresholds.	true	No
`maxSize`	number	Largest a single segment response can be to still be eligible for inlining.	2048	No
`maxBundleSize`	number	Largest total size that can be inlined into one bundled prefetch response along a path.	10240	No

productionBrowserSourceMaps

API

Source: https://nextjs.org/docs/app/api-reference/config/next-config-js/productionBrowserSourceMaps

Explains how to enable browser source map generation during production builds in Next.js using the productionBrowserSourceMaps configuration flag.

productionBrowserSourceMaps

Source Maps are enabled by default during development. During production builds, they are disabled to prevent you leaking your source on the client, unless you specifically opt-in with the...

next.config.js

```
module.exports = {
  productionBrowserSourceMaps: true,
}
```

Parameters

Name	Type	Description	Default	Required
productionBrowserSourceMaps	boolean	Enables browser source map generation during production builds.	False	No

next.config.js: proxyClientMaxBodySize

API

Source: https://nextjs.org/docs/app/api-reference/config/next-config-js/proxyClientMaxBodySize

Configuration option to set a size limit on the buffered request body when using proxy in Next.js, preventing excessive memory usage. Defaults to 10MB.

Introduction

This feature is currently experimental and subject to change, it's not recommended for production. Try it out and share your feedback on GitHub. Last updated October 20, 2025

When proxy is used,...

Options

String format (recommended)

Specify the size using a human-readable string format:

Supported units: b, kb, mb, gb

next.config.ts

```
import type { NextConfig } from 'next'

const nextConfig: NextConfig = {
  experimental: {
    proxyClientMaxBodySize: '1mb',
  },
}

export default nextConfig
```

Number format

Alternatively, specify the size in bytes as a number:

next.config.ts

```ts
import type { NextConfig } from 'next'

const nextConfig: NextConfig = {
  experimental: {
    proxyClientMaxBodySize: 1048576, // 1MB in bytes
  },
}

export default nextConfig
```

Behavior

When a request body exceeds the configured limit:

- Next.js will buffer only the first N bytes (up to the limit)
- A warning will be logged to the console indicating the route that exceeded the...

Example

proxy.ts

```ts
import { NextRequest, NextResponse } from 'next/server'

export async function proxy(request: NextRequest) {
  // Next.js automatically buffers the body with the configured size limit
  // You can…
```

app/api/upload/route.ts

```ts
import { NextRequest, NextResponse } from 'next/server'

export async function POST(request: NextRequest) {
  // ...and the body is still available in your route handler
  const body = await…
```

Good to know

- This setting only applies when proxy is used in your application
- The default limit of 10MB is designed to balance memory usage and typical use cases
- The limit applies per-request, not globally...

Parameters

Name	Type	Description	Default	Required
`proxyClientMaxBodySize`	string | number	Sets a size limit on the buffered request body when proxy is used, preventing excessive memory usage. String format supports units b, kb, mb, gb; number format specifies bytes. If exceeded, only the...	10mb	No

reactCompiler

API

Source: https://nextjs.org/docs/app/api-reference/config/next-config-js/reactCompiler

This page documents the `reactCompiler` configuration option in Next.js, which enables the React Compiler to automatically optimize component rendering, reducing the need for manual memoization.

How It Works

The React Compiler runs through a Babel plugin. To keep builds fast, Next.js uses a custom SWC optimization that only applies the React Compiler to relevant files—like those with JSX or React...

bash

```
pnpm add -D babel-plugin-react-compiler
```

next.config.ts

```ts
import type { NextConfig } from 'next'

const nextConfig: NextConfig = {
  reactCompiler: true,
}

export default nextConfig
```

Annotations

You can configure the compiler to run in "opt-in" mode as follows:

Then, you can annotate specific components or hooks with the `"use memo"` directive from React to opt-in:

> ***Note:*** *You can...*

next.config.ts

```
import type { NextConfig } from 'next'

const nextConfig: NextConfig = {
  reactCompiler: {
    compilationMode: 'annotation',
  },
}

export default nextConfig
```

app/page.tsx

```
export default function Page() {
  'use memo'
  // ...
}
```

Parameters

Name	Type	Description	Default	Required
reactCompiler	boolean | { compilationMode?: 'annotation' | 'all' }	Enables the React Compiler. When set to true, the compiler runs on all relevant files. When set to an object with compilationMode: 'annotation', the compiler runs in opt-in mode, requiring 'use memo'...		No

reactMaxHeadersLength

API

Source: https://nextjs.org/docs/app/api-reference/config/next-config-js/reactMaxHeadersLength

This page describes the `reactMaxHeadersLength` option in `next.config.js`, which controls the maximum length of headers emitted by React during prerendering, with a default value of 6000.

reactMaxHeadersLength

During prerendering, React can emit headers that can be added to the response. These can be used to improve performance by allowing the browser to preload resources like fonts, scripts, and...

next.config.js

```
module.exports = {
  reactMaxHeadersLength: 1000,
}
```

Parameters

Name	Type	Description	Default	Required
`reactMaxHeadersLength`	number	Maximum length of headers emitted by React during prerendering. Lower this value if a reverse proxy truncates long headers.	6000	No

next.config.js: sassOptions | Next.js

API

Source: https://nextjs.org/docs/app/api-reference/config/next-config-js/sassOptions

Reference for the `sassOptions` configuration option in next.config.js, which allows you to configure the Sass compiler.

sassOptions

`sassOptions` allow you to configure the Sass compiler.

> ***Good to know:***
>
> - *`sassOptions` are not typed outside of `implementation` because Next.js does not maintain the other possible...*

next.config.ts

```ts
import type { NextConfig } from 'next'

const sassOptions = {
  additionalData: `
    $var: red;
  `,
}

const nextConfig: NextConfig = {
  sassOptions: {
    ...sassOptions,
    implementation:…
```

Parameters

Name	Type	Description	Default	Required
`sassOptions`	object	Options passed to the Sass compiler.		No
`sassOptions.implementation`	string	Specifies the Sass implementation to use, e.g. 'sass-embedded'.		No
`sassOptions.additionalData`	string	Additional data to prepend to every Sass file, e.g. Sass variables.		No
`sassOptions.functions`	object	Custom Sass functions. Only supported with webpack; not available with Turbopack.		No

serverActions

API

Source: https://nextjs.org/docs/app/api-reference/config/next-config-js/serverActions

Options for configuring Server Actions behavior in Next.js application, including allowed origins and body size limit.

allowedOrigins

A list of extra safe origin domains from which Server Actions can be invoked. Next.js compares the origin of a Server Action request with the host domain, ensuring they match to prevent CSRF attacks....

```javascript
/** @type {import('next').NextConfig} */

module.exports = {
  experimental: {
    serverActions: {
      allowedOrigins: ['my-proxy.com', '*.my-proxy.com'],
    },
  },
}
```

bodySizeLimit

By default, the maximum size of the request body sent to a Server Action is 1MB, to prevent the consumption of excessive server resources in parsing large amounts of data, as well as potential DDoS...

```javascript
```

```
/** @type {import('next').NextConfig} */

module.exports = {
  experimental: {
    serverActions: {
      bodySizeLimit: '2mb',
    },
  },
}
```

Enabling Server Actions (v13)

Server Actions became a stable feature in Next.js 14, and are enabled by default. However, if you are using an earlier version of Next.js, you can enable them by setting `experimental.serverActions`...

javascript

```
/** @type {import('next').NextConfig} */
const config = {
  experimental: {
    serverActions: true,
  },
}

module.exports = config
```

Parameters

Name	Type	Description	Default	Required
allowedOrigins	string[]	A list of extra safe origin domains from which Server Actions can be invoked. Next.js compares the origin of a Server Action request with the host domain, ensuring they match to prevent CSRF attacks....		No
bodySizeLimit	string | number	The maximum size of the request body sent to a Server Action. Defaults to 1MB. Can take the number of bytes or any string format supported by bytes, e.g. '500kb' or '3mb'. The limit applies to the...	'1mb'	No
serverActions	boolean	In Next.js 13, set to true to enable Server Actions. In Next.js 14 and later, Server Actions are enabled by default and this option is not needed.	true (in v14+)	No

serverComponentsHmrCache

API

Source: https://nextjs.org/docs/app/api-reference/config/next-config-js/serverComponentsHmrCache

The experimental `serverComponentsHmrCache` option allows you to cache `fetch` responses in Server Components across HMR refreshes in local development, improving performance and reducing API costs.

serverComponentsHmrCache

This feature is currently experimental and subject to change, it's not recommended for production. Try it out and share your feedback on GitHub.

The experimental `serverComponentsHmrCache` option...

typescript

```typescript
import type { NextConfig } from 'next'

const nextConfig: NextConfig = {
  experimental: {
    serverComponentsHmrCache: false, // defaults to true
  },
}

export default nextConfig
```

Parameters

Name	Type	Description	Default	Required
`serverComponentsHmrCache`	boolean	When set to false, disables the HMR cache for fetch responses in Server Components during development.	true	No

staleTimes

API

Source: https://nextjs.org/docs/app/api-reference/config/next-config-js/staleTimes

This page documents the experimental `staleTimes` configuration option in `next.config.js`, which controls client cache revalidation times for dynamic and static page segments.

staleTimes

This feature is currently experimental and subject to change, it's not recommended for production. Try it out and share your feedback on...

next.config.js

```javascript
/** @type {import('next').NextConfig} */
const nextConfig = {
  experimental: {
    staleTimes: {
      dynamic: 30,
      static: 180,
    },
  },
}

module.exports = nextConfig
```

Version History

Version	Changes
`v15.0.0`	The `dynamic` `staleTimes` default changed from 30s to 0s.
`v14.2.0`	Experimental `staleTimes` introduced.

Parameters

Name	Type	Description	Default	Required
`dynamic`	number	Used when the page is neither statically generated nor fully prefetched (e.g. with `prefetch={true}`). Default: 0 seconds (not cached).	0	No
`static`	number	Used for statically generated pages, or when the `prefetch` prop on `Link` is set to `true`, or when calling `router.prefetch`. Default: 5 minutes.	300 (5 minutes)	No

next.config.js: staticGeneration*

`API`

Source: https://nextjs.org/docs/app/api-reference/config/next-config-js/staticGeneration

Explains the experimental `staticGeneration*` options in `next.config.js` that configure the Static Generation process, including retry count, maximum concurrency per worker, and minimum pages per...

staticGeneration*

This feature is currently experimental and subject to change, it's not recommended for production. Try it out and share your feedback on GitHub. The `staticGeneration*` options allow you to configure...

next.config.ts

```ts
import type { NextConfig } from 'next'

const nextConfig: NextConfig = {
  experimental: {
    staticGenerationRetryCount: 1,
    staticGenerationMaxConcurrency: 8,…
```

Config Options

The following options are available:

- `staticGenerationRetryCount`: The number of times to retry a failed page generation before failing the build.
- `staticGenerationMaxConcurrency`: The maximum...

Parameters

Name	Type	Description	Default	Required
`staticGenerationRetryCount`	number	The number of times to retry a failed page generation before failing the build.		No
`staticGenerationMaxConcurrency`	number	The maximum number of pages to be processed per worker.		No
`staticGenerationMinPagesPerWorker`	number	The minimum number of pages to be processed before starting a new worker.		No

next.config.js: supportsImmutableAssets | Next.js

API

Source: https://nextjs.org/docs/app/api-reference/config/next-config-js/supportsImmutableAssets

This page describes the `supportsImmutableAssets` configuration option in next.config.js, used primarily by adapter authors to opt out of immutable static assets when an adapter has enabled support...

supportsImmutableAssets

Attention: This option is primarily intended for adapter authors. App developers should only set it when troubleshooting adapter-specific issues.

**Enabling...

next.config.js

```
/** @type {import('next').NextConfig} */
const nextConfig = {
  supportsImmutableAssets: false,
}

module.exports = nextConfig
```

Version History

Version	Changes
`v16.3.0`	Added support for immutable static assets.

Parameters

Name	Type	Description	Default	Required
`supportsImmutableAssets`	boolean	Set to `false` to opt out of immutable static assets when your adapter has enabled support for them. If the adapter has not enabled this feature, this option has no effect.		No

transpilePackages

API

Source: https://nextjs.org/docs/app/api-reference/config/next-config-js/transpilePackages

Use transpilePackages to compile and bundle a dependency instead of treating it as untouched runtime code. This page explains when the option is needed and provides configuration examples.

Overview

Use `transpilePackages` to compile and bundle a dependency instead of treating it as untouched runtime code. Values are package names, including scoped names like `@scope/pkg`. Paths and glob...

next.config.js

```
/** @type {import('next').NextConfig} */
const nextConfig = {
  transpilePackages: ['package-name', '@scope/pkg'],
}

module.exports = nextConfig
```

When you need it

Turbopack transpiles workspace packages (npm, pnpm, or Yarn workspaces) in your monorepo automatically under both routers. Webpack does the same for the App Router. Add a package to...

Version History

Version	Changes
`v13.0.0`	`transpilePackages` added.

Parameters

Name	Type	Description	Default	Required
`transpilePackages`	string[]	Package names to compile and bundle a dependency instead of treating it as untouched runtime code. Values are package names, including scoped names like `@scope/pkg`. Paths and glob patterns are not...		No

turbopack

API

Source: https://nextjs.org/docs/app/api-reference/config/next-config-js/turbopack

The `turbopack` option lets you customize Turbopack to transform different files and change how modules are resolved.

Overview

The `turbopack` option lets you customize Turbopack to transform different files and change how modules are resolved.

> ***Good to know** : The `turbopack` option...*

next.config.ts

```ts
import type { NextConfig } from 'next'

const nextConfig: NextConfig = {
  turbopack: {
    // ...
  },
}

export default nextConfig
```

Reference

Options

The following options are available for the `turbopack` configuration:

Option	Description
`root`	Sets the application root directory. Should be an absolute path.
`rules` ...	

Supported loaders

The following loaders have been tested to work with Turbopack's webpack loader implementation, but many other webpack loaders should work as well even if not listed here:

-...

Missing Webpack loader features

Turbopack uses the `loader-runner` library to execute webpack loaders, which provides most of the standard loader API. However, some features are not...

Examples

Root directory

Turbopack uses the root directory to resolve modules. Files outside of the project root are not resolved.

The reason files are not resolved outside of the project root is to improve cache...

next.config.js

```
const path = require('path')
module.exports = {
  turbopack: {
    root: path.join(__dirname, '..'),
  },
}
```

Configuring webpack loaders

If you need loader support beyond what's built in, many webpack loaders already work with Turbopack. There are currently some limitations:

- Only a core subset of the webpack loader API is...

next.config.js

```
module.exports = {
  turbopack: {
    rules: {
      '*.svg': {
        loaders: ['@svgr/webpack'],
        as: '*.js',
      },
    },
  },
}
```

next.config.js

```
module.exports = {
  turbopack: {
    rules: {
      '*.svg': {
        loaders: [
          {
            loader: '@svgr/webpack',
            options: {
              icon: true,
            },…
```

Advanced webpack loader conditions

You can further restrict where a loader runs using the advanced `condition` syntax:

- Supported boolean operators are `{all: [...]}`, `{any: [...]}` and `{not: ...}`.
- Supported customizable...

next.config.js

```
module.exports = {
  turbopack: {
    rules: {
      // '*' will match all file paths, but we restrict where our
      // rule runs with a condition.
      '*': {
        condition: {
          all:…
```

next.config.js

```
module.exports = {
  turbopack: {
    rules: {
      '*.svg': [
        {
          condition: 'browser',
          loaders: ['@svgr/webpack'],
          as: '*.js',
        },
        {…
```

Module types

You can set the module type directly without using a loader. This is useful for changing how files are processed, similar to webpack's `type`...

next.config.js

```
module.exports = {
  turbopack: {
    rules: {
      '*.svg': {
        type: 'asset',
      },
    },
  },
}
```

app/page.tsx

```
import svgUrl from './icon.svg'

export default function Page() {
  return <img src={svgUrl} alt="Icon" />
}
```

Inline loader configuration with import attributes

You can apply a Turbopack loader to an individual import using the `with` clause (import attributes). This is specified per-import rather than globally via `turbopack.rules`.

This is useful when you...

app/page.tsx

```
// Apply a raw loader to import a .txt file as a JavaScript module
import rawText from '../data.txt' with { turbopackLoader: 'raw-loader', turbopackAs: '*.js' }

export default function Page() {…
```

app/page.tsx

```
import value from '../data.js' with { turbopackLoader: 'string-replace-loader', turbopackLoaderOptior
```

Resolving aliases

Turbopack can be configured to modify module resolution through aliases, similar to webpack's `resolve.alias` configuration.

To...

next.config.js

```
module.exports = {
  turbopack: {
    resolveAlias: {
      underscore: 'lodash',
      mocha: { browser: 'mocha/browser-entry.js' },
    },
  },
}
```

Resolving custom extensions

Turbopack can be configured to resolve modules with custom extensions, similar to webpack's `resolve.extensions` configuration.

To...

next.config.js

```
module.exports = {
  turbopack: {
    resolveExtensions: ['.mdx', '.tsx', '.ts', '.jsx', '.js', '.mjs', '.json'],
  },
}
```

Debug IDs

Turbopack can be configured to generate debug IDs in JavaScript bundles and source maps.

To configure debug IDs, use the `debugIds` ...

next.config.js

```
module.exports = {
  turbopack: {
    debugIds: true,
  },
}
```

Version History

Version	Changes
`16.2.0`	`turbopackLoader` import attributes were added.
`16.2.0`	`turbopack.rules.*.type` was added.
`16.2.0`	...

Parameters

Name	Type	Description	Default	Required
`root`	string	Sets the application root directory. Should be an absolute path.		No
`rules`	object	List of supported webpack loaders to apply when running with Turbopack.		No
`resolveAlias`	object	Map aliased imports to modules to load in their place.		No
`resolveExtensions`	array	List of extensions to resolve when importing files.		No
`debugIds`	boolean	Enable generation of debug IDs in JavaScript bundles and source maps.		No

turbopackChunking

API

Source: https://nextjs.org/docs/app/api-reference/config/next-config-js/turbopackChunking

Configuration options for Turbopack's production JavaScript chunker in next.config.js, including size thresholds, component chunks, and heuristics for merging chunks.

Overview

This feature is currently experimental and subject to change, it's not recommended for production. Try it out and share your feedback on…

next.config.ts

```
import type { NextConfig } from 'next'

const nextConfig = {
  experimental: {
    turbopackChunking: {
      minChunkSize: 50000,
      maxChunkCountPerGroup: 40,
      maxMergeChunkSize: 200000,…
```

Size Thresholds

The following options control how aggressively Turbopack merges chunks and how large a chunk is allowed to grow. Sizes are in bytes of uncompressed, unminified code (roughly 5x the size of the…

Component Chunks

Producing component chunks is an experimental feature that aims to give you the initial page load benefits of merged chunks without sacrificing reusability. This feature lets the runtime dynamically…

Heuristics

These change the assumptions the chunker makes when weighing whether merging two chunks is worth it.

