Custom Server

▶ **Examples**

By default, Next.js includes its own server with `next start`. If you have an existing backend, you can still use it with Next.js (this is not a custom server). A custom Next.js server allows you to start a server 100% programmatically in order to use custom server patterns. Most of the time, you will not need this – but it's available for complete customization.

> ***Note:** A custom server **cannot** be deployed on Vercel.*

> *Before deciding to use a custom server, please keep in mind that it should only be used when the integrated router of Next.js can't meet your app requirements. A custom server will remove important performance optimizations, like **serverless functions** and **Automatic Static Optimization**.*

Take a look at the following example of a custom server:

```
// server.js
const { createServer } = require('http')
const { parse } = require('url')
const next = require('next')

const dev = process.env.NODE_ENV !== 'production'
const hostname = 'localhost'
const port = 3000
// when using middleware `hostname` and `port` must be provided below
const app = next({ dev, hostname, port })
const handle = app.getRequestHandler()

app.prepare().then(() => {
  createServer(async (req, res) => {
    try {
      // Be sure to pass `true` as the second argument to `url.parse`.
      // This tells it to parse the query portion of the URL.
      const parsedUrl = parse(req.url, true)
      const { pathname, query } = parsedUrl

      if (pathname === '/a') {
        await app.render(req, res, '/a', query)
      } else if (pathname === '/b') {
        await app.render(req, res, '/b', query)
      } else {
        await handle(req, res, parsedUrl)
      }
    } catch (err) {
      console.error('Error occurred handling', req.url, err)
      res.statusCode = 500
      res.end('internal server error')
    }
  }).listen(port, (err) => {
    if (err) throw err
    console.log(`> Ready on http://${hostname}:${port}`)
  })
})
```

> *`server.js` doesn't go through babel or webpack. Make sure the syntax and sources this file requires are compatible with the current node version you are running.*

To run the custom server you'll need to update the `scripts` in `package.json` like so:

```
"scripts": {
  "dev": "node server.js",
  "build": "next build",
  "start": "NODE_ENV=production