- `firstPageLoadPriority` (a number between `0` and `1`): how heavily to weight the benefit…

Parameters

Name	Type	Description	Default	Required
`minChunkSize`	number	Turbopack will avoid creating more than one chunk smaller than this size by merging small chunks into larger ones. Raising this number will produce fewer, larger chunks. Meanwhile, lowering it will...	50000	No
`maxChunkCountPerGroup`	number	Turbopack will not emit more than this many chunks per chunk group (eg. a route or a dynamic import). Lowering this number will lead to more aggressive merging and less network requests per-page....	40	No
`maxMergeChunkSize`	number	Turbopack never merges a chunk larger than this size with other chunks. This keeps the code in large chunks from being duplicated across multiple large output chunks.	200000	No
`generateComponentChunks`	boolean	When enabled, each merged production chunk also emits its constituent component chunks alongside it, so the browser runtime can fetch individual component chunks.	False	No
`minComponentChunkSize`	number	Component chunks smaller than this size are folded into a single component instead of being emitted on their own, to avoid producing many tiny chunks.	20000	No
`firstPageLoadPriority`	number	How heavily to weight the benefit of merging chunks for a single page load. Higher values merge more eagerly. If you don't have a better value, your site's bounce rate is a good approximation.		No
`priorityRoutes`	array<RegExp>	Routes that are often the first page a visitor lands on (e.g. the homepage). Their client-side bundles are merged more eagerly to reduce the single-route request		No

		cost, at the cost of extra requests...		
`priorityBoost`	number	A multiplier on the single-request probability of priorityRoutes routes. Higher values merge those routes' bundles more aggressively.	1.5	No
`requestCost`	number	The estimated cost of an additional request, in bytes of uncompressed, unminified code. Larger values bias toward fewer, larger chunks and fewer requests overall.	200000	No

Turbopack FileSystem Caching

`API`

Source: https://nextjs.org/docs/app/api-reference/config/next-config-js/turbopackFileSystemCache

This page describes the Turbopack FileSystem Cache configuration options in Next.js, which enable caching of Turbopack's work across dev and build commands to speed up subsequent runs.

Usage

Turbopack FileSystem Cache enables Turbopack to reduce work across `next dev` or `next build` commands. When enabled, Turbopack will save and restore data under the `.next` directory between runs,...

typescript

```typescript
import type { NextConfig } from 'next'

const nextConfig: NextConfig = {
  experimental: {
    turbopackFileSystemCacheForDev: true,
    turbopackFileSystemCacheForBuild: true,
  },
}

export…
```

Options

- `turbopackFileSystemCacheForDev` (default: `true`): caches Turbopack's work for `next dev` in `.next/dev/cache/turbopack`. Restarting the dev server reuses the previous compilation.

-...

Build environments

The build cache lives in `.next/cache`. Builds only get faster when that directory is restored before each build.

- **Self-hosted builds** : reuse the same working directory between builds....

Version History

Version	Changes
`v16.3.0`	FileSystem caching is enabled by default for builds
`v16.1.0`	FileSystem caching is enabled by default for development
`v16.0.0`	Beta...

Parameters

Name	Type	Description	Default	Required
`turbopackFileSystemCacheForDev`	boolean	caches Turbopack's work for `next dev` in `.next/dev/cache/turbopack`. Restarting the dev server reuses the previous compilation.	true	No
`turbopackFileSystemCacheForBuild`	boolean	caches Turbopack's work for `next build` in `.next/cache/turbopack`. Subsequent builds start warm. See Build environments.	true	No

turbopack.ignoreIssue

`API`

Source: https://nextjs.org/docs/app/api-reference/config/next-config-js/turbopackIgnoreIssue

The `turbopack.ignoreIssue` option allows you to filter out specific Turbopack errors and warnings so they do not appear in the CLI output or the error overlay. This is useful for suppressing known...

turbopack.ignoreIssue

The `turbopack.ignoreIssue` option allows you to filter out specific Turbopack errors and warnings so they do not appear in the CLI output or the error overlay. This is useful for suppressing known...

Usage

```
next.config.ts
```

```
import type { NextConfig } from 'next'

const nextConfig: NextConfig = {
  turbopack: {
    ignoreIssue: [
      {
        path: '**/vendor/**',
      },
    ],
  },
}

export default nextConfig
```

Options

Each rule in the `ignoreIssue` array is an object with the following fields:

Field	Type	Required	Description
`path`	`string | RegExp`	Yes	Matches against...

path

A **glob pattern** (when a string) or **regular expression** that matches against the file path where the issue originated.

next.config.js

```
module.exports = {
  turbopack: {
    ignoreIssue: [
      // Glob pattern: suppress issues from any file under vendor/
      { path: '**/vendor/**' },
      // RegExp: suppress issues from files…
```

title

An **exact string match** (when a string) or **regular expression** that matches against the issue title.

next.config.js

```
module.exports = {
  turbopack: {
    ignoreIssue: [
      {
        path: '**/src/**',
        title: 'Module not found',
      },
    ],
  },
}
```

description

An **exact string match** (when a string) or **regular expression** that matches against the issue description.

next.config.js

```
module.exports = {
  turbopack: {
    ignoreIssue: [
      {
        path: '**/src/**',
        description: /Cannot find module 'optional-dep'/,
      },
    ],
  },
}
```

Examples

Suppressing warnings for optional dependencies

If your code uses `try/catch` around an optional `require()` call, Turbopack may report a "Module not found" warning. You can suppress it:

next.config.ts

```
import type { NextConfig } from 'next'

const nextConfig: NextConfig = {
  turbopack: {
    ignoreIssue: [
      {
        path: '**/lib/optional-feature/**',
        title: 'Module not found',…
```

Combining multiple rules

You can specify multiple rules to suppress different issues:

next.config.js

```
module.exports = {
  turbopack: {
    ignoreIssue: [
      { path: '**/vendor/**' },
      { path: '**/legacy/**', title: 'Module not found' },
      { path: /generated\//, description: /expected…
```

Version History

Version	Changes
`v16.2.0`	`turbopack.ignoreIssue` introduced.

Next Steps

Learn more about Turbopack configuration.

- turbopack: Configure Next.js with Turbopack-specific options
- [Turbopack: Turbopack is an…

Parameters

Name	Type	Description	Default	Required
`path`	string | RegExp	Matches against the file path of the issue. A glob pattern (when a string) or a regular expression.		Yes
`title`	string | RegExp	Matches against the issue title. An exact string match (when a string) or a regular expression.		No
`description`	string | RegExp	Matches against the issue description. An exact string match (when a string) or a regular expression.		No

next.config.js: turbopackMemoryEviction | Next.js

`API`

Source: https://nextjs.org/docs/app/api-reference/config/next-config-js/turbopackMemoryEviction

Explains the experimental `turbopackMemoryEviction` option in Next.js, which controls whether Turbopack reclaims memory when the FileSystem cache is enabled, including its three possible settings and...

Usage

`turbopackMemoryEviction` controls whether Turbopack reclaims memory while the persistent (FileSystem) cache is enabled. After Turbopack writes a snapshot of its cache to disk, it can 'evict' the...

next.config.ts

```
import type { NextConfig } from 'next'

const nextConfig: NextConfig = {
  experimental: {
    turbopackMemoryEviction: 'auto',
  },
}

export default nextConfig
```

Version Changes

Version	Changes
`v16.3.0`	`turbopackMemoryEviction` released as experimental.

Parameters

Name	Type	Description	Default	Required
`turbopackMemoryEviction`	string | boolean	Controls whether Turbopack reclaims memory while the persistent (FileSystem) cache is enabled. Accepts `false`, `'auto'`, or `'full'`.	'auto'	No

turbopackRustReactCompiler

API

Source: https://nextjs.org/docs/app/api-reference/config/next-config-js/turbopackRustReactCompiler

The `experimental.turbopackRustReactCompiler` option enables the native Rust version of the React Compiler, running it directly inside Turbopack as native code instead of through Node.js with the...

turbopackRustReactCompiler

This feature is currently experimental and subject to change, it's not recommended for production. Try it out and share your feedback on GitHub.

The `experimental.turbopackRustReactCompiler` option...

next.config.ts

```ts
import type { NextConfig } from 'next'

const nextConfig: NextConfig = {
  // Enable the React Compiler
  reactCompiler: true,
  experimental: {
    // Use the Rust port instead of the Babel…
```

Good to know

- This option requires `reactCompiler` to be enabled. It selects which implementation runs, but does not turn the compiler on by itself.
- This option is only supported with Turbopack. Using it with...

Version History

Version	Changes
`v16.3.0`	Introduced the experimental `turbopackRustReactCompiler` option for the native Rust React Compiler.

typedRoutes

API

Source: https://nextjs.org/docs/app/api-reference/config/next-config-js/typedRoutes

Configuration option for Next.js that enables statically typed links. Requires TypeScript in the project.

typedRoutes

Note: This option has been marked as stable, so you should use `typedRoutes` instead of `experimental.typedRoutes`.

Support for statically typed links. This feature requires using TypeScript in your...

next.config.js

```
/** @type {import('next').NextConfig} */
const nextConfig = {
  typedRoutes: true,
}

module.exports = nextConfig
```

typescript

API

Source: https://nextjs.org/docs/app/api-reference/config/next-config-js/typescript

Configure TypeScript behavior with the `typescript` option in `next.config.js`.

typescript

Configure TypeScript behavior with the `typescript` option in `next.config.js`:

next.config.js

```
module.exports = {
  typescript: {
    ignoreBuildErrors: false,
    tsconfigPath: 'tsconfig.json',
  },
}
```

Options

ignoreBuildErrors

Next.js fails your **production build** (`next build`) when TypeScript errors are present in your project.

If you'd like Next.js to dangerously produce production code even when your application has...

next.config.js

```
module.exports = {
  typescript: {
    // !! WARN !!
    // Dangerously allow production builds to successfully complete even if
    // your project has type errors.
    // !! WARN !!…
```

tsconfigPath

Use a different TypeScript configuration file for builds or tooling:

See the TypeScript configuration page for more details.

next.config.js

```
module.exports = {
  typescript: {
    tsconfigPath: 'tsconfig.build.json',
  },
}
```

Parameters

Name	Type	Description	Default	Required
ignoreBuildErrors	boolean	Allow production builds to complete even with TypeScript errors.	false	No
tsconfigPath	string	Path to a custom `tsconfig.json` file.	'tsconfig.json'	No

useLightningcss

API

Source: https://nextjs.org/docs/app/api-reference/config/next-config-js/useLightningcss

Experimental support for using Lightning CSS with webpack, and configuration options for controlling CSS feature transpilation.

useLightningcss

This feature is currently experimental and subject to change, it's not recommended for production. Try it out and share your feedback on GitHub.

Experimental support for using Lightning CSS with...

next.config.ts

```
import type { NextConfig } from 'next'

const nextConfig: NextConfig = {
  experimental: {
    useLightningcss: false, // default, ignored on Turbopack
  },
}

export default nextConfig
```

lightningCssFeatures

By default, Lightning CSS decides which CSS features to transpile based on your browserslist targets. The `lightningCssFeatures` option lets you override this by forcing specific features to always...

next.config.ts

```
import type { NextConfig } from 'next'

const nextConfig: NextConfig = {
  experimental: {
    useLightningcss: true,
    lightningCssFeatures: {
      // Always transpile these features, even if…
```

Options

Option	Type	Description
`include`	`string[]`	Features to always transpile, regardless of browser targets.
`exclude`	`string[]`	Features to never transpile,...

Available features

Individual features:

Feature name	Description
`nesting`	CSS Nesting
`not-selector-list`	`:not` with multiple selectors
`dir-selector`	`:dir()` selector
...	

Version History

Version	Changes
`16.2.0`	`lightningCssFeatures` added.
`15.1.0`	Support for `useSwcCss` was removed from Turbopack.
`14.2.0`	Turbopack's default CSS processor was...

Parameters

Name	Type	Description	Default	Required
`useLightningcss`	boolean	Enables Lightning CSS for webpack. Defaults to false. Ignored on Turbopack.	false	No
`lightningCssFeatures.include`	string[]	Features to always transpile, regardless of browser targets.		No
`lightningCssFeatures.exclude`	string[]	Features to never transpile, even when browser targets would require them.		No

next.config.js: useTypeScriptCli | Next.js

API

Source: https://nextjs.org/docs/app/api-reference/config/next-config-js/useTypeScriptCli

Documents the experimental `experimental.useTypeScriptCli` configuration option, which controls whether `next build` uses the project-local `tsc` CLI or the TypeScript JavaScript compiler API for...

useTypeScriptCli

This feature is currently experimental and subject to change, it's not recommended for production. Try it out and share your feedback on GitHub.

By...

bash

```
pnpm add -D typescript@^7
```

next.config.ts

```ts
import type { NextConfig } from 'next'

const nextConfig: NextConfig = {
  experimental: {
    useTypeScriptCli: false,
  },
}

export default nextConfig
```

Behavior

- Next.js continues to generate `next-env.d.ts` and route types and to apply its recommended `tsconfig` settings before running the checker.
- TypeScript diagnostics are printed directly from `tsc`....

Parameters

Name	Type	Description	Default	Required
`experimental.useTypeScriptCli`	boolean	When true, `next build` uses the project-local `tsc` CLI command for type checking. When false, Next.js loads the TypeScript JavaScript compiler API instead. This is experimental; setting it to false...	true	No

Custom Webpack Config

API

Source: https://nextjs.org/docs/app/api-reference/config/next-config-js/webpack

Describes how to customize the webpack configuration in Next.js using the `webpack` option in `next.config.js`, including the properties available in the second argument and an example of extending...

Custom Webpack Config

Good to know : changes to webpack config are not covered by semver so proceed at your own risk

Before continuing to add custom webpack configuration to your application make sure Next.js doesn't...

next.config.js

```
module.exports = {
  webpack: (
    config,
    { buildId, dev, isServer, defaultLoaders, nextRuntime, webpack }
  ) => {
    // Important: return the modified config
    return config
  },
}
```

javascript

```
// Example config for adding a loader that depends on babel-loader
// This source was taken from the @next/mdx plugin source:
//…
```

nextRuntime

Notice that `isServer` is `true` when `nextRuntime` is `"edge"` or `"nodejs"`, `nextRuntime` `"edge"` is currently for proxy and Server Components in edge runtime only.

Parameters

Name	Type	Description	Default	Required
`config`	Object	The webpack configuration object that should be modified and returned.		Yes
`buildId`	String	The build id, used as a unique identifier between builds.		Yes
`dev`	Boolean	Indicates if the compilation will be done in development.		Yes
`isServer`	Boolean	It's `true` for server-side compilation, and `false` for client-side compilation.		Yes
`nextRuntime`	String | undefined	The target runtime for server-side compilation; either `"edge"` or `"nodejs"`, it's `undefined` for client-side compilation.		Yes
`defaultLoaders`	Object	Default loaders used internally by Next.js. Contains `babel`: `Object` - Default `babel-loader` configuration.		Yes
`webpack`	Object	The webpack instance used by Next.js.		Yes

use cache: private

`API`

Source: https://nextjs.org/docs/app/api-reference/directives/use-cache-private

This page documents the 'use cache: private' directive in Next.js, which allows functions to access runtime request APIs within a cached scope while caching results only in the browser's memory and...

Overview

The `'use cache: private'` directive allows functions to access runtime request APIs like `cookies()`, `headers()`, and `searchParams` within a cached scope. However, results are **never stored on...

Usage

To use `'use cache: private'`, enable the `cacheComponents` flag in your `next.config.ts` file:

Then add `'use cache: private'` to...

```
next.config.ts
```

```ts
import type { NextConfig } from 'next'

const nextConfig: NextConfig = {
  cacheComponents: true,
}

export default nextConfig
```

Basic example

In this example, we demonstrate that you can access cookies within a `'use cache: private'` scope:

> ***Good to know*** *: The* `stale` *time must be at least 30 seconds for per-link prefetching to work,...*

app/product/[id]/page.tsx

```tsx
import { Suspense } from 'react'
import { cookies } from 'next/headers'
import { cacheLife, cacheTag } from 'next/cache'

export async function generateStaticParams() {
  return [{ id: '1'…
```

Request APIs allowed in private caches

The following request-specific APIs can be used inside `'use cache: private'` functions:

API	Allowed in `use cache`	Allowed in `'use cache: private'`
`cookies()`	No	...

Version History

Version	Changes
`v16.0.0`	`"use cache: private"` is enabled with the Cache Components feature.

Related

View related API references.

- ### use cache - Learn how to use the "use cache" directive to cache data in your Next.js application.
- [###...

Guide

Creating an Adapter

GUIDE

Source: https://nextjs.org/docs/app/api-reference/adapters/creating-an-adapter

This page explains how to create a custom adapter for Next.js by implementing the NextAdapter interface, including the interface definition and a minimal example.

Creating an Adapter

An adapter is a module that exports an object implementing the `NextAdapter` interface.

The interface can be imported from the `next` package:

The interface is defined as follows:

typescript

```typescript
import type { NextAdapter } from 'next'
```

typescript

```typescript
type Route = {
  source?: string
  sourceRegex: string
  destination?: string
  headers?: Record<string, string>
  has?: RouteHas[]
  missing?: RouteHas[]
  status?: number
  priority?: boolean
}…
```

Basic Adapter Structure

Here's a minimal adapter example:

my-adapter.js

```javascript
/** @type {import('next').NextAdapter} */
const adapter = {
  name: 'my-custom-adapter',

  async modifyConfig(config, { phase }) {
    // Modify the Next.js config based on the build phase
    if…
```

Implementing PPR in an Adapter

GUIDE

Source: https://nextjs.org/docs/app/api-reference/adapters/implementing-ppr-in-an-adapter

This page explains how to implement Partial Prerendering (PPR) in a Next.js adapter, covering build-time seeding of fallback shells and postponed state, runtime streaming of cached shells with...

Overview

For partially prerendered app routes, `onBuildComplete` gives you the data needed to seed and resume PPR:

- `outputs.prerenders[].fallback.filePath`: path to the generated fallback shell (for...

1. Seed shell + postponed state at build time

my-adapter.ts

```
import { readFile } from 'node:fs/promises'

async function seedPprEntries(outputs: AdapterOutputs) {
  for (const prerender of outputs.prerenders) {
    const fallback = prerender.fallback
    if…
```

2. Runtime flow: serve cached shell and resume in background

At request time, you can stream a single response that is the concatenation of:

- cached HTML shell stream
- resumed render stream (generated after invoking `handler` with postponed state)

text

```
Client
  | GET /ppr-route
  v
Adapter Router
  |
  |-- read cached shell + postponedState ---> Platform Cache
  |<------------- cache hit -----------------|
  |
  |-- create responseStream =…
```

3. Update cache with `requestMeta.onCacheEntryV2`

`requestMeta.onCacheEntryV2` is called when a response cache entry is looked up or generated. Use it to persist updated shell/postponed data.

- `requestMeta.onCacheEntry` still works, but is...

my-adapter.ts

```
await handler(req, res, {
  waitUntil,
  requestMeta: {
    postponed: cachedPprEntry?.postponedState,
    onCacheEntryV2: async (cacheEntry, meta) => {
      if (cacheEntry.value?.kind ===…
```

text

```
Entrypoint (handler)
  | onCacheEntryV2(cacheEntry, { url })
  v
requestMeta.onCacheEntryV2 callback
  |
  |-- if APP_PAGE ---> persist html + postponedState + headers ---> Platform Cache
  |
  '--…
```

Testing Adapters

GUIDE

Source: https://nextjs.org/docs/app/api-reference/adapters/testing-adapters

Next.js provides a test harness for validating adapters, including end-to-end tests for deployment. This page describes the environment variables and script contracts required for custom deploy,...

Testing Adapters

Next.js provides a test harness for validating adapters. Running the end-to-end tests for deployment.

Example GitHub Actions workflow:

The test harness looks for these environment variables:

-...

.github/workflows/test-e2e-deploy.yml

```
name: test-e2e-deploy

on:
  workflow_dispatch:
    inputs:
      nextjsRef:
        description: 'Next.js repo ref (branch/tag/SHA)'
        default: 'canary'
        type: string
  # schedule:
  #…
```

Custom deploy script contract

The deploy script `NEXT_TEST_DEPLOY_SCRIPT_PATH` is executed with `cwd` set to the isolated temporary app created by the Next.js test harness.

The deploy script must follow this contract:

- Exit…

scripts/e2e-deploy.sh

```
#!/usr/bin/env bash
set -euo pipefail

# Install the adapter, build the app, and deploy or start it.
node -e "
const…
```

Custom logs script contract

The logs script `NEXT_TEST_DEPLOY_LOGS_SCRIPT_PATH` is executed with `cwd` set to the isolated temporary app created by the Next.js test harness.

Additionally it receives `NEXT_TEST_DIR` and…

scripts/e2e-logs.sh

```
#!/usr/bin/env bash
set -euo pipefail

if [ -f ".adapter-build.log" ]; then
  cat ".adapter-build.log"
fi

if [ -f ".adapter-server.log" ]; then
  echo "=== .adapter-server.log ==="
  cat…
```

Custom cleanup script contract

The cleanup script `NEXT_TEST_CLEANUP_SCRIPT_PATH` is executed with `cwd` set to the isolated temporary app created by the Next.js test harness.

Additionally it receives `NEXT_TEST_DIR` and...

Adapters: Use Cases | Next.js

GUIDE

Source: https://nextjs.org/docs/app/api-reference/adapters/use-cases

Common use cases for Next.js adapters, including deployment platform integration, asset processing, monitoring, custom bundling, build validation, and route generation.

Use Cases

Common use cases for adapters include:

- **Deployment Platform Integration** : Automatically configure build outputs for specific hosting platforms
- **Asset Processing** : Transform or optimize...

Overview

Next.js Docs

OVERVIEW

Source: https://nextjs.org/docs

This page is the main documentation index for Next.js, providing an overview of what Next.js is, how to use the docs, the difference between App Router and Pages Router, prerequisites, accessibility...

What is Next.js?

Next.js is a React framework for building full-stack web applications. You use React Components to build user interfaces, and Next.js for additional features and optimizations. It also automatically...

How to use the docs

The docs are organized into 3 sections:

- Getting Started: Step-by-step tutorials to help you create a new application and learn the core Next.js features.
- Guides: Tutorials on specific use cases,...

App Router and Pages Router

Next.js has two different routers:

- App Router: The newer router that supports new React features like Server Components.
- Pages Router: The original router, still supported and being...

React version handling

The App Router and Pages Router handle React versions differently:

- App Router: Uses React canary releases built-in, which include all the stable React 19 changes, as well as newer features being...

Pre-requisite knowledge

Our documentation assumes some familiarity with web development. Before getting started, it'll help if you're comfortable with:

- HTML
- CSS
- JavaScript
- React

If you're new to React or need a...

Accessibility

For the best experience when using a screen reader, we recommend using Firefox and NVDA, or Safari and VoiceOver.

Join our Community

If you have questions about anything related to Next.js, you're always welcome to ask our community on GitHub Discussions, Discord, X (Twitter), and Reddit.

Next Steps

Create your first application and learn the core Next.js features.

App Router

OVERVIEW

Source: https://nextjs.org/docs/app

Overview of the Next.js App Router, a file-system based router that leverages React's latest features such as Server Components, Suspense, and Server Functions.

App Router

The App Router is a file-system based router that uses React's latest features such as Server Components, Suspense, and Server Functions.

Next Steps

Learn the fundamentals of building an App Router project, from installation to layouts, navigation, server and client components.

- Installation: Learn how to create a new Next.js application with...

API Reference

OVERVIEW

Source: https://nextjs.org/docs/app/api-reference

Index page for Next.js App Router API references, listing categories such as Directives, Components, File-system conventions, Functions, Configuration, CLI, Adapters, Edge Runtime, and Turbopack.

Directives

Directives are used to modify the behavior of your Next.js application.

Components

API Reference for Next.js built-in components.

File-system conventions

API Reference for Next.js file-system conventions.

Functions

API Reference for Next.js Functions and Hooks.

Configuration

Learn how to configure Next.js applications.

CLI

API Reference for the Next.js Command Line Interface (CLI) tools.

Adapters

Build deployment adapters for Next.js platforms and infrastructure.

Edge Runtime

API Reference for the Edge Runtime.

Turbopack

Turbopack is an incremental bundler optimized for JavaScript and TypeScript, written in Rust, and built into Next.js.

Adapters

`OVERVIEW`

Source: https://nextjs.org/docs/app/api-reference/adapters

Overview of Next.js deployment adapter documentation, covering configuration, creation, API reference, testing, routing, PPR, runtime integration, entrypoints, output

types, routing information, use...

Adapters

Use this section to build and validate deployment adapters that integrate with the Next.js build and runtime model.

Configuration

Configure `adapterPath` or `NEXT_ADAPTER_PATH` to use a custom deployment adapter.

Creating an Adapter

Create an adapter module that implements the `NextAdapter` interface.

API Reference

Reference for `modifyConfig` and `onBuildComplete` in the `NextAdapter` interface.

Testing Adapters

Validate adapters with the Next.js compatibility test harness and custom lifecycle scripts.

Routing with @next/routing

Use `@next/routing` to apply Next.js route matching behavior in adapters.

Implementing PPR in an Adapter

Implement Partial Prerendering support in an adapter using fallback output and cache hooks.

Runtime Integration

Understand how build-time adapters and runtime cache interfaces work together.

Invoking Entrypoints

Invoke Node.js and Edge build entrypoints with adapter runtime context.

Output Types

Reference for all build output types exposed to adapters.

Routing Information

Reference for routing phases and route fields exposed in `onBuildComplete`.

Use Cases

Common patterns and examples for deployment adapter implementations.

Supporting Immutable Static Assets

Support immutable static assets in an adapter

Components

OVERVIEW

Source: https://nextjs.org/docs/app/api-reference/components

This page provides an index of Next.js built-in components for optimizing fonts, forms, images, links, and scripts.

Components

This page is also available as Markdown: request this page's URL with an `Accept: text/markdown` header. For an index of Next.js documentation, see /docs/llms.txt.

- [Font:…

Configuration

OVERVIEW

Source: https://nextjs.org/docs/app/api-reference/config

Overview page for Next.js configuration options, linking to documentation for next.config.js, TypeScript, and ESLint.

Configuration

next.config.js: Learn how to configure your application with next.config.js.

TypeScript: Next.js provides a TypeScript-first development experience for building your React application.

ESLint:…

Reference

Adapters: Supporting Immutable Static Assets | Next.js

REFERENCE

Source: https://nextjs.org/docs/app/api-reference/adapters/immutable-static-assets

Explains how Next.js adapters can support immutable static assets, including the `supportsImmutableAssets` config flag, the shared-namespace runtime behavior of immutable assets, content-hash...

Supporting Immutable Static Assets

See `config.supportsImmutableAssets` for end-user-facing information about this feature.

When...

Adapter Implementation

You need to:

- In the `modifyConfig`, set the `config.supportsImmutableAssets` property to `true` (if it's not already set to `false` by the user) to signal that you support deploying immutable...

my-adapter.js

```
/** @type {import('next').NextAdapter} */
const adapter = {
  name: 'my-custom-adapter',

  async modifyConfig(config, { phase }) {
    if (phase === 'phase-production-build') {…
```

Adapters: Invoking Entrypoints | Next.js

REFERENCE

Source: https://nextjs.org/docs/app/api-reference/adapters/invoking-entrypoints

This page describes how build output entrypoints use a `handler(..., ctx)` interface for Node.js and Edge runtimes, including how adapters can invoke entrypoints directly and use `requestMeta`...

Invoking Entrypoints

Build output entrypoints use a `handler(..., ctx)` interface, with runtime-specific request/response types.

Node.js runtime (`runtime: 'nodejs'`)

Node.js entrypoints use the following interface:

When invoking Node.js entrypoints directly, adapters can pass helpers directly on `requestMeta` instead of relying on internals. Some of the...

typescript

```
handler(
  req: IncomingMessage,
  res: ServerResponse,
  ctx: {
    waitUntil?: (promise: Promise<void>) => void
    requestMeta?: RequestMeta
  }
): Promise<void>
```

javascript

```
await handler(req, res, {
  requestMeta: {
    // Relative path from process.cwd() to the Next.js project directory.
    relativeProjectDir: '.',
    // Optional hostname used by route handlers when…
```

Edge runtime (`runtime: 'edge'`) (deprecated)

> *The Edge Runtime is deprecated. New routes should use the Node.js runtime.*

Edge entrypoints use the following interface:

The shape is aligned around...

typescript

```
handler(
  request: Request,
  ctx: {
    waitUntil?: (prom: Promise<void>) => void
    signal?: AbortSignal
    requestMeta?: RequestMeta
  }
): Promise<Response>
```

typescript

```
{
  modulePath: string // Absolute path to the module registered in the edge runtime
  entryKey: string // Canonical key used by the edge entry registry
  handlerExport: string // Export name to…
```

javascript

```
const entry = await globalThis._ENTRIES[output.edgeRuntime.entryKey]
const handler = entry[output.edgeRuntime.handlerExport]
await handler(request, ctx)
```

API Reference: CLI | Next.js

REFERENCE

Source: https://nextjs.org/docs/app/api-reference/cli

Overview of the two Next.js Command Line Interface (CLI) tools: create-next-app and next, with links to their API references.

CLI

Next.js comes with **two** Command Line Interface (CLI) tools:

- `create-next-app`: Quickly create a new Next.js application using the default template or an...

create-next-app

Create Next.js apps using one command with the create-next-app CLI.

next CLI

Learn how to run and build your application with the Next.js CLI.

Components: Image Component | Next.js

REFERENCE

Source: https://nextjs.org/docs/app/api-reference/components/image

Extraction fallback content.

Components: Image Component | Next.js

This page is also available as Markdown: request this page's URL with an `Accept: text/markdown` header. For an index of Next.js documentation , see /docs/llms.txt.Copy page

Image Component

Last updated May 4, 2026

The Next.js Image component extends the HTML `<img>` element for automatic image optimization.

app/page.js

Reference

Props

The following props are available:

Prop	Example	Type	Status
src	src="/profile.png"	String	Required
alt	alt="Picture of the author"	String	Required
width	width={500}	Integer (px)	-
height	height={500}	Integer (px)	-
fill	fill={true}	Boolean	-
loader	loader={imageLoader}	Function	-
sizes	sizes="(max-width: 768px) 100vw, 33vw"	String	-
quality	quality={80}	Integer (1-100)	-
preload	preload={true}	Boolean	-
placeholder	placeholder="blur"	String	-
style	style={{objectFit: "contain"}}	Object	-
onLoadingComplete	onLoadingComplete={img => done())}	Function	Deprecated
onLoad	onLoad={event => done())}	Function	-
onError	onError(event => fail()}	Function	-
loading	loading="lazy"	String	-
blurDataURL	blurDataURL="data:image/jpeg..."	String	-
unoptimized	unoptimized={true}	Boolean	-
overrideSrc	overrideSrc="/seo.png"	String	-
decoding	decoding="async"	String	-

src

The source of the image. Can be one of the following:

An internal path string.

An absolute external URL (must be configured with remotePatterns).

A static import.

> ***Good to know*** *: For security reasons, the Image Optimization API using the default loader will not forward headers when fetching the* `src` *image. If the* `src` *image requires authentication, consider using the unoptimized property to disable Image Optimization.*

`alt`

The `alt` property is used to describe the image for screen readers and search engines. It is also the fallback text if images have been disabled or an error occurs while loading the image.

It should contain text that could replace the image without changing the meaning of the page. It is not meant to supplement the image and should not repeat information that is already provided in the captions above or below the image.

If the image is purely decorative or not intended for the user, the `alt` property should be an empty string (`alt=""`).

> *Learn more about image accessibility guidelines.*

`width` **and** `height`

The `width` and `height` properties represent the intrinsic image size in pixels. This property is used to infer the correct **aspect ratio** used by browsers to reserve space for the image and avoid layout shift during loading. It does not determine the *rendered size* of the image, which is controlled by CSS.

You **must** set both `width` and `height` properties unless:

- The image is statically imported.
- The image has the `fill` property

If the height and width are unknown, we recommend using the `fill` property.

`fill`

A boolean that causes the image to expand to the size of the parent element.

Positioning :

- The parent element **must** assign `position: "relative"` , `"fixed"` , `"absolute"` .
- By default, the `<img>` element uses `position: "absolute"` .

Object Fit :

If no styles are applied to the image, the image will stretch to fit the container. You can use `objectFit` to control cropping and scaling.

- `"contain"`: The image will be scaled down to fit the container and preserve aspect ratio.
- `"cover"`: The image will fill the container and be cropped.

> *Learn more about `position` and `object-fit`.*

`loader`

A custom function used to generate the image URL. The function receives the following parameters, and returns a URL string for the image:

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

> ***Good to know** : Using props like `onLoad`, which accept a function, requires using Client Components to serialize the provided function.*

Alternatively, you can use the loaderFile configuration in `next.config.js` to configure every instance of `next/image` in your application, without passing a prop.

`sizes`

Define the sizes of the image at different breakpoints. Used by the browser to choose the most appropriate size from the generated `srcset`.

`sizes` should be used when:

- The image is using the `fill` prop
- CSS is used to make the image responsive

If `sizes` is missing, the browser assumes the image will be as wide as the viewport (`100vw`). This can cause unnecessarily large images to be downloaded.

In addition, `sizes` affects how `srcset` is generated:

- Without `sizes`: Next.js generates a limited `srcset` (e.g. 1x, 2x), suitable for fixed-size images.
- With `sizes`: Next.js generates a full `srcset` (e.g. 640w, 750w, etc.), optimized for responsive layouts.

> *Learn more about `srcset` and `sizes` on web.dev and mdn.*

`quality`

An integer between `1` and `100` that sets the quality of the optimized image. Higher values increase file size and visual fidelity. Lower values reduce file size but may affect sharpness.

If you've configured qualities in `next.config.js`, the value must match one of the allowed entries.

> ***Good to know** : If the original image is already low quality, setting a high quality value will increase the file size without improving appearance.*

`style`

Allows passing CSS styles to the underlying image element.

> ***Good to know** : If you're using the `style` prop to set a custom width, be sure to also set `height: 'auto'` to preserve the image's aspect ratio.*

`preload`

A boolean that indicates if the image should be preloaded.

- `true`: Preloads the image by inserting a `<link>` in the `<head>`.
- `false`: Does not preload the image.

When to use it:

- The image is the Largest Contentful Paint (LCP) element.
- The image is above the fold, typically the hero image.
- You want to begin loading the image in the `<head>`, before its discovered later in the `<body>`.

When not to use it:

- When you have multiple images that could be considered the Largest Contentful Paint (LCP) element depending on the viewport.
- When the `loading` property is used.
- When the `fetchPriority` property is used.

In most cases, you should use `loading="eager"` or `fetchPriority="high"` instead of `preload`.

`priority`

Starting with Next.js 16, the `priority` property has been deprecated in favor of the `preload` property in order to make the behavior clear.

`loading`

Controls when the image should start loading.

- `lazy`: Defer loading the image until it reaches a calculated distance from the viewport.
- `eager`: Load the image immediately, regardless of its position in the page.

Use `eager` only when you want to ensure the image is loaded immediately.

> *Learn more about the `loading` attribute.*

`placeholder`

Specifies a placeholder to use while the image is loading, improving the perceived loading performance.

- `empty`: No placeholder while the image is loading.
- `blur`: Use a blurred version of the image as a placeholder. Must be used with the `blurDataURL` property.
- `data:image/...`: Uses the Data URL as the placeholder.

Examples:

- `blur` placeholder
- Shimmer effect with data URL `placeholder` prop
- Color effect with `blurDataURL` prop

> *Learn more about the `placeholder` attribute.*

`blurDataURL`

A Data URL to be used as a placeholder image before the image successfully loads. Can be automatically set or used with the `placeholder="blur"` property.

The image is automatically enlarged and blurred, so a very small image (10px or less) is recommended.

Automatic

If `src` is a static import of a `jpg`, `png`, `webp`, or `avif` file, `blurDataURL` is added automatically—unless the image is animated.

Manually set

If the image is dynamic or remote, you must provide `blurDataURL` yourself. To generate one, you can use:

- A online tool like png-pixel.com
- A library like Plaiceholder

A large blurDataURL may hurt performance. Keep it small and simple.

Examples:

- Default `blurDataURL` prop
- Color effect with `blurDataURL` prop

`onLoad`

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

The callback function will be called with one argument, the event which has a `target` that references the underlying `<img>` element.

> ***Good to know*** *: Using props like `onLoad`, which accept a function, requires using Client Components to serialize the provided function.*

`onError`

A callback function that is invoked if the image fails to load.

> ***Good to know*** *: Using props like `onError`, which accept a function, requires using Client Components to serialize the provided function.*

`unoptimized`

A boolean that indicates if the image should be optimized. This is useful for images that do not benefit from optimization such as small images (less than 1KB), vector images (SVG), or animated images (GIF).

- `true`: The source image will be served as-is from the `src` instead of changing quality, size, or format.
- `false`: The source image will be optimized.

Since Next.js 12.3.0, this prop can be assigned to all images by updating `next.config.js` with the following configuration:

next.config.js

`overrideSrc`

When providing the `src` prop to the `<Image>` component, both the `srcset` and `src` attributes are generated automatically for the resulting `<img>`.

input.js output.html

In some cases, it is not desirable to have the `src` attribute generated and you may wish to override it using the `overrideSrc` prop.

For example, when upgrading an existing website from `<img>` to `<Image>`, you may wish to maintain the same `src` attribute for SEO purposes such as image ranking or avoiding recrawl.

input.js output.html

`decoding`

A hint to the browser indicating if it should wait for the image to be decoded before presenting other content updates or not.

- `async`: Asynchronously decode the image and allow other content to be rendered before it completes.
- `sync`: Synchronously decode the image for atomic presentation with other content.
- `auto`: No preference. The browser chooses the best approach.

> *Learn more about the `decoding` attribute.*

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.

Deprecated props

`onLoadingComplete`

> ***Warning** : Deprecated in Next.js 14, use `onLoad` instead.*

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

The callback function will be called with one argument, a reference to the underlying `<img>` element.

> ***Good to know** : Using props like `onLoadingComplete`, which accept a function, requires using Client Components to serialize the provided function.*

Configuration options

You can configure the Image Component in `next.config.js`. The following options are available:

`localPatterns`

Use `localPatterns` in your `next.config.js` file to allow images from specific local paths to be optimized and block all others.

next.config.js

The example above will ensure the `src` property of `next/image` must start with `/assets/images/` and must not have a query string. Attempting to optimize any other path will respond with `400` Bad Request error.

> ***Good to know*** *: Omitting the* `search` *property allows all search parameters which could allow malicious actors to optimize URLs you did not intend. Try using a specific value like* `search: '?v=2'` *to ensure an exact match.*

`remotePatterns`

Use `remotePatterns` in your `next.config.js` file to allow images from specific external paths and block all others. This ensures that only external images from your account can be served.

next.config.js

You can also configure `remotePatterns` using the object:

next.config.js

The example above will ensure the `src` property of `next/image` must start with `https://example.com/account123/` and must not have a query string. Any other protocol, hostname, port, or unmatched path will respond with `400` Bad Request.

Wildcard Patterns:

Wildcard patterns can be used for both `pathname` and `hostname` and have the following syntax:

- `*` match a single path segment or subdomain
- `**` match any number of path segments at the end or subdomains at the beginning. This syntax does not work in the middle of the pattern.

next.config.js

This allows subdomains like `image.example.com`. Query strings and custom ports are still blocked.

> ***Good to know*** *: When omitting* `protocol`*,* `port`*,* `pathname`*, or* `search` *then the wildcard* `**` *is implied. This is not recommended because it may allow malicious actors to optimize urls you did not*

> *intend.*

Query Strings :

You can also restrict query strings using the `search` property:

next.config.js

The example above will ensure the `src` property of `next/image` must start with `https://assets.example.com` and must have the exact query string `?v=1727111025337`. Any other protocol or query string will respond with `400` Bad Request.

Note that any allowed `remotePatterns` that respond with a redirect will follow the redirect from the remote image server without validating `remotePatterns` again on the redirect location. You can reduce or disable redirects by configuring maximumRedirects.

`loaderFile`

`loaderFiles` allows you to use a custom image optimization service instead of Next.js.

next.config.js

The path must be relative to the project root. The file must export a default function that returns a URL string:

my/image/loader.js

Example:

- Custom Image Loader Configuration

> *Alternatively, you can use the `loader` prop to configure each instance of `next/image`.*

`path`

If you want to change or prefix the default path for the Image Optimization API, you can do so with the `path` property. The default value for `path` is `/_next/image`.

next.config.js

`deviceSizes`

`deviceSizes` allows you to specify a list of device width breakpoints. These widths are used when the `next/image` component uses `sizes` prop to ensure the correct image is served for the user's device.

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

next.config.js

`imageSizes`

`imageSizes` allows you to specify a list of image widths. These widths are concatenated with the array of device sizes to form the full array of sizes used to generate image srcset.

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

next.config.js

`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`.

`qualities`

`qualities` allows you to specify a list of image quality values.

If not configuration is provided, the default below is used:

next.config.js

> ***Good to know*** *: This field is required starting with Next.js 16 because unrestricted access could allow malicious actors to optimize more qualities than you intended.*

You can add more image qualities to the allowlist, such as the following:

next.config.js

In the example above, only four qualities are allowed: 25, 50, 75, and 100.

If the `quality` prop does not match a value in this array, the closest allowed value will be used.

If the REST API is visited directly with a quality that does not match a value in this array, the server will return a 400 Bad Request response.

`formats`

`formats` allows you to specify a list of image formats to be used.

next.config.js

Next.js automatically detects the browser's supported image formats via the request's `Accept` header in order to determine the best output format.

If the `Accept` header 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), it will use the original image's format.

You can enable AVIF support, which will fallback to the original format of the src image if the browser does not support AVIF:

next.config.js

You can also enable both AVIF and WebP formats together. AVIF will be preferred for browsers that support it, with WebP as a fallback:

next.config.js

> ***Good to know** :*
>
> - *We still recommend using WebP for most use cases.*
> - *AVIF generally takes 50% 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, but subsequent requests that are cached will be faster.*
> - *When using multiple formats, Next.js will cache each format separately. This means increased storage requirements compared to using a single format, as both AVIF and WebP versions of images will be stored for different browser support.*
> - *If you self-host with a Proxy/CDN in front of Next.js, you must configure the Proxy to forward the `Accept` header.*

`minimumCacheTTL`

`minimumCacheTTL` allows you to 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`.

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

next.config.js

You can increase the TTL to reduce the number of revalidations and potentially lower cost:

next.config.js

The expiration (or rather Max Age) of the optimized image is defined by either the `minimumCacheTTL` or the upstream image `Cache-Control` header, whichever is larger.

If you need to change the caching behavior per image, you can configure `headers` to set the `Cache-Control` header on the upstream image (e.g. `/some-asset.jpg`, not `/_next/image` itself).

There is no mechanism to invalidate the cache at this time, so its best to keep `minimumCacheTTL` low. Otherwise you may need to manually change the `src` prop or delete the cached file `<distDir>/cache/images`.

`disableStaticImages`

`disableStaticImages` allows you to disable static image 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`:

next.config.js

`maximumRedirects`

The default image optimization loader will follow HTTP redirects when fetching remote images up to 3 times.

next.config.js

For your convenience, these redirects do not need to satisfy remotePatterns.

You can configure the number of redirects to follow when fetching remote images. Setting the value to `0` will disable following redirects.

next.config.js

`maximumDiskCacheSize`

The default image optimization loader will write optimized images to disk so subsequent requests can be served faster from the disk cache.

You can configure the maximum disk cache size in bytes, for example 500 MB:

next.config.js

You can also disable the disk cache entirely by setting the value to `0`.

next.config.js

If no value is configured, the default behavior is to check the current available disk space once during startup and use 50%.

When the disk cache exceeds the configured size, the least recently used optimized images will be deleted until the cache is under the limit again.

Alternatively, you can implement your own cache handler using `cacheHandler` which will ignore the `maximumDiskCacheSize` configuration.

`maximumResponseBody`

The default image optimization loader will fetch source images up to 50 MB in size.

next.config.js

If you know all your source images are small, you can protect memory constrained servers by reducing this to a smaller value such as 5 MB.

next.config.js

`dangerouslyAllowLocalIP`

In rare cases when self-hosting Next.js on a private network, you may want to allow optimizing images from local IP addresses on the same network. This is not recommended for most users because it could allow malicious users to access content on your internal network.

By default, the value is false.

next.config.js

If you need to optimize remote images hosted elsewhere in your local network, you can set the value to true.

next.config.js

This might be necessary when hosting Next.js in a VPC with split-horizon DNS and you receive status 400 Bad Request. Only enable once you understand the SSRF risk.

`dangerouslyAllowSVG`

`dangerouslyAllowSVG` allows you to serve SVG images.

next.config.js

By default, Next.js does not optimize SVG images for a few reasons:

- SVG is a vector format meaning it can be resized losslessly.
- SVG has many of the same features as HTML/CSS, which can lead to vulnerabilities without proper Content Security Policy (CSP) headers.

We recommend using the `unoptimized` prop when the `src` prop is known to be SVG. This happens automatically when `src` ends with `".svg"`.

In addition, it is strongly recommended to also set `contentDispositionType` to force the browser to download the image, as well as `contentSecurityPolicy` to prevent scripts embedded in the image from executing.

next.config.js

`contentDispositionType`

`contentDispositionType` allows you to configure the `Content-Disposition` header.

next.config.js

`contentSecurityPolicy`

`contentSecurityPolicy` allows you to configure the `Content-Security-Policy` header for images. This is particularly important when using `dangerouslyAllowSVG` to prevent scripts embedded in the image from executing.

next.config.js

By default, the loader sets the `Content-Disposition` header to `attachment` for added protection since the API can serve arbitrary remote images.

The default value is `attachment` which forces the browser to download the image when visiting directly. This is particularly important when `dangerouslyAllowSVG` is true.

You can optionally configure `inline` to allow the browser to render the image when visiting directly, without downloading it.

Deprecated configuration options

`domains`

> ***Warning*** *: Deprecated since Next.js 14 in favor of strict* `remotePatterns` *in order to protect your application from malicious users.*

Similar to `remotePatterns`, the `domains` configuration can be used to provide a list of allowed hostnames for external images. However, the `domains` configuration does not support wildcard pattern matching and it cannot restrict protocol, port, or pathname.

Since most remote image servers are shared between multiple tenants, it's safer to use `remotePatterns` to ensure only the intended images are optimized.

Below is an example of the `domains` property in the `next.config.js` file:

next.config.js

Functions

`getImageProps`

The `getImageProps` function can be used to get the props that would be passed to the underlying `<img>` element, and instead pass them to another component, style, canvas, etc.

This also avoid calling React `useState()` so it can lead to better performance, but it cannot be used with the `placeholder` prop because the placeholder will never be removed.

Known browser bugs

This `next/image` component uses browser native lazy loading, which may fallback to eager loading for older browsers before Safari 15.4. When using the blur-up placeholder, older browsers before Safari 12 will fallback to empty placeholder. When using styles with `width` / `height` of `auto`, it is possible to cause Layout Shift on older browsers before Safari 15 that don't preserve the aspect ratio. For more details, see this MDN video.

- Safari 15 - 16.3 display a gray border while loading. Safari 16.4 fixed this issue. Possible solutions:
- Use CSS `@supports (font: -apple-system-body) and (-webkit-appearance: none) { img[loading="lazy"] { clip-path: inset(0.6px) } }`
- Use `loading="eager"` if the image is above the fold

- Firefox 67+ displays a white background while loading. Possible solutions:
- Enable AVIF `formats`
- Use `placeholder`

Examples

Styling images

Styling the Image component is similar to styling a normal `<img>` element, but there are a few guidelines to keep in mind:

Use `className` or `style`, not `styled-jsx`. In most cases, we recommend using the `className` prop. This can be an imported CSS Module, a global stylesheet, etc.

You can also use the `style` prop to assign inline styles.

When using `fill`, the parent element must have `position: relative` or `display: block`. This is necessary for the proper rendering of the image element in that layout mode.

You cannot use styled-jsx because it's scoped to the current component (unless you mark the style as `global`).

Responsive images with a static export

When you import a static image, Next.js automatically sets its width and height based on the file. You can make the image responsive by setting the style:

<img src="/_next/image?

python

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

export default function Page() {
  return (
    <Image
      src="/profile.png"
      width={500}
      height={500}
      alt="Picture of the author"
    />
  )
}
```

text

```
<Image src="/profile.png" />
```

text

```
<Image src="https://example.com/profile.png" />
```

python

```
import profile from './profile.png'

export default function Page() {
  return <Image src={profile} />
}
```

text

```
<Image src="/profile.png" width={500} height={500} />
```

text

```
<Image src="/profile.png" fill={true} />
```

python

```
'use client'

import Image from 'next/image'

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

export default function Page() {
  return (
    <Image
      loader={imageLoader}
      src="me.png"
      alt="Picture of the author"
      width={500}
      height={500}
    />
  )
}
```

python

```
import Image from 'next/image'

export default function Page() {
  return (
    <div className="grid-element">
      <Image
        fill
        src="/example.png"
        sizes="(max-width: 768px) 100vw, (max-width: 1200px) 50vw, 33vw"
      />
    </div>
  )
}
```

scilab

```
// Default quality is 75
<Image quality={75} />
```

gdscript

```
const imageStyle = {
  borderRadius: '50%',
  border: '1px solid #fff',
  width: '100px',
  height: 'auto',
}

export default function ProfileImage() {
  return <Image src="..." style={imageStyle} />
}
```

gdscript

```
// Default preload is false
<Image preload={false} />
```

gdscript

```
// Defaults to lazy
<Image loading="lazy" />
```

scilab

```
// defaults to empty
<Image placeholder="empty" />
```

text

```
<Image placeholder="blur" blurDataURL="..." />
```

text

```
<Image onLoad={(e) => console.log(e.target.naturalWidth)} />
```

text

```
<Image onError={(e) => console.error(e.target.id)} />
```

python

```
import Image from 'next/image'

const UnoptimizedImage = (props) => {
  // Default is false
  return <Image {...props} unoptimized />
}
```

gdscript

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

text

```
<Image src="/profile.jpg" />
```

scdoc

```
<img
  srcset="
    /_next/image?url=%2Fprofile.jpg&w=640&q=75 1x,
    /_next/image?url=%2Fprofile.jpg&w=828&q=75 2x
  "
  src="/_next/image?url=%2Fprofile.jpg&w=828&q=75"
/>
```

text

```
<Image src="/profile.jpg" overrideSrc="/override.jpg" />
```

scdoc

```
<img
  srcset="
    /_next/image?url=%2Fprofile.jpg&w=640&q=75 1x,
    /_next/image?url=%2Fprofile.jpg&w=828&q=75 2x
  "
  src="/override.jpg"
/>
```

scilab

```
// Default is async
<Image decoding="async" />
```

text

```
'use client'

<Image onLoadingComplete={(img) => console.log(img.naturalWidth)} />
```

gdscript

```
module.exports = {
  images: {
    localPatterns: [
      {
        pathname: '/assets/images/**',
        search: '',
      },
    ],
  },
}
```

gdscript

```
module.exports = {
  images: {
    remotePatterns: [new URL('https://example.com/account123/**')],
  },
}
```

gdscript

```
module.exports = {
  images: {
    remotePatterns: [
      {
        protocol: 'https',
        hostname: 'example.com',
        port: '',
        pathname: '/account123/**',
        search: '',
      },
    ],
  },
}
```

gdscript

```
module.exports = {
  images: {
    remotePatterns: [
      {
        protocol: 'https',
        hostname: '**.example.com',
        port: '',
        search: '',
      },
    ],
  },
}
```

gdscript

```
module.exports = {
  images: {
    remotePatterns: [
      {
        protocol: 'https',
        hostname: 'assets.example.com',
        search: '?v=1727111025337',
      },
    ],
  },
}
```

gdscript

```
module.exports = {
  images: {
    loader: 'custom',
    loaderFile: './my/image/loader.js',
  },
}
```

gdscript

```
'use client'

export default function myImageLoader({ src, width, quality }) {
  return `https://example.com/${src}?w=${width}&q=${quality || 75}`
}
```

gdscript

```
module.exports = {
  images: {
    path: '/my-prefix/_next/image',
  },
}
```

gdscript

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

gdscript

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

gdscript

```
module.exports = {
  images: {
    qualities: [75],
  },
}
```

gdscript

```
module.exports = {
  images: {
    qualities: [25, 50, 75, 100],
  },
}
```

gdscript

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

gdscript

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

gdscript

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

gdscript

```
module.exports = {
  images: {
    minimumCacheTTL: 14400, // 4 hours
  },
}
```

gdscript

```
module.exports = {
  images: {
    minimumCacheTTL: 2678400, // 31 days
  },
}
```

gdscript

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

gdscript

```
module.exports = {
  images: {
    maximumRedirects: 3,
  },
}
```

gdscript

```
module.exports = {
  images: {
    maximumRedirects: 0,
  },
}
```

gdscript

```
module.exports = {
  images: {
    maximumDiskCacheSize: 500_000_000,
  },
}
```

gdscript

```
module.exports = {
  images: {
    maximumDiskCacheSize: 0,
  },
}
```

gdscript

```
module.exports = {
  images: {
    maximumResponseBody: 50_000_000,
  },
}
```

gdscript

```
module.exports = {
  images: {
    maximumResponseBody: 5_000_000,
  },
}
```

gdscript

```
module.exports = {
  images: {
    dangerouslyAllowLocalIP: false,
  },
}
```

gdscript

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

gdscript

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

text

```
<Image src="/my-image.svg" unoptimized />
```

gdscript

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

gdscript

```
module.exports = {
  images: {
    contentDispositionType: 'inline',
  },
}
```

gdscript

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

gdscript

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

python

```
import { getImageProps } from 'next/image'

const { props } = getImageProps({
  src: 'https://example.com/image.jpg',
  alt: 'A scenic mountain view',
  width: 1200,
  height: 800,
})

function ImageWithCaption() {
  return (
    <figure>
      <img {...props} />
      <figcaption>A scenic mountain view</figcaption>
    </figure>
  )
}
```

python

```
import styles from './styles.module.css'

export default function MyImage() {
  return <Image className={styles.image} src="/my-image.png" alt="My Image" />
}
```

gdscript

```
export default function MyImage() {
  return (
    <Image style={{ borderRadius: '8px' }} src="/my-image.png" alt="My Image" />
  )
}
```

xml+django

```
<div style={{ position: 'relative' }}>
  <Image fill src="/my-image.png" alt="My Image" />
</div>
```

python

```
import Image from 'next/image'
import mountains from '../public/mountains.jpg'

export default function Responsive() {
  return (
    <div style={{ display: 'flex', flexDirection: 'column' }}>
      <Image
        alt="Mountains"
        // Importing an image will
        // automatically set the width and height
        src={mountains}
        sizes="100vw"
        // Make the image display full width
        // and preserve its aspect ratio
        style={{
          width: '100%',
          height: 'auto',
        }}
      />
    </div>
  )
}
```

python

```
import Image from 'next/image'

export default function Page({ photoUrl }) {
  return (
    <Image
      src={photoUrl}
      alt="Picture of the author"
      sizes="100vw"
      style={{
        width: '100%',
        height: 'auto',
      }}
      width={500}
      height={300}
    />
  )
}
```

python

```
import Image from 'next/image'
import mountains from '../public/mountains.jpg'

export default function Fill() {
  return (
    <div
      style={{
        display: 'grid',
        gridGap: '8px',
        gridTemplateColumns: 'repeat(auto-fit, minmax(400px, auto))',
      }}
    >
      <div style={{ position: 'relative', width: '400px' }}>
        <Image
          alt="Mountains"
          src={mountains}
          fill
          sizes="(min-width: 808px) 50vw, 100vw"
          style={{
            objectFit: 'cover', // cover, contain, none
          }}
        />
      </div>
      {/* And more images in the grid... */}
    </div>
  )
}
```

python

```
import Image from 'next/image'
import mountains from '../public/mountains.jpg'

export default function Background() {
  return (
    <Image
      alt="Mountains"
      src={mountains}
      placeholder="blur"
      quality={100}
      fill
      sizes="100vw"
      style={{
        objectFit: 'cover',
      }}
    />
  )
}
```

python

```
import Image from 'next/image'

export default function Page() {
  return (
    <Image
      src="https://s3.amazonaws.com/my-bucket/profile.png"
      alt="Picture of the author"
      width={500}
      height={500}
    />
  )
}
```

gdscript

```
module.exports = {
  images: {
    remotePatterns: [
      {
        protocol: 'https',
        hostname: 's3.amazonaws.com',
        port: '',
        pathname: '/my-bucket/**',
        search: '',
      },
    ],
  },
}
```

gas

```
.imgDark {
  display: none;
}

@media (prefers-color-scheme: dark) {
  .imgLight {
    display: none;
  }
  .imgDark {
    display: unset;
  }
}
```

python

```
import styles from './theme-image.module.css'
import Image, { ImageProps } from 'next/image'

type Props = Omit<ImageProps, 'src' | 'preload' | 'loading'> & {
  srcLight: string
  srcDark: string
}

const ThemeImage = (props: Props) => {
  const { srcLight, srcDark, ...rest } = props

  return (
    <>
      <Image {...rest} src={srcLight} className={styles.imgLight} />
      <Image {...rest} src={srcDark} className={styles.imgDark} />
    </>
  )
}
```

python

```
import { getImageProps } from 'next/image'

export default function Home() {
  const common = { alt: 'Art Direction Example', sizes: '100vw' }
  const {
    props: { srcSet: desktop },
  } = getImageProps({
    ...common,
    width: 1440,
    height: 875,
    quality: 80,
    src: '/desktop.jpg',
  })
  const {
    props: { srcSet: mobile, ...rest },
  } = getImageProps({
    ...common,
    width: 750,
    height: 1334,
    quality: 70,
    src: '/mobile.jpg',
  })

  return (
    <picture>
      <source media="(min-width: 1000px)" srcSet={desktop} />
      <source media="(min-width: 500px)" srcSet={mobile} />
      <img {...rest} style={{ width: '100%', height: 'auto' }} />
    </picture>
  )
}
```

python

```
import { getImageProps } from 'next/image'

function getBackgroundImage(srcSet = '') {
  const imageSet = srcSet
    .split(', ')
    .map((str) => {
      const [url, dpi] = str.split(' ')
      return `url("${url}") ${dpi}`
    })
    .join(', ')
  return `image-set(${imageSet})`
}

export default function Home() {
  const {
    props: { srcSet },
  } = getImageProps({ alt: '', width: 128, height: 128, src: '/img.png' })
  const backgroundImage = getBackgroundImage(srcSet)
  const style = { height: '100vh', width: '100vw', backgroundImage }

  return (
    <main style={style}>
      <h1>Hello World</h1>
    </main>
  )
}
```

ESLint Plugin

REFERENCE

Source: https://nextjs.org/docs/app/api-reference/config/eslint

Explains how to configure ESLint for Next.js using eslint-config-next, including setup, available rules, examples, and migrating existing ESLint configurations.

ESLint Plugin

Next.js provides an ESLint configuration package, `eslint-config-next`, that makes it easy to catch common issues in your application. It includes...

Setup ESLint

Get linting working quickly with the ESLint CLI (flat config):

bash

```
pnpm add -D eslint eslint-config-next
```

eslint.config.mjs

```
import { defineConfig, globalIgnores } from 'eslint/config'
import nextVitals from 'eslint-config-next/core-web-vitals'

const eslintConfig = defineConfig([
  ...nextVitals,
  // Override default…
```

bash

```
pnpm exec eslint .
```

Reference

The `eslint-config-next` package includes the `recommended` rule-sets from the following ESLint plugins:

- `eslint-plugin-react` -...

Rules

The `@next/eslint-plugin-next` rules included are:

Enabled in recommended config	Rule	Description
	...	

Examples

Specifying a root directory within a monorepo

If you're using `@next/eslint-plugin-next` in a project where Next.js isn't installed in your root directory (such as a monorepo), you can tell `@next/eslint-plugin-next` where to find your Next.js...

eslint.config.mjs

```
import { defineConfig } from 'eslint/config'
import eslintNextPlugin from '@next/eslint-plugin-next'

const eslintConfig = defineConfig([
  {
    files: ['**/*.{js,jsx,ts,tsx}'],
    plugins: {…
```

Disabling rules

If you would like to modify or disable any rules provided by the supported plugins (`react`, `react-hooks`, `next`), you can directly change them using the `rules` property in your...

eslint.config.mjs

```
import { defineConfig, globalIgnores } from 'eslint/config'
import nextVitals from 'eslint-config-next/core-web-vitals'

const eslintConfig = defineConfig([
  ...nextVitals,
  {
    rules: {…
```

With Core Web Vitals

Enable the `eslint-config-next/core-web-vitals` configuration in your ESLint config.

`eslint-config-next/core-web-vitals` upgrades certain lint rules in `@next/eslint-plugin-next` from warnings to...

eslint.config.mjs

```
import { defineConfig, globalIgnores } from 'eslint/config'
import nextVitals from 'eslint-config-next/core-web-vitals'

const eslintConfig = defineConfig([
  ...nextVitals,
  // Override default…
```

With TypeScript

In addition to the Next.js ESLint rules, `create-next-app --typescript` will also add TypeScript-specific lint rules with `eslint-config-next/typescript` to your config:

Those rules are based on...

eslint.config.mjs

```
import { defineConfig, globalIgnores } from 'eslint/config'
import nextVitals from 'eslint-config-next/core-web-vitals'
import nextTs from 'eslint-config-next/typescript'

const eslintConfig =…
```

With Prettier

ESLint also contains code formatting rules, which can conflict with your existing Prettier setup. We recommend including...

bash

```
pnpm add -D eslint-config-prettier
```

eslint.config.mjs

```
import { defineConfig, globalIgnores } from 'eslint/config'
import nextVitals from 'eslint-config-next/core-web-vitals'
import prettier from 'eslint-config-prettier/flat'

const eslintConfig =…
```

Running lint on staged files

If you would like to use ESLint with lint-staged to run the linter on staged git files, add the following to the `.lintstagedrc.js` file in the root of your...

.lintstagedrc.js

```
const path = require('path')

const buildEslintCommand = (filenames) =>
  `eslint --fix ${filenames
    .map((f) => `"${path.relative(process.cwd(), f)}"`)
    .join(' ')}`

module.exports = {…
```

Migrating existing config

If you already have ESLint configured in your application, there are two approaches to integrate Next.js linting rules, depending on your setup.

Using the plugin directly

Use `@next/eslint-plugin-next` directly if you have any of the following already configured:

- Conflicting plugins installed separately or through another config (such as `airbnb` or `react-app`):...

bash

```
pnpm add -D @next/eslint-plugin-next
```

eslint.config.mjs

```
import { defineConfig } from 'eslint/config'
import nextPlugin from '@next/eslint-plugin-next'

const eslintConfig = defineConfig([
  // Your other configurations...
  {
    files:…
```

Adding to existing config

If you're adding Next.js to an existing ESLint setup, spread the Next.js config into your array:

When you spread `...nextConfig`, you're adding multiple config objects that include file patterns,...

eslint.config.mjs

```
import nextConfig from 'eslint-config-next/core-web-vitals'
// Your other config imports...

const eslintConfig = [
  // Your other configurations...
  ...nextConfig,
]

export default eslintConfig
```

Version History

Version	Changes
`v16.0.0`	`next lint` and the `eslint` next.config.js option were removed in favor of the ESLint CLI. A...

next.config.js

REFERENCE

Source: https://nextjs.org/docs/app/api-reference/config/next-config-js

Overview of Next.js configuration via next.config.js, including setup, ECMAScript modules, function-based configuration, TypeScript support, and a list of all available configuration options.

next.config.js

Next.js can be configured through a `next.config.js` file in the root of your project directory (for example, by `package.json`) with a default export.

next.config.js

```
// @ts-check

/** @type {import('next').NextConfig} */
const nextConfig = {
  /* config options here */
}

module.exports = nextConfig
```

ECMAScript Modules

`next.config.js` is a regular Node.js module, not a JSON file. It gets used by the Next.js server and build phases, and it's not included in the browser build.

If you need [ECMAScript...

next.config.mjs

```
// @ts-check

/**
 * @type {import('next').NextConfig}
 */
const nextConfig = {
  /* config options here */
}

export default nextConfig
```

Configuration as a Function

You can also use a function:

next.config.mjs

```
// @ts-check

export default (phase, { defaultConfig }) => {
  /**
   * @type {import('next').NextConfig}
   */
  const nextConfig = {
    /* config options here */
  }
  return nextConfig
}
```

Async Configuration

Since Next.js 12.1.0, you can use an async function:

next.config.js

```
// @ts-check

module.exports = async (phase, { defaultConfig }) => {
  /**
   * @type {import('next').NextConfig}
   */
  const nextConfig = {
    /* config options here */
  }
  return nextConfig
}
```

Phase

`phase` is the current context in which the configuration is loaded. You can see the [available phases] (https://github.com/vercel/next.js/blob/5e6b008b561caf2710ab7be63320a3d549474a5b/packages/next/sh...

next.config.js

```
// @ts-check

const { PHASE_DEVELOPMENT_SERVER } = require('next/constants')

module.exports = (phase, { defaultConfig }) => {
  if (phase === PHASE_DEVELOPMENT_SERVER) {
    return {
      /*…
```

TypeScript

If you are using TypeScript in your project, you can use `next.config.ts` to use TypeScript in your configuration:

The commented lines are the place where you can put the configs allowed by...

next.config.ts

```
import type { NextConfig } from 'next'

const nextConfig: NextConfig = {
  /* config options here */
}

export default nextConfig
```

Unit Testing (experimental)

Starting in Next.js 15.1, the `next/experimental/testing/server` package contains utilities to help unit test `next.config.js` files.

The `unstable_getResponseFromNextConfig` function runs the...

js

```
import {
  getRedirectUrl,
  unstable_getResponseFromNextConfig,
} from 'next/experimental/testing/server'

const response = await unstable_getResponseFromNextConfig({
  url:…
```

Available configuration options

This page documents all the available configuration options:

- **adapterPath**: Configure a custom adapter for Next.js to hook into the build process.
- **allowedDevOrigins**: Use...

basePath

REFERENCE

Source: https://nextjs.org/docs/app/api-reference/config/next-config-js/basePath

Explains how to use the basePath config option in next.config.js to deploy a Next.js application under a sub-path of a domain.

basePath

To deploy a Next.js application under a sub-path of a domain you can use the `basePath` config option.

`basePath` allows you to set a path prefix for the application. For example, to use `/docs`...

next.config.js

```
module.exports = {
  basePath: '/docs',
}
```

Links

When linking to other pages using `next/link` and `next/router` the `basePath` will be automatically applied.

For example, using `/about` will automatically become `/docs/about` when `basePath` is...

jsx

```jsx
export default function HomePage() {
  return (
    <>
      <Link href="/about">About Page</Link>
    </>
  )
}
```

html

```html
<a href="/docs/about">About Page</a>
```

Images

When using the `next/image` component, you will need to add the `basePath` in front of `src`.

For example, using `/docs/me.png` will properly serve your...

jsx

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

function Home() {
  return (
    <>
      <h1>My Homepage</h1>
      <Image
        src="/docs/me.png"
        alt="Picture of the author"
        width={500}…
```

Parameters

Name	Type	Description	Default	Required
basePath	string	Path prefix for the application. Defaults to an empty string.	''	No

next.config.js: cssChunking

REFERENCE

Source: https://nextjs.org/docs/app/api-reference/config/next-config-js/cssChunking

This page documents the experimental `cssChunking` option in Next.js configuration, which controls how CSS files are split and re-ordered into chunks to improve performance. It covers the available...

cssChunking

This feature is currently experimental and subject to change, it's not recommended for production. Try it out and share your feedback on GitHub.

CSS Chunking is a strategy used to improve the...

next.config.ts

```
import type { NextConfig } from 'next'

const nextConfig = {
  experimental: {
    cssChunking: true, // default
  },
} satisfies NextConfig

export default nextConfig
```

Options

- `true` **(default)** (**webpack and Turbopack**): Next.js will try to merge CSS files whenever possible, determining explicit and implicit dependencies between files from import order to reduce the...

Choosing a strategy

For most applications, the default (`true`) is the right choice in either bundler: it merges CSS to make fewer requests. Reach for another strategy only for a specific reason.

In Turbopack, that...

Debugging what a route actually uses

While some unused CSS is acceptable, and most apps do not need to change anything, it is worth keeping render-blocking CSS in check. Lighthouse flags this as a **Reduce unused CSS** opportunity with...

Balancing requests and grouping

The `graph` strategy groups CSS into shared chunks to cut requests. Turn it on with the string form, which uses the default tuning:

next.config.ts

```
import type { NextConfig } from 'next'

const nextConfig = {
  experimental: {
    cssChunking: 'graph',
  },
} satisfies NextConfig

export default nextConfig
```

Balancing requests and grouping (continued)

To shift that balance, pass an object instead. Both `requestCost` and `weightDistribution` are optional, so include only the one you want to change:

next.config.ts

```
import type { NextConfig } from 'next'

const nextConfig = {
  experimental: {
    cssChunking: {
      type: 'graph',
      requestCost: 100000,
      weightDistribution: 0.1,
    },
  },
}…
```

Balancing requests and grouping (options)

- `requestCost` (default `20000`): the estimated cost, in bytes, of each additional CSS request. Larger values bias toward fewer, larger shared chunks, and fewer requests overall.

-…

How `graph` decides what to merge

Merging CSS into shared chunks is what the default (`true`) already does; `graph` just lets you control where it draws the line between merging and splitting.

Take two routes that share a…

Graph algorithm overview

At a high level, the algorithm works with individual CSS files. It starts from the ordered list of CSS each route imports:

```
/dashboard  → [reset.css, theme.css, layout.css,…
```

next.config.js: deploymentId

REFERENCE

Source: https://nextjs.org/docs/app/api-reference/config/next-config-js/deploymentId

Documentation for the Next.js `deploymentId` configuration option, which sets a deployment identifier for version skew protection and cache busting during rolling deployments. Covers configuration,…

deploymentId

The `deploymentId` option allows you to set an identifier for your deployment. This identifier is used for version skew protection and…

next.config.js

```
module.exports = {
  deploymentId: 'my-deployment-id',
}
```

bash

```
NEXT_DEPLOYMENT_ID=my-deployment-id next build
```

How it works

When a `deploymentId` is configured, Next.js:

- Appends `?dpl=<deploymentId>` to static asset URLs (JavaScript, CSS, images)
- Adds an `x-deployment-id` header to client-side navigation requests -...

Use cases

Rolling deployments

During a rolling deployment, some server instances may be running the new version while others are still running the old version. Without a deployment ID, users might receive a mix of old and new...

Multi-server environments

When running multiple instances of your Next.js application behind a load balancer, all instances for the same deployment should use the same `deploymentId`.

next.config.js

```
module.exports = {
  deploymentId: process.env.DEPLOYMENT_VERSION || process.env.GIT_SHA,
}
```

Version History

Version	Changes
`v14.1.4`	`deploymentId` stabilized as top-level config option.
`v13.4.10`	`experimental.deploymentId` introduced.

Related

- Self-Hosting - Version Skew
- generateBuildId

distDir

REFERENCE

Source: https://nextjs.org/docs/app/api-reference/config/next-config-js/distDir

Documentation for the distDir configuration option in next.config.js, which lets you specify a custom build directory instead of the default .next folder.

distDir

You can specify a name to use for a custom build directory to use instead of `.next`.

Open `next.config.js` and add the `distDir` config:

Now if you run `next build` Next.js will use `build`...

next.config.js

```
module.exports = {
  distDir: 'build',
}
```

next.config.js: exportPathMap | Next.js

REFERENCE

Source: https://nextjs.org/docs/app/api-reference/config/next-config-js/exportPathMap

This page documents exportPathMap, a legacy Next.js configuration option for specifying a mapping of request paths to page destinations during static export.

exportPathMap

This is a legacy API and no longer recommended. It's still supported for backward compatibility.

> *This feature is exclusive to `next export` and currently **deprecated** in favor of...*

next.config.js

```
module.exports = {
  exportPathMap: async function (
    defaultPathMap,
    { dev, dir, outDir, distDir, buildId }
  ) {
    return {
      '/': { page: '/' },
      '/about': { page: '/about' },…
```

Adding a trailing slash

It is possible to configure Next.js to export pages as `index.html` files and require trailing slashes, `/about` becomes `/about/index.html` and is routable via `/about/`. This was the default...

next.config.js

```
module.exports = {
  trailingSlash: true,
}
```

Customizing the output directory

`next export` will use `out` as the default output directory, you can customize this using the `-o` argument, like so:

```
next export -o outdir
```

```
...
```

bash

```
next export -o outdir
```

httpAgentOptions

REFERENCE

Source: https://nextjs.org/docs/app/api-reference/config/next-config-js/httpAgentOptions

Explains the httpAgentOptions configuration in next.config.js, which controls HTTP Keep-Alive behavior for server-side fetch() calls. Shows how to disable Keep-Alive by setting keepAlive to false.

httpAgentOptions

In Node.js versions prior to 18, Next.js automatically polyfills `fetch()` with undici and enables [HTTP...

next.config.js

```
module.exports = {
  httpAgentOptions: {
    keepAlive: false,
  },
}
```

Parameters

Name	Type	Description	Default	Required
keepAlive	boolean	When set to false, disables HTTP Keep-Alive for all fetch() calls on the server-side.	True	No

next.config.js: images

REFERENCE

Source: https://nextjs.org/docs/app/api-reference/config/next-config-js/images

This page explains how to configure the `images` option in `next.config.js` to use a custom image loader for cloud providers, including example loader implementations for various providers.

images

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 `next.config.js` with the following:

This `loaderFile` must...

next.config.js

```
module.exports = {
  images: {
    loader: 'custom',
    loaderFile: './my/image/loader.js',
  },
}
```

my/image/loader.js

```
'use client'

export default function myImageLoader({ src, width, quality }) {
  return `https://example.com/${src}?w=${width}&q=${quality || 75}`
}
```

Example Loader Configuration

Akamai

```
// Docs: https://techdocs.akamai.com/ivm/reference/test-images-on-demand
export default function akamaiLoader({ src, width, quality }) {
  return `https://example.com/${src}?imwidth=${width}`
}
```

AWS CloudFront

```
// Docs: https://aws.amazon.com/developer/application-security-performance/articles/image-optimization
export default function cloudfrontLoader({ src, width, quality }) {
  const url = new…
```

Cloudinary

```
// Demo: https://res.cloudinary.com/demo/image/upload/w_300,c_limit,q_auto/turtles.jpg
export default function cloudinaryLoader({ src, width, quality }) {
  const params = ['f_auto', 'c_limit',…
```

Cloudflare

```
// Docs: https://developers.cloudflare.com/images/transform-images
export default function cloudflareLoader({ src, width, quality }) {
  const params = [`width=${width}`, `quality=${quality || 75}`,…
```

Contentful

```
// Docs: https://www.contentful.com/developers/docs/references/images-api/
export default function contentfulLoader({ src, width, quality }) {
  const url = new URL(`https://example.com${src}`)…
```

Fastly

```
// Docs: https://developer.fastly.com/reference/io/
export default function fastlyLoader({ src, width, quality }) {
  const url = new URL(`https://example.com${src}`)
  url.searchParams.set('auto',…
```

Gumlet

```
// Docs: https://docs.gumlet.com/reference/image-transform-size
export default function gumletLoader({ src, width, quality }) {
  const url = new URL(`https://example.com${src}`)…
```

ImageEngine

```
// Docs: https://support.imageengine.io/hc/en-us/articles/360058880672-Directives
export default function imageengineLoader({ src, width, quality }) {
  const compression = 100 - (quality || 50)…
```

Imgix

```
// Demo: https://static.imgix.net/daisy.png?format=auto&fit=max&w=300
export default function imgixLoader({ src, width, quality }) {
  const url = new URL(`https://example.com${src}`)
  const params…
```

PixelBin

```
// Doc (Resize): https://www.pixelbin.io/docs/transformations/basic/resize/#width-w
// Doc (Optimise): https://www.pixelbin.io/docs/optimizations/quality/#image-quality-when-delivering
// Doc (Auto…
```

Sanity

```
// Docs: https://www.sanity.io/docs/image-urls
export default function sanityLoader({ src, width, quality }) {
  const prj = 'zp7mbokg'
  const dataset = 'production'
  const url = new…
```

Sirv

```
// Docs: https://sirv.com/help/articles/dynamic-imaging/
export default function sirvLoader({ src, width, quality }) {
  const url = new URL(`https://example.com${src}`)
  const params =…
```

Supabase

```
// Docs: https://supabase.com/docs/guides/storage/image-transformations#nextjs-loader
export default function supabaseLoader({ src, width, quality }) {
  const url = new…
```

Thumbor

```
// Docs: https://thumbor.readthedocs.io/en/latest/
export default function thumborLoader({ src, width, quality }) {
  const params = [`${width}x0`, `filters:quality(${quality || 75})`]
  return…
```

ImageKit.io

```
// Docs: https://imagekit.io/docs/image-transformation
export default function imageKitLoader({ src, width, quality }) {
  const params = [`w-${width}`, `q-${quality || 80}`]
  return…
```

Nitrogen AIO

```
// Docs: https://docs.n7.io/aio/intergrations/
export default function aioLoader({ src, width, quality }) {
  const url = new URL(src, window.location.href)
  const params = url.searchParams
  const…
```

mdxRs

REFERENCE

Source: https://nextjs.org/docs/app/api-reference/config/next-config-js/mdxRs

Explains the experimental `mdxRs` config option in Next.js, which enables the Rust compiler for MDX files when used with `@next/mdx`.

mdxRs

For experimental use with `@next/mdx`. Compiles MDX files using the new Rust compiler.

This feature is currently experimental and subject to change, it's not recommended for production. Try it out…

next.config.js

```
const withMDX = require('@next/mdx')()

/** @type {import('next').NextConfig} */
const nextConfig = {
  pageExtensions: ['ts', 'tsx', 'mdx'],
  experimental: {
    mdxRs: true,
  },
}…
```

next.config.js: output | Next.js

REFERENCE

Source: https://nextjs.org/docs/app/api-reference/config/next-config-js/output

This page documents the `output` option in `next.config.js`, which enables Next.js to trace page dependencies during builds and optionally create a standalone folder containing only the necessary...

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...

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'...

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...

next.config.js

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

Terminal

```
cp -r public .next/standalone/ && cp -r .next/static .next/standalone/.next/
```

Terminal

```
node .next/standalone/server.js
```

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...

packages/web-app/next.config.js

```
const path = require('path')

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

next.config.js

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

next.config.js

```
module.exports = {
  outputFileTracingIncludes: {
    '/products/*': ['src/lib/payments/**/*'],
    '/*': ['src/config/runtime/**/*.json'],
  },
  outputFileTracingExcludes: {
    '/api/*':…
```

next.config.js

```
module.exports = {
  outputFileTracingIncludes: {
    '/*': ['src/i18n/locales/**/*.json'],
  },
}
```

next.config.js

```
const path = require('path')

module.exports = {
  // Trace from the monorepo root
  outputFileTracingRoot: path.join(__dirname, '../../'),
  outputFileTracingIncludes: {
    '/route1':…
```

next.config.js

```
module.exports = {
  outputFileTracingIncludes: {
    '/*': ['node_modules/sharp/**/*', 'node_modules/aws-crt/dist/bin/**/*'],
  },
}
```

Parameters

Name	Type	Description	Default	Required
`output`	string	Set to 'standalone' to create a standalone folder at `.next/standalone` containing only the necessary files for a production deployment, including select files in `node_modules`.		No
`outputFileTracingRoot`	string	Sets the root directory used for output file tracing. Useful in monorepo setups to include files outside the project directory.		No
`outputFileTracingExcludes`	object	An object whose keys are route globs (matched with picomatch against the route path) and whose values are glob patterns resolved from the project root that specify files to exclude from the trace.		No
`outputFileTracingIncludes`	object	An object whose keys are route globs (matched with picomatch against the route path) and whose values are glob patterns resolved from the project root that specify files to include in the trace.		No

outputHashSalt

`REFERENCE`

Source: https://nextjs.org/docs/app/api-reference/config/next-config-js/outputHashSalt

This page documents the `outputHashSalt` option in Next.js configuration, which adds a salt string to output filenames to invalidate cached assets across deployments.

Overview

`outputHashSalt` is an option that incorporates a configurable salt string into every content-addressed output filename (chunks, assets). Changing this value forces all output hashes to change, which...

Configuration

To configure the output hash salt, set `outputHashSalt` in `next.config.js`:

next.config.js

```
/** @type {import('next').NextConfig} */
const nextConfig = {
  outputHashSalt: 'my-deployment-salt',
}

module.exports = nextConfig
```

Bundler Support

This works with both Webpack and Turbopack bundlers.

Environment Variable

The `NEXT_HASH_SALT` environment variable can also be used for the same purpose. When both are set, the values are **concatenated** (`outputHashSalt + NEXT_HASH_SALT`) to form the effective salt....

```bash
NEXT_HASH_SALT=my-deployment-salt next build
```

Version History

next.config.js: poweredByHeader

`REFERENCE`

Source: https://nextjs.org/docs/app/api-reference/config/next-config-js/poweredByHeader

Explains how to disable the `x-powered-by` header in Next.js by setting `poweredByHeader: false` in `next.config.js`.

poweredByHeader

By default Next.js will add the `x-powered-by` header. To opt-out of it, open `next.config.js` and disable the `poweredByHeader` config:

```next.config.js
module.exports = {
  poweredByHeader: false,
}
```

next.config.js: reactStrictMode | Next.js

`REFERENCE`

Source: https://nextjs.org/docs/app/api-reference/config/next-config-js/reactStrictMode

This page documents the reactStrictMode configuration option in next.config.js, which enables React Strict Mode in Next.js applications, with notes on default behavior and migration options.

reactStrictMode

Good to know: Since Next.js 13.5.1, Strict Mode is true by default with app router, so the above configuration is only necessary for pages. You can still disable Strict Mode by setting...

```next.config.js
module.exports = {
  reactStrictMode: true,
}
```

Parameters

Name	Type	Description	Default	Required
`reactStrictMode`	boolean	Enables React Strict Mode in the Next.js application. Defaults to true since Next.js 13.5.1 for the app router, but can be set to false to disable.	true (for app router since 13.5.1)	No

next.config.js: redirects | Next.js

REFERENCE

Source: https://nextjs.org/docs/app/api-reference/config/next-config-js/redirects

Documentation for the `redirects` key in `next.config.js`, which allows redirecting incoming request paths to different destination paths, including path matching, header/cookie/query matching,...

redirects

Redirects allow you to redirect an incoming request path to a different destination path.

To use redirects you can use the `redirects` key in `next.config.js`:

`redirects` can be defined as a...

next.config.js

```
module.exports = {
  redirects() {
    return [
      {
        source: '/about',
        destination: '/',
        permanent: true,
      },
    ]
  },
}
```

js

```
{
  source: '/old-blog/:path*',
  destination: '/blog/:path*',
  permanent: false
}
```

Path Matching

Path matches are allowed, for example `/old-blog/:slug` will match `/old-blog/first-post` (no nested paths):

The pattern `/old-blog/:slug` matches `/old-blog/first-post` and `/old-blog/post-1` but...

next.config.js

```
module.exports = {
  redirects() {
    return [
      {
        source: '/old-blog/:slug',
        destination: '/news/:slug', // Matched parameters can be used in the destination
        permanent:…
```

Wildcard Path Matching

To match a wildcard path you can use `*` after a parameter, for example `/blog/:slug*` will match `/blog/a/b/c/d/hello-world`:

next.config.js

```
module.exports = {
  redirects() {
    return [
      {
        source: '/blog/:slug*',
        destination: '/news/:slug*', // Matched parameters can be used in the destination
        permanent:…
```

Regex Path Matching

To match a regex path you can wrap the regex in parentheses after a parameter, for example `/post/:slug(\d{1,})` will match `/post/123` but not `/post/abc`:

The following characters `(`, `)`, `{`,...

next.config.js

```
module.exports = {
  redirects() {
    return [
      {
        source: '/post/:slug(\\d{1,})',
        destination: '/news/:slug', // Matched parameters can be used in the destination…
```

next.config.js

```
module.exports = {
  redirects() {
    return [
      {
        // this will match `/english(default)/something` being requested
        source: '/english\\(default\\)/:slug',
        destination:…
```

Header, Cookie, and Query Matching

To only match a redirect when header, cookie, or query values also match the `has` field or don't match the `missing` field can be used. Both the `source` and all `has` items must match and all...

next.config.js

```
module.exports = {
  redirects() {
    return [
      // if the header `x-redirect-me` is present,
      // this redirect will be applied
      {
        source: '/:path((?!another-page$).*)',…
```

Redirects with basePath support

When leveraging `basePath` support with redirects each `source` and `destination` is automatically prefixed with the `basePath` unless you…

next.config.js

```
module.exports = {
  basePath: '/docs',

  redirects() {
    return [
      {
        source: '/with-basePath', // automatically becomes /docs/with-basePath
        destination: '/another', //…
```

Redirects with i18n support

When implementing redirects with internationalization in the App Router, you can include locales in `next.config.js` redirects, but only as hardcoded paths.

For dynamic or per-request locale…

next.config.js

```
module.exports = {
  redirects() {
    return [
      {
        // Manually handle locale prefixes for App Router
        source: '/en/old-path',
        destination: '/en/new-path',…
```

Other Redirects

- Inside API Routes and Route Handlers, you can redirect based on the incoming request. -…

Version History

Version	Changes
`v13.3.0`	`missing` added.
`v10.2.0`	`has` added.
`v9.5.0`	`redirects` added.

Parameters

Name	Type	Description	Default	Required
`source`	string	The incoming request path pattern.		Yes
`destination`	string	The path you want to route to.		Yes
`permanent`	boolean	If `true` will use the 308 status code which instructs clients/search engines to cache the redirect forever, if `false` will use the 307 status code which is temporary and is not cached.		No
`basePath`	boolean | undefined	If false the `basePath` won't be included when matching, can be used for external redirects only.		No
`locale`	boolean | undefined	Whether the locale should not be included when matching.		No
`has`	Array<{ type: 'header' | 'cookie' | 'host' | 'query', key: string, value?: string }>	An array of has objects with the `type`, `key` and `value` properties. Both the `source` and all `has` items must match for the redirect to be applied.		No
`missing`	Array<{ type: 'header' | 'cookie' | 'host' | 'query', key: string, value?: string }>	An array of missing objects with the `type`, `key` and `value` properties. All `missing` items must not match for the redirect to be applied.		No
`statusCode`	number	A custom status code for older HTTP Clients to properly redirect. Can be used instead of the `permanent` property, but not both. To ensure IE11 compatibility, a `Refresh` header is automatically...		No

next.config.js: rewrites | Next.js

`REFERENCE`

Source: https://nextjs.org/docs/app/api-reference/config/next-config-js/rewrites

Documentation for the rewrites configuration option in next.config.js, which allows mapping incoming request paths to different destination paths, acting as a URL proxy.

rewrites

Rewrites allow you to map an incoming request path to a different destination path. Rewrites act as a URL proxy and mask the destination path, making it appear the user hasn't changed their location...

next.config.js

```
module.exports = {
  rewrites() {
    return [
      {
        source: '/about',
        destination: '/',
      },
    ]
  },
}
```

Rewrite properties

The following properties are available on each rewrite object:

Rewrites array vs object

When the `rewrites` function returns an array, rewrites are applied after checking the filesystem (pages and `/public` files) and before dynamic routes. When the `rewrites` function returns an object...

next.config.js

```
module.exports = {
  rewrites() {
    return {
      beforeFiles: [
        // These rewrites are checked after headers/redirects
        // and before all files including _next/public files which…
```

Order of checks

The order Next.js routes are checked is: headers are checked/applied, redirects are checked/applied, proxy, `beforeFiles` rewrites: for each entry, if `source`, `has`, and `missing` matches the...

Rewrite parameters

When using parameters in a rewrite the parameters will be passed in the query by default when none of the parameters are used in the `destination`. If a parameter is used in the destination none of...

next.config.js

```
module.exports = {
  rewrites() {
    return [
      {
        source: '/old-about/:path*',
        destination: '/about', // The :path parameter isn't used here so will be automatically passed i
```

next.config.js

```
module.exports = {
  rewrites() {
    return [
      {
        source: '/docs/:path*',
        destination: '/:path*', // The :path parameter is used here so will not be automatically passed
```

next.config.js

```
module.exports = {
  rewrites() {
    return [
      {
        source: '/:first/:second',
        destination: '/:first?second=:second',
        // Since the :first parameter is used in the…
```

Path Matching

Path matches are allowed, for example `/blog/:slug` will match `/blog/first-post` (no nested paths). The pattern `/blog/:slug` matches `/blog/first-post` and `/blog/post-1` but not `/blog/a/b` (no...

next.config.js

```
module.exports = {
  rewrites() {
    return [
      {
        source: '/blog/:slug',
        destination: '/news/:slug', // Matched parameters can be used in the destination
      },
    ]
  },
}
```

Wildcard Path Matching

To match a wildcard path you can use `*` after a parameter, for example `/blog/:slug*` will match `/blog/a/b/c/d/hello-world`.

next.config.js

```
module.exports = {
  rewrites() {
    return [
      {
        source: '/blog/:slug*',
        destination: '/news/:slug*', // Matched parameters can be used in the destination
      },
    ]
  },
}
```

Regex Path Matching

To match a regex path you can wrap the regex in parenthesis after a parameter, for example `/blog/:slug(\d{1,})` will match `/blog/123` but not `/blog/abc`. The following characters `(`, `)`, `{`,...

next.config.js

```
module.exports = {
  rewrites() {
    return [
      {
        source: '/old-blog/:post(\d{1,})',
        destination: '/blog/:post', // Matched parameters can be used in the destination
      },…
```

next.config.js

```
module.exports = {
  rewrites() {
    return [
      {
        // this will match `/english(default)/something` being requested
        source: '/english\(default\)/:slug',
        destination:…
```

Header, Cookie, and Query Matching

To only match a rewrite when header, cookie, or query values also match the `has` field or don't match the `missing` field can be used. Both the `source` and all `has` items must match and all...

next.config.js

```
module.exports = {
  rewrites() {
    return [
      // if the header `x-rewrite-me` is present,
      // this rewrite will be applied
      {
        source: '/:path*',
        has: [
          {…
```

Rewriting to an external URL

Rewrites allow you to rewrite to an external URL. This is especially useful for incrementally adopting Next.js. The following is an example rewrite for redirecting the `/blog` route of your main app...

next.config.js

```
module.exports = {
  rewrites() {
    return [
      {
        source: '/blog',
        destination: 'https://example.com/blog',
      },
      {
        source: '/blog/:slug',
        destination:…
```

next.config.js

```
module.exports = {
  trailingSlash: true,
  rewrites() {
    return [
      {
        source: '/blog/',
        destination: 'https://example.com/blog/',
      },
      {
        source:…
```

Incremental adoption of Next.js

You can also have Next.js fall back to proxying to an existing website after checking all Next.js routes. This way you don't have to change the rewrites configuration when migrating more pages to...

next.config.js

```
module.exports = {
  rewrites() {
    return {
      fallback: [
        {
          source: '/:path*',
          destination: `https://custom-routes-proxying-endpoint.vercel.app/:path*`,
        },…
```

Rewrites with basePath support

When leveraging `basePath` support with rewrites each `source` and `destination` is automatically prefixed with the `basePath` unless you add `basePath: false` to the rewrite.

next.config.js

```
module.exports = {
  basePath: '/docs',

  rewrites() {
    return [
      {
        source: '/with-basePath', // automatically becomes /docs/with-basePath
        destination: '/another', //…
```

Version History

Version	Changes
`v13.3.0`	`missing` added.
`v10.2.0`	`has` added.
`v9.5.0`	Headers added.

Parameters

Name	Type	Description	Default	Required
`source`	String	The incoming request path pattern.		Yes
`destination`	String	The path you want to route to.		Yes
`basePath`	false | undefined	If false the basePath won't be included when matching, can be used for external rewrites only.		No
`locale`	false | undefined	Whether the locale should not be included when matching.		No
`has`	Array	An array of has objects with the type, key and value properties.		No
`missing`	Array	An array of missing objects with the type, key and value properties.		No

serverExternalPackages

REFERENCE

Source: https://nextjs.org/docs/app/api-reference/config/next-config-js/serverExternalPackages

Describes the `serverExternalPackages` configuration option in Next.js, which allows opting out specific dependencies from Server Components bundling to use native Node.js require.

serverExternalPackages

Dependencies used inside Server Components and Route Handlers will automatically be bundled...

```javascript
/** @type {import('next').NextConfig} */
const nextConfig = {
  serverExternalPackages: ['@acme/ui'],
}

module.exports = nextConfig
```

next.config.js: taint | Next.js

REFERENCE

Source: https://nextjs.org/docs/app/api-reference/config/next-config-js/taint

Explains the experimental `taint` configuration option in Next.js, which enables React APIs for tainting objects and values to prevent sensitive data from being sent to the client. Includes usage,...

Usage

The `taint` option enables support for experimental React APIs for tainting objects and values. This feature helps prevent sensitive data from being accidentally passed to the client. When enabled,...

next.config.ts

```
import type { NextConfig } from 'next'

const nextConfig: NextConfig = {
  experimental: {
    taint: true,
  },
}

export default nextConfig
```

Caveats

Tainting can only keep track of objects by reference. Copying an object creates an untainted version, which loses all guarantees given by the API. You'll need to taint the copy.

Tainting cannot keep…

Examples

Tainting an object reference

In this case, the `getUserDetails` function returns data about a given user. We taint the user object reference, so that it cannot cross a Server-Client boundary. For example, assuming `UserCard` is…

typescript

```
import { experimental_taintObjectReference } from 'react'

function getUserDetails(id: string): UserDetails {
  const user = await db.queryUserById(id)

  experimental_taintObjectReference(
    'Do…
```

typescript

```
export async function ContactPage({
  params,
}: {
  params: Promise<{ id: string }>
}) {
  const { id } = await params
  const userDetails = await getUserDetails(id)

  return (
    <UserCard…
```

typescript

```
export async function ContactPage({
  params,
}: {
  params: Promise<{ id: string }>
}) {
  const userDetails = await getUserDetails(id)

  // Throws an error
  return <UserCard user={userDetails}…
```

Tainting a unique value

In this case, we can access the server configuration by awaiting calls to `config.getConfigDetails`. However, the system configuration contains the `SERVICE_API_KEY` that we don't want to expose to...

typescript

```
import { experimental_taintUniqueValue } from 'react'

function getSystemConfig(): SystemConfig {
  const config = await config.getConfigDetails()

  experimental_taintUniqueValue(
    'Do not pass…
```

typescript

```
export async function Dashboard() {
  const systemConfig = await getSystemConfig()

  return <ClientDashboard version={systemConfig.SERVICE_API_VERSION} />
}
```

typescript

```
export async function Dashboard() {
  const systemConfig = await getSystemConfig()
  // Someone makes a mistake in a PR
  const version = systemConfig.SERVICE_API_KEY

  return <ClientDashboard…
```

typescript

```
export async function Dashboard() {
  const systemConfig = await getSystemConfig()
  // Someone makes a mistake in a PR
  const version = `version::${systemConfig.SERVICE_API_KEY}`

  return…
```

trailingSlash

REFERENCE

Source: https://nextjs.org/docs/app/api-reference/config/next-config-js/trailingSlash

Describes the `trailingSlash` configuration option in Next.js, which controls whether URLs with trailing slashes are redirected to non-trailing-slash versions or vice versa, with exceptions for...

trailingSlash

By default Next.js will redirect URLs with trailing slashes to their counterpart without a trailing slash. For example `/about/` will redirect to `/about`. You can configure this behavior to act the...

next.config.js

```
module.exports = {
  trailingSlash: true,
}
```

Version History

Version	Changes
`v9.5.0`	`trailingSlash` added.

Parameters

Name	Type	Description	Default	Required
`trailingSlash`	boolean	When true, URLs without trailing slashes are redirected to their counterparts with trailing slashes. When false (default), URLs with trailing slashes are redirected to their counterparts without...	False	No

turbopackLocalPostcssConfig

`REFERENCE`

Source: https://nextjs.org/docs/app/api-reference/config/next-config-js/turbopackLocalPostcssConfig

This page documents the `turbopackLocalPostcssConfig` option in Next.js configuration, which changes how Turbopack resolves `postcss.config.js` files, allowing per-directory configs to take...

Overview

The `turbopackLocalPostcssConfig` option changes how Turbopack resolves `postcss.config.js` files. When enabled, Turbopack searches for the config starting from the CSS file's own directory first,...

Usage

To enable the option, set `experimental.turbopackLocalPostcssConfig` to `true` in your `next.config.ts` file.

next.config.ts

```
import type { NextConfig } from 'next'

const nextConfig: NextConfig = {
  experimental: {
    turbopackLocalPostcssConfig: true,
  },
}

export default nextConfig
```

Behavior

The table below shows the config resolution order for each setting:

- `false` (default): Project root → CSS file's directory
- `true`: CSS file's directory → project root

With the default behavior,...

Example

This is useful for projects that need different PostCSS transforms in different directories, such as a monorepo with multiple apps or design system packages. The following directory structure...

```text
my-app/
├── postcss.config.js          ← fallback (applied if no local config is found)
├── app/
│   └── page.module.css        ← uses root config
└── packages/
    └── ui/
        ├──…
```

Version History

The following table lists the version history for this feature:

- `v16.3.0`: `turbopackLocalPostcssConfig` introduced.

next.config.js: urlImports | Next.js

REFERENCE

Source: https://nextjs.org/docs/app/api-reference/config/next-config-js/urlImports

Documentation for the experimental urlImports feature in Next.js, which allows importing modules directly from external URLs, with configuration, security model, lockfile behavior, and examples.

urlImports

URL imports are an experimental feature that allows you to import modules directly from external servers (instead of from the local disk).

> ***Warning** : Only use domains that you trust to download...*

```next.config.js
module.exports = {
  experimental: {
    urlImports: ['https://example.com/assets/', 'https://cdn.skypack.dev'],
  },
}
```

```javascript
import { a, b, c } from 'https://example.com/assets/some/module.js'
```

Security Model

This feature is being designed with **security as the top priority** . To start, we added an experimental flag forcing you to explicitly allow the domains you accept URL imports from. We're working...

Lockfile

When using URL imports, Next.js will create a `next.lock` directory containing a lockfile and fetched assets. This directory **must be committed to Git** , not ignored by `.gitignore` .

- When...

Examples

Skypack

Static Image Imports

URLs in CSS

Asset Imports

```javascript
import confetti from 'https://cdn.skypack.dev/canvas-confetti'
import { useEffect } from 'react'

export default () => {
  useEffect(() => {
    confetti()
  })
  return <p>Hello</p>
}
```

```javascript
import Image from 'next/image'
import logo from 'https://example.com/assets/logo.png'

export default () => (
  <div>
    <Image src={logo} placeholder="blur" />
  </div>
)
```

```css
.className {
  background: url('https://example.com/assets/hero.jpg');
}
```

```javascript
```

```
const logo = new URL('https://example.com/assets/file.txt', import.meta.url)

console.log(logo.pathname)

// prints "/_next/static/media/file.a9727b5d.txt"
```

next.config.js: useOffline | Next.js

REFERENCE

Source: https://nextjs.org/docs/app/api-reference/config/next-config-js/useOffline

This page documents the experimental `useOffline` configuration option in Next.js, which enables offline connectivity detection, automatic retry of failed navigation, prefetch, and Server Action...

useOffline

This feature is currently experimental and subject to change, it's not recommended for production. Try it out and share your feedback on GitHub.

The `useOffline` configuration option enables offline...

next.config.ts

```
import type { NextConfig } from 'next'

const nextConfig: NextConfig = {
  experimental: {
    useOffline: true,
  },
}

export default nextConfig
```

How retry works

The offline state is entered through one of two paths:

- **Browser event.** Next.js registers a `window.addEventListener('offline', ...)` listener. When the OS reports the network interface as down,...

The connectivity check

Each check issues a single `HEAD` request to the current page's URL with the RSC header set, the same endpoint navigations use. The request is aborted after 200 ms.

Two outcomes count as...

Backoff

Delays between checks are stepped, not exponential, and capped at 3 seconds:

Attempt	Delay before next check
1	500 ms
2	1 s
3	2 s
4 and after	3 s

The…

Giving up

The polling loop never gives up on its own. It continues at the 3-second cap until a check succeeds or the page unloads. A device that goes offline for hours and then regains connectivity will have…

Retry of framework requests

While the offline state is active, any navigation, prefetch, or Server Action waits for the next connectivity check to succeed, whether it was newly issued or already in flight when the connection…

Traffic at reconnection

A single client does not produce a runaway burst of traffic against its origin:

- While the client is offline, a failed `fetch()` rejects locally at the browser's network layer. The request never…

Version History

Version	Changes
`v16.x.0`	`experimental.useOffline` configuration option introduced.

webVitalsAttribution

`REFERENCE`

Source: https://nextjs.org/docs/app/api-reference/config/next-config-js/webVitalsAttribution

Explains the experimental webVitalsAttribution option in next.config.js, which enables per-metric Web Vitals attribution to help pinpoint the source of Web Vitals issues.

webVitalsAttribution

This feature is currently experimental and subject to change, it's not recommended for production. Try it out and share your feedback on GitHub.

When…

next.config.js

```
module.exports = {
  experimental: {
    webVitalsAttribution: ['CLS', 'LCP'],
  },
}
```

Configuration: TypeScript | Next.js

REFERENCE

Source: https://nextjs.org/docs/app/api-reference/config/typescript

Extraction fallback content.

Configuration: TypeScript | Next.js

This page is also available as Markdown: request this page's URL with an `Accept: text/markdown` header. For an index of Next.js documentation , see /docs/llms.txt.Copy page

TypeScript

Last updated August 3, 2026

Next.js comes with built-in TypeScript, automatically installing the necessary packages and configuring the proper settings when you create a new project with `create-next-app`.

To add TypeScript to an existing project, rename a file to `.ts` / `.tsx`. Run `next dev` and `next build` to automatically install the necessary dependencies and add a `tsconfig.json` file with the recommended config options.

> ***Good to know*** *: If you already have a `jsconfig.json` file, copy the `paths` compiler option from the old `jsconfig.json` into the new `tsconfig.json` file, and delete the old `jsconfig.json` file.*

Using TypeScript 7

TypeScript 7 does not currently provide the JavaScript compiler API. To use TypeScript 7 during `next build`, install it in your project:

pnpmnpmyarnbunTerminal

Next.js uses the project-local `tsc` CLI by default, so no additional configuration is required. To use the JavaScript compiler API instead, set `experimental.useTypeScriptCli` to `false`.

> ***Good to know*** *:*
>
> - *CLI type checking prints the native `tsc` diagnostics. It does not apply Next.js-specific code frames or rewrite errors for routes, pages, layouts, or route handlers.*

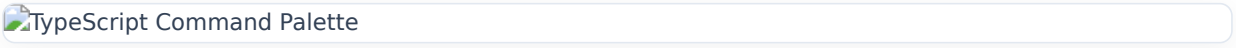
> - *The CLI checks the complete project selected by your `tsconfig` file. This includes test files and `.next/dev/types` when they are included by that configuration. `next build --debug-build-paths` does not narrow the files that are type checked and produces a warning when used with this option.*
> - *`typescript.tsconfigPath` continues to select the configuration passed to `tsc`. `typescript.ignoreBuildErrors` skips the type-checking step, including the CLI checker.*
> - *`experimental.useTypeScriptCli` is experimental and its behavior may change.*

IDE Plugin

Next.js includes a custom TypeScript plugin and type checker, which VSCode and other code editors can use for advanced type-checking and auto-completion.

You can enable the plugin in VS Code by:

- Opening the command palette (`Ctrl/⌘` + `Shift` + `P`)
- Searching for "TypeScript: Select TypeScript Version"
- Selecting "Use Workspace Version"



Now, when editing files, the custom plugin will be enabled. By default, the project-local `tsc` CLI is used when running `next build`. Set `experimental.useTypeScriptCli` to `false` to use the custom type checker instead.

The TypeScript plugin can help with:

- Warning if invalid values for segment config options are passed.
- Showing available options and in-context documentation.
- Ensuring the `'use client'` directive is used correctly.
- Ensuring client hooks (like `useState`) are only used in Client Components.

> 🎥 ***Watch:*** *Learn about the built-in TypeScript plugin → YouTube (3 minutes)*

End-to-End Type Safety

The Next.js App Router has **enhanced type safety** . This includes:

- **No serialization of data between fetching function and page** : You can `fetch` directly in components, layouts, and pages on the server. This data *does not* need to be serialized (converted to a string) to be passed to the client side for consumption in React. Instead, since `app` uses Server Components by default, we can use values like `Date`, `Map`, `Set`, and more without any extra steps. Previously, you needed to manually type the boundary between server and client with Next.js-specific types.

- **Streamlined data flow between components** : With the removal of `_app` in favor of root layouts, it is now easier to visualize the data flow between components and pages. Previously, data flowing between individual `pages` and `_app` were difficult to type and could introduce confusing bugs. With colocated data fetching in the App Router, this is no longer an issue.

Data Fetching in Next.js now provides as close to end-to-end type safety as possible without being prescriptive about your database or content provider selection.

We're able to type the response data as you would expect with normal TypeScript. For example:

app/page.tsxTypeScriptJavaScriptTypeScript

For *complete* end-to-end type safety, this also requires your database or content provider to support TypeScript. This could be through using an ORM or type-safe query builder.

Route-Aware Type Helpers

Next.js generates global helpers for App Router route types. These are available without imports and are generated during `next dev`, `next build`, or via `next typegen`:

- `PageProps`
- `LayoutProps`
- `RouteContext`

`next-env.d.ts`

Next.js generates a `next-env.d.ts` file in your project root. This file references Next.js type definitions, allowing TypeScript to recognize non-code imports (images, stylesheets, etc.) and Next.js-specific types.

Running `next dev`, `next build`, or `next typegen` regenerates this file.

> ***Good to know*** *:*
>
> - *`next-env.d.ts` is managed by Next.js. Its contents are an implementation detail and may change over time. Add it to `.gitignore`. If your project already tracks the file, remove it from Git. Do not edit this file manually.*
> - *The file must be in your `tsconfig.json` `include` array (`create-next-app` does this automatically).*

Examples

Type Checking Next.js Configuration Files

You can use TypeScript and import types in your Next.js configuration by using `next.config.ts`.

next.config.ts

Module resolution in `next.config.ts` is currently limited to CommonJS. However, ECMAScript Modules (ESM) syntax is available when using Node.js native TypeScript resolver for Node.js v22.10.0 and higher.

When using the `next.config.js` file, you can add some type checking in your IDE using JSDoc as below:

next.config.js

Using Node.js Native TypeScript Resolver for `next.config.ts`

> ***Note** : Available on Node.js v22.10.0+ and only when the feature is enabled. Next.js does not enable it.*

Next.js detects the Node.js native TypeScript resolver via `process.features.typescript`, added in **v22.10.0** . When present, `next.config.ts` can use native ESM, including top-level `await` and dynamic `import()`. This mechanism inherits the capabilities and limitations of Node's resolver.

In Node.js versions **v22.18.0+** , `process.features.typescript` is enabled by default. For versions between **v22.10.0** and **22.17.x** , opt in with `NODE_OPTIONS=--experimental-transform-types`:

Terminal

For CommonJS Projects (Default)

Although `next.config.ts` supports native ESM syntax in CommonJS projects, Node.js will still assume `next.config.ts` is a CommonJS file by default, resulting in Node.js reparsing the file as ESM when module syntax is detected. Therefore, we recommend using the `next.config.mts` file for CommonJS projects to explicitly indicate it's an ESM module:

next.config.mts

For ESM Projects

When `"type"` is set to `"module"` in `package.json`, your project uses ESM. Learn more about this setting in the Node.js docs. In this case, you can write `next.config.ts` directly with ESM syntax.

> ***Good to know** : When using `"type": "module"` in your `package.json`, all `.js` and `.ts` files in your project are treated as ESM modules by default. You may need to rename files with CommonJS syntax to `.cjs` or `.cts` extensions if needed.*

Statically Typed Links

Next.js can statically type links to prevent typos and other errors when using `next/link`, improving type safety when navigating between pages.

Works in both the Pages and App Router for the `href` prop in `next/link`. In the App Router, it also types `next/navigation` methods like `push`, `replace`, and `prefetch`. It does not type `next/router` methods in Pages Router.

Literal `href` strings are validated, while non-literal `href`s may require a cast with `as Route`.

To opt-into this feature, `typedRoutes` needs to be enabled and the project needs to be using TypeScript.

next.config.ts

Next.js will generate a link definition in `.next/types` that contains information about all existing routes in your application, which TypeScript can then use to provide feedback in your editor about invalid links.

> ***Good to know** : If you set up your project without `create-next-app`, ensure the generated Next.js types are included by adding `.next/types/**/*.ts` to the `include` array in your `tsconfig.json`:*

tsconfig.json

Currently, support includes any string literal, including dynamic segments. For non-literal strings, you need to manually cast with `as Route`. The example below shows both `next/link` and `next/navigation` usage:

app/example-client.tsx

The same applies for redirecting routes defined by proxy:

proxy.ts app/some/page.tsx

To accept `href` in a custom component wrapping `next/link`, use a generic:

You can also type a simple data structure and iterate to render links:

components/nav-items.ts

Then, map over the items to render `Link`s:

components/nav.tsx

> ***How does it work?***
>
> *When running, `next typegen`, `next dev` or `next build`, Next.js generates a hidden `.d.ts` file inside `.next` that contains information about all existing routes in your application (all valid routes as the `href` type of `Link`). This `.d.ts` file is included in `tsconfig.json` and the TypeScript compiler will check that `.d.ts` and provide feedback in your editor about invalid links.*

Type IntelliSense for Environment Variables

During development, Next.js generates a `.d.ts` file in `.next/types` that contains information about the loaded environment variables for your editor's IntelliSense. If the same environment variable key is defined in multiple files, it is deduplicated according to the Environment Variable Load Order.

To opt-into this feature, `experimental.typedEnv` needs to be enabled and the project needs to be using TypeScript.

next.config.ts

> ***Good to know*** *: Types are generated based on the environment variables loaded at development runtime, which excludes variables from* `.env.production*` *files by default. To include production-specific variables, run* `next dev` *with* `NODE_ENV=production`*.*

With Async Server Components

To use an `async` Server Component with TypeScript, ensure you are using TypeScript `5.1.3` or higher and `@types/react` `18.2.8` or higher.

If you are using an older version of TypeScript, you may see a `'Promise<Element>' is not a valid JSX element` type error. Updating to the latest version of TypeScript and `@types/react` should resolve this issue.

Incremental type checking

Since `v10.2.1` Next.js supports incremental type checking when enabled in your `tsconfig.json`, this can help speed up type checking in larger applications.

Custom `tsconfig` path

In some cases, you might want to use a different TypeScript configuration for builds or tooling. To do that, set `typescript.tsconfigPath` in `next.config.ts` to point Next.js to another `tsconfig` file.

next.config.ts

For example, switch to a different config for production builds:

next.config.ts Why you might use a separate `tsconfig` for builds

You might need to relax checks in scenarios like monorepos, where the build also validates shared dependencies that don't match your project's standards, or when loosening checks in CI to continue delivering while migrating locally to stricter TypeScript settings (and still wanting your IDE to highlight misuse).

For example, if your project uses `useUnknownInCatchVariables` but some monorepo dependencies still assume `any`:

tsconfig.build.json

This keeps your editor strict via `tsconfig.json` while allowing the production build to use relaxed settings.

> ***Good to know** :*
>
> - *IDEs typically read `tsconfig.json` for diagnostics and IntelliSense, so you can still see IDE warnings while production builds use the alternate config. Mirror critical options if you want parity in the editor.*
> - *In development, only `tsconfig.json` is watched for changes. If you edit a different file name via `typescript.tsconfigPath`, restart the dev server to apply changes.*
> - *The configured file is used in `next dev`, `next build`, and `next typegen`.*

Disabling TypeScript errors in production

Next.js fails your **production build** (`next build`) when TypeScript errors are present in your project.

If you'd like Next.js to dangerously produce production code even when your application has errors, you can disable the built-in type checking step.

If disabled, be sure you are running type checks as part of your build or deploy process, otherwise this can be very dangerous.

Open `next.config.ts` and enable the `ignoreBuildErrors` option in the `typescript` config:

next.config.ts

> ***Good to know** : You can run `tsc --noEmit` to check for TypeScript errors yourself before building. This is useful for CI/CD pipelines where you'd like to check for TypeScript errors before deploying.*

Custom type declarations

When you need to declare custom types, you might be tempted to modify `next-env.d.ts`. However, this file is automatically generated, so any changes you make will be overwritten. Instead, you should create a new file, let's call it `new-types.d.ts`, and reference it in your `tsconfig.json`:

tsconfig.json

Version Changes

Version	Changes
`v15.0.0`	`next.config.ts` support added for TypeScript projects.
`v13.2.0`	Statically typed links are available in beta.
`v12.0.0`	SWC is now used by default to compile TypeScript and TSX for faster builds.
`v10.2.1`	Incremental type checking support added when enabled in your `tsconfig.json`.

Was this helpful?

supported.Send

carbon

```
pnpm add -D typescript@^7
```

gdscript

```
async function getData() {
  const res = await fetch('https://api.example.com/...')
  // The return value is *not* serialized
  // You can return Date, Map, Set, etc.
  return res.json()
}

export default async function Page() {
  const name = await getData()

  return '...'
}
```

python

```
import type { NextConfig } from 'next'

const nextConfig: NextConfig = {
  /* config options here */
}

export default nextConfig
```

gdscript

```
// @ts-check

/** @type {import('next').NextConfig} */
const nextConfig = {
  /* config options here */
}

module.exports = nextConfig
```

carbon

```
NODE_OPTIONS=--experimental-transform-types next <command>
```

python

```
import type { NextConfig } from 'next'

// Top-level await and dynamic import are supported
const flags = await import('./flags.js').then((m) => m.default ?? m)

const nextConfig: NextConfig = {
  /* config options here */
  typedRoutes: Boolean(flags?.typedRoutes),
}

export default nextConfig
```

python

```
import type { NextConfig } from 'next'

const nextConfig: NextConfig = {
  typedRoutes: true,
}

export default nextConfig
```

carbon

```
{
  "include": [
    "next-env.d.ts",
    ".next/types/**/*.ts",
    "**/*.ts",
    "**/*.tsx"
  ],
  "exclude": ["node_modules"]
}
```

python

```
'use client'

import type { Route } from 'next'
import Link from 'next/link'
import { useRouter } from 'next/navigation'

export default function Example() {
  const router = useRouter()
  const slug = 'nextjs'

  return (
    <>
      {/* Link: literal and dynamic */}
      <Link href="/about" />
      <Link href={`/blog/${slug}`} />
      <Link href={('/blog/' + slug) as Route} />
      {/* TypeScript error if href is not a valid route */}
      <Link href="/aboot" />

      {/* Router: literal and dynamic strings are validated */}
      <button onClick={() => router.push('/about')}>Push About</button>
      <button onClick={() => router.replace(`/blog/${slug}`)}>
        Replace Blog
      </button>
      <button onClick={() => router.prefetch('/contact')}>
        Prefetch Contact
      </button>

      {/* For non-literal strings, cast to Route */}
      <button onClick={() => router.push(('/blog/' + slug) as Route)}>
        Push Non-literal Blog
      </button>
    </>
  )
}
```

python

```
import { NextRequest, NextResponse } from 'next/server'

export function proxy(request: NextRequest) {
  if (request.nextUrl.pathname === '/proxy-redirect') {
    return NextResponse.redirect(new URL('/', request.url))
  }

  return NextResponse.next()
}
```

python

```
import type { Route } from 'next'

export default function Page() {
  return <Link href={'/proxy-redirect' as Route}>Link Text</Link>
}
```

python

```
import type { Route } from 'next'
import Link from 'next/link'

function Card<T extends string>({ href }: { href: Route<T> | URL }) {
  return (
    <Link href={href}>
      <div>My Card</div>
    </Link>
  )
}
```

python

```
import type { Route } from 'next'

type NavItem<T extends string = string> = {
  href: T
  label: string
}

export const navItems: NavItem<Route>[] = [
  { href: '/', label: 'Home' },
  { href: '/about', label: 'About' },
  { href: '/blog', label: 'Blog' },
]
```

python

```
import Link from 'next/link'
import { navItems } from './nav-items'

export function Nav() {
  return (
    <nav>
      {navItems.map((item) => (
        <Link key={item.href} href={item.href}>
          {item.label}
        </Link>
      ))}
    </nav>
  )
}
```

python

```
import type { NextConfig } from 'next'

const nextConfig: NextConfig = {
  experimental: {
    typedEnv: true,
  },
}

export default nextConfig
```

python

```
import type { NextConfig } from 'next'

const nextConfig: NextConfig = {
  typescript: {
    tsconfigPath: 'tsconfig.build.json',
  },
}

export default nextConfig
```

python

```
import type { NextConfig } from 'next'

const isProd = process.env.NODE_ENV === 'production'

const nextConfig: NextConfig = {
  typescript: {
    tsconfigPath: isProd ? 'tsconfig.build.json' : 'tsconfig.json',
  },
}

export default nextConfig
```

text

```
{
  "extends": "./tsconfig.json",
  "compilerOptions": {
    "useUnknownInCatchVariables": false
  }
}
```

python

```
import type { NextConfig } from 'next'

const nextConfig: NextConfig = {
  typescript: {
    // !! WARN !!
    // Dangerously allow production builds to successfully complete even if
    // your project has type errors.
    // !! WARN !!
    ignoreBuildErrors: true,
  },
}

export default nextConfig
```

carbon

```
{
  "compilerOptions": {
    "skipLibCheck": true
    //...truncated...
  },
  "include": [
    "new-types.d.ts",
    "next-env.d.ts",
    ".next/types/**/*.ts",
    "**/*.ts",
    "**/*.tsx"
  ],
  "exclude": ["node_modules"]
}
```

Directives

REFERENCE

Source: https://nextjs.org/docs/app/api-reference/directives

This page provides an index of Next.js directives, including use cache, use client, and use server, with links to their detailed documentation.

Directives

The following directives are available:

- use cache: Learn how to use the "use cache" directive to cache data in your Next.js application.
- [use...

Directives: use cache | Next.js

REFERENCE

Source: https://nextjs.org/docs/app/api-reference/directives/use-cache

Extraction fallback content.

Directives: use cache | Next.js

This page is also available as Markdown: request this page's URL with an `Accept: text/markdown` header. For an index of Next.js documentation , see /docs/llms.txt.Copy page

use cache

Last updated July 22, 2026

The `use cache` directive allows you to mark a route, React component, or a function as cacheable. It can be used at the top of a file to indicate that all exports in the file should be cached, or inline at the top of a function or component to cache the return value. Functions and components that use `use cache` must be async.

> ***Good to know:***

> - *To use cookies or headers, read them outside cached scopes and pass values as arguments. This is the preferred pattern.*
> - *If the in-memory cache isn't sufficient for runtime data, `'use cache: remote'` allows platforms to provide a dedicated cache handler, though it requires a network roundtrip to check the cache and typically incurs platform fees.*
> - *For compliance requirements or when you can't refactor to pass runtime data as arguments to a `use cache` scope, see `'use cache: private'`.*

Usage

`use cache` is a Cache Components feature. To enable it, add the `cacheComponents` option to your `next.config.ts` file:

next.config.tsTypeScriptJavaScriptTypeScript

Then, add `use cache` at the file, component, or function level. All functions and components using `use cache` must be async. When used at file level, every exported function becomes a cached function and must also be async:

How `use cache` works

Cache keys

A cache entry's key is generated using a serialized version of its inputs, which includes:

- **Build ID** - Unique per build, changing this invalidates all cache entries. If `deploymentId` is configured, it overrides the build ID for cache key purposes.
- **Function ID** - A secure hash of the function's location and signature in the codebase
- **Serializable arguments** - Props (for components) or function arguments
- **HMR refresh hash** (development only) - Invalidates cache on hot module replacement

When a cached function references variables from outer scopes, those variables are automatically captured and bound as arguments, making them part of the cache key.

lib/data.ts

In the snippet above, `userId` is captured from the outer scope and `filter` is passed as an argument, so both become part of the `getData` function's cache key. This means different user and filter combinations will have separate cache entries.

> ***Good to know:** When a cached function reads root parameters, only the ones it actually reads become part of its cache key.*

Serialization

Arguments to cached functions and their return values must be serializable.

For a complete reference, see:

- Serializable arguments - Uses **React Server Components** serialization
- Serializable return types - Uses **React Client Components** serialization

> ***Good to know:*** *Arguments and return values use different serialization systems. Server Component serialization (for arguments) is more restrictive than Client Component serialization (for return values). This means you can return JSX elements but cannot accept them as arguments unless using pass-through patterns.*

Supported types

Arguments:

- Primitives: `string`, `number`, `boolean`, `null`, `undefined`
- Plain objects: `{ key: value }`
- Arrays: `[1, 2, 3]`
- Dates, Maps, Sets, TypedArrays, ArrayBuffers
- React elements (as pass-through only)

Return values:

- Same as arguments, plus JSX elements

Unsupported types

- Class instances
- Functions (except as pass-through)
- Symbols, WeakMaps, WeakSets
- URL instances

app/components/user-card.tsx

Pass-through (non-serializable arguments)

You can accept non-serializable values **as long as you don't introspect them** . This enables composition patterns with `children` and Server Actions:

app/components/cached-wrapper.tsx

You can also pass Server Actions through cached components:

app/components/cached-form.tsx

Constraints

Cached functions execute in an isolated environment. The following constraints ensure cache behavior remains predictable and secure.

Request-time APIs

Cached functions and components **cannot** access runtime APIs like `cookies()`, `headers()`, or `searchParams`, and the restriction follows the call stack: a helper the cached function calls that reads one of these fails the same way, with the `next-request-in-use-cache` error. On a dynamically rendered route this surfaces when the route runs, so it can pass `next build` and fail under `next start`. Read these values outside the cached scope and pass them as arguments.

Runtime caching considerations

While `use cache` is designed primarily to include uncached data in the static shell, it can also cache data at runtime using in-memory LRU (Least Recently Used) storage.

With the default in-memory handler, runtime cache behavior depends on your hosting environment:

Environment	Runtime Caching Behavior
Serverless	Cache entries typically don't persist across requests (each request can be a different instance), or during revalidation. Build-time caching works normally.
Self-hosted	Cache entries persist across requests. Control cache size with `cacheMaxMemorySize`.

For example, in a serverless environment, a cached function shared by two pages executes on each static shell revalidation, whereas in self-hosted or environments with persistent memory, the cached output is reused if it's still fresh.

If the default in-memory cache isn't enough, consider `use cache: remote` which allows platforms to provide a dedicated cache handler (like Redis or KV database). This helps reduce hits against data sources not scaled to your total traffic, though it comes with costs (storage, network latency, platform fees).

With the default in-memory handler, serverless instances are ephemeral, so entries may not be reused between requests, unlike with `use cache: remote`. Neither caching directive carries over to a new deploy, because the cache key includes the build (or `deploymentId`) ID.

For data that needs to persist across deploys, use `unstable_cache` for non-`fetch` functions or the `fetch` cache.

Very rarely, for compliance requirements or when you can't refactor your code to pass runtime data as arguments to a `use cache` scope, you might need `use cache: private`.

Draft Mode

When Draft Mode is enabled, all cached functions and components re-execute on every request, and results are not saved to the cache. This ensures draft content is always fresh without requiring any changes to your caching code.

You can read `isEnabled` from `draftMode()` inside a `use cache` scope, however, other runtime APIs like `cookies()` and `headers()` are not allowed, even when Draft Mode is active. See Passing runtime values to cached functions for the recommended pattern.

app/components/content.tsx

Calling `enable()` or `disable()` inside a caching directive scope will also throw an error. Draft Mode can only be toggled in Route Handlers or Server Actions.

React.cache isolation

`React.cache` operates in an isolated scope inside `use cache` boundaries. Values stored via `React.cache` outside a `use cache` function are not visible inside it.

This means you cannot use `React.cache` to pass data into a `use cache` scope:

This isolation ensures cached functions have predictable, self-contained behavior. To pass data into a `use cache` scope, use function arguments instead.

`use cache` at runtime

On the **server** , cache entries are stored in-memory and respect the `revalidate` and `expire` times from your `cacheLife` configuration. You can customize the cache storage by configuring `cacheHandlers` in your `next.config.js` file.

On the **client** , content from the server cache is stored in the browser's memory for the duration defined by the `stale` time. The client router enforces a **minimum 30-second stale time** , regardless of configuration.

The `x-nextjs-stale-time` response header communicates cache lifetime from server to client, ensuring coordinated behavior.

Revalidation

Cached functions revalidate based on the `revalidate` and `expire` times in their `cacheLife` profile, or on-demand through tags. These two approaches are not mutually exclusive and are often paired:

- Time-based : refresh automatically after a set duration with `cacheLife` .
- On-demand : invalidate after a mutation with `cacheTag` and `revalidateTag` or `updateTag` .

For example, a blog post that changes only when its author edits it can use a long `cacheLife` like `max` with a `cacheTag` , then invalidate on demand when the post is saved. A list of recent posts that updates throughout the day can use a shorter profile like `hours` to refresh on its own, without manual invalidation.

Time-based revalidation

Set an explicit cache lifetime with `cacheLife` in every `use cache` scope. It makes the cache behavior clear at the call site, instead of depending on the `default` profile or surrounding caches.

lib/data.ts

If you omit `cacheLife` , the `default` profile applies and the lifetime is no longer explicit at the call site:

- **stale** : 5 minutes (client-side)
- **revalidate** : 15 minutes (server-side)
- **expire** : never expires by time

lib/data.ts

Nesting a short-lived use cache inside one without an explicit cacheLife fails the build during prerendering. See Nested short-lived caches for the rule and fix.

On-demand revalidation

Use `cacheTag` , `updateTag` , or `revalidateTag` for on-demand cache invalidation:

lib/data.ts app/actions.ts

Both `cacheLife` and `cacheTag` integrate across client and server caching layers, meaning you configure your caching semantics in one place and they apply everywhere.

Examples

Caching an entire route with `use cache`

To prerender an entire route, add `use cache` to the top of **both** the `layout` and `page` files. Each of these segments are treated as separate entry points in your application, and will be cached independently.

app/layout.tsxTypeScriptJavaScriptTypeScript

Any components imported and nested in `page` file are part of the cache output associated with the `page`.

app/page.tsxTypeScriptJavaScriptTypeScript

> ***Good to know** :*
>
> - *If `use cache` is added only to the `layout` or the `page` , only that route segment and any components imported into it will be cached.*

Caching a component's output with `use cache`

You can use `use cache` at the component level to cache any fetches or computations performed within that component. The cache entry will be reused as long as the serialized props produce the same value in each instance.

app/components/bookings.tsxTypeScriptJavaScriptTypeScript

Caching function output with `use cache`

Since you can add `use cache` to any asynchronous function, you aren't limited to caching components or routes only. You might want to cache a network request, a database query, or a slow computation.

app/actions.tsTypeScriptJavaScriptTypeScript

> ***Good to know:*** *When a cached directive (`use cache`, `use cache: private`, or `use cache: remote`) is at the top of a file, you can import its exported functions into a Client Component and call them directly; they run on the server and return the result, similar to a Server Function. Prefer calling cached functions on the server and passing results down as props.*

Interleaving

In React, composition with `children` or slots is a well-known pattern for building flexible components. When using `use cache`, you can continue to compose your UI in this way. Anything included as `children`, or other compositional slots, in the returned JSX will be passed through the cached component without affecting its cache entry.

As long as you don't directly reference any of the JSX slots inside the body of the cacheable function itself, their presence in the returned output won't affect the cache entry.

app/page.tsxTypeScriptJavaScriptTypeScript

You can also pass Server Actions through cached components to Client Components without invoking them inside the cacheable function.

app/page.tsxTypeScriptJavaScriptTypeScript

app/ClientComponent.tsxTypeScriptJavaScriptTypeScript

Troubleshooting

Debugging cache behavior

Verbose logging

Set `NEXT_PRIVATE_DEBUG_CACHE=1` for verbose cache logging:

> ***Good to know:*** *This environment variable also logs ISR and other caching mechanisms. See Verifying correct production behavior for more details.*

Console log replays

In development, console logs from cached functions appear with a `Cache` prefix.

Build Hangs (Cache Timeout)

If your build hangs, you're accessing Promises that resolve to uncached or runtime data, created outside a `use cache` boundary. The cached function waits for data that can't resolve during the build, causing a timeout after 50 seconds.

When the build timeouts you'll see this error message:

> *Error: Filling a cache during prerender timed out, likely because request-specific arguments such as params, searchParams, cookies() or uncached data were used inside "use cache".*

Common ways this happens: passing such Promises as props, accessing them via closure, or retrieving them from shared storage (Maps).

> ***Good to know:*** *Directly calling* `cookies()` *or* `headers()` *inside* `use cache` *fails immediately with a different error, not a timeout.*

Passing runtime data Promises as props:

app/page.tsx

Await the `cookies` store in the `Dynamic` component, and pass a cookie value to the `Cached` component.

Shared deduplication storage:

app/page.tsx

Use Next.js's built-in `fetch()` deduplication or use separate Maps for cached and uncached contexts.

Platform Support

Deployment Option	Supported

| Node.js server | Yes | | Docker container | Yes | | Static export | No | | Adapters | Platform-specific |

Learn how to configure caching when self-hosting Next.js.

Version History

Version	Changes

| `v16.0.0` | `"use cache"` is enabled with the Cache Components feature. | | `v15.0.0` | `"use cache"` is introduced as an experimental feature. |

Related

View related API references.[### use cache: private

Learn how to use the "use cache: private" directive to cache functions that access runtime request APIs.](/docs/app/api-reference/directives/use-cache-private)[### cacheComponents

Learn how to enable the cacheComponents flag in Next.js.](/docs/app/api-reference/config/next-config-js/cacheComponents)[### cacheLife

Learn how to set up cacheLife configurations in Next.js.](/docs/app/api-reference/config/next-config-js/cacheLife)[### cacheHandlers

Configure custom cache handlers for use cache directives in Next.js.](/docs/app/api-reference/config/next-config-js/cacheHandlers)[### cacheTag

Learn how to use the cacheTag function to manage cache invalidation in your Next.js application.](/docs/app/api-reference/functions/cacheTag)[### cacheLife

Learn how to use the cacheLife function to set the cache expiration time for a cached function or component.](/docs/app/api-reference/functions/cacheLife)[### revalidateTag

API Reference for the revalidateTag function.](/docs/app/api-reference/functions/revalidateTag)

Was this helpful?

supported.Send

python

```python
import type { NextConfig } from 'next'

const nextConfig: NextConfig = {
  cacheComponents: true,
}

export default nextConfig
```

gdscript

```gdscript
// File level
'use cache'

export default async function Page() {
  // ...
}

// Component level
export async function MyComponent() {
  'use cache'
  return <></>
}

// Function level
export async function getData() {
  'use cache'
  const data = await fetch('/api/data')
  return data
}
```

gdscript

```gdscript
async function Component({ userId }: { userId: string }) {
  const getData = async (filter: string) => {
    'use cache'
    // Cache key includes both userId (from closure) and filter (argument)
    return fetch(`/api/users/${userId}/data?filter=${filter}`)
  }

  return getData('active')
}
```

xml

```
// Valid - primitives and plain objects
async function UserCard({
  id,
  config,
}: {
  id: string
  config: { theme: string }
}) {
  'use cache'
  return <div>{id}</div>
}

// Invalid - class instance
async function UserProfile({ user }: { user: UserClass }) {
  'use cache'
  // Error: Cannot serialize class instance
  return <div>{user.name}</div>
}
```

xml

```
async function CachedWrapper({ children }: { children: ReactNode }) {
  'use cache'
  // Don't read or modify children - just pass it through
  return (
    <div className="wrapper">
      <header>Cached Header</header>
      {children}
    </div>
  )
}

// Usage: children can be dynamic
export default function Page() {
  return (
    <CachedWrapper>
      <DynamicComponent /> {/* Not cached, passed through */}
    </CachedWrapper>
  )
}
```

xml

```
async function CachedForm({ action }: { action: () => Promise<void> }) {
  'use cache'
  // Don't call action here - just pass it through
  return <form action={action}>{/* ... */}</form>
}
```

python

```
import { draftMode } from 'next/headers'

async function Content() {
  'use cache'

  const { isEnabled } = await draftMode()
  const url = isEnabled
    ? 'https://draft.example.com/content'
    : 'https://production.example.com/content'

  const data = await fetch(url)
  return <article>{/* ... */}</article>
}
```

python

```
import { cache } from 'react'

const store = cache(() => ({ current: null as string | null }))

function Parent() {
  const shared = store()
  shared.current = 'value from parent'
  return <Child />
}

async function Child() {
  'use cache'
  const shared = store()
  // shared.current is null, not 'value from parent'
  // use cache has its own isolated React.cache scope
  return <div>{shared.current}</div>
}
```

python

```
import { cacheLife } from 'next/cache'

async function getData() {
  'use cache'
  cacheLife('hours') // Use built-in 'hours' profile
  return fetch('/api/data')
}
```

teratermmacro

```
async function getData() {
  'use cache'
  // Implicitly uses the 'default' profile
  return fetch('/api/data')
}
```

python

```
import { cacheTag } from 'next/cache'

async function getProducts() {
  'use cache'
  cacheTag('products')
  return fetch('/api/products')
}
```

python

```
'use server'

import { updateTag } from 'next/cache'

export async function updateProduct() {
  await db.products.update(...)
  updateTag('products') // Invalidates all 'products' caches
}
```

xml

```
'use cache'

export default async function Layout({ children }: { children: ReactNode }) {
  return <div>{children}</div>
}
```

xml

```
'use cache'

async function Users() {
  const users = await fetch('/api/users')
  // loop through users
}

export default async function Page() {
  return (
    <main>
      <Users />
    </main>
  )
}
```

gdscript

```
export async function Bookings({ type = 'haircut' }: BookingsProps) {
  'use cache'
  async function getBookingsData() {
    const data = await fetch(`/api/bookings?type=${encodeURIComponent(type)}`)
    return data
  }
  return //...
}

interface BookingsProps {
  type: string
}
```

gdscript

```
export async function getData() {
  'use cache'

  const data = await fetch('/api/data')
  return data
}
```

xml

```
export default async function Page() {
  const uncachedData = await getData()
  return (
    // Pass compositional slots as props, e.g. header and children
    <CacheComponent header={<h1>Home</h1>}>
      {/* DynamicComponent is provided as the children slot */}
      <DynamicComponent data={uncachedData} />
    </CacheComponent>
  )
}

async function CacheComponent({
  header, // header: a compositional slot, injected as a prop
  children, // children: another slot for nested composition
}: {
  header: ReactNode
  children: ReactNode
}) {
  'use cache'
  const cachedData = await fetch('/api/cached-data')
  return (
    <div>
      {header}
      <PrerenderedComponent data={cachedData} />
      {children}
    </div>
  )
}
```

python

```
import ClientComponent from './ClientComponent'

export default async function Page() {
  const performUpdate = async () => {
    'use server'
    // Perform some server-side update
    await db.update(...)
  }

  return <CachedComponent performUpdate={performUpdate} />
}

async function CachedComponent({
  performUpdate,
}: {
  performUpdate: () => Promise<void>
}) {
  'use cache'
  // Do not call performUpdate here
  return <ClientComponent action={performUpdate} />
}
```

xml

```
'use client'

export default function ClientComponent({
  action,
}: {
  action: () => Promise<void>
}) {
  return <button onClick={action}>Update</button>
}
```

scdoc

```
NEXT_PRIVATE_DEBUG_CACHE=1 npm run dev
# or for production
NEXT_PRIVATE_DEBUG_CACHE=1 npm run start
```

python

```
import { cookies } from 'next/headers'
import { Suspense } from 'react'

export default function Page() {
  return (
    <Suspense fallback={<div>Loading...</div>}>
      <Dynamic />
    </Suspense>
  )
}

async function Dynamic() {
  const cookieStore = cookies()
  return <Cached promise={cookieStore} /> // Build hangs
}

async function Cached({ promise }: { promise: Promise<unknown> }) {
  'use cache'
  const data = await promise // Waits for runtime data during build
  return <p>..</p>
}
```

python

```
// Problem: Map stores dynamic Promises, accessed by cached code
import { Suspense } from 'react'

const cache = new Map<string, Promise<string>>()

export default function Page() {
  return (
    <>
      <Suspense fallback={<div>Loading...</div>}>
        <Dynamic id="data" />
      </Suspense>
      <Cached id="data" />
    </>
  )
}

async function Dynamic({ id }: { id: string }) {
  // Stores dynamic Promise in shared Map
  cache.set(
    id,
    fetch(`https://api.example.com/${id}`).then((r) => r.text())
  )
  return <p>Dynamic</p>
}

async function Cached({ id }: { id: string }) {
  'use cache'
  return <p>{await cache.get(id)}</p> // Build hangs - retrieves dynamic Promise
}
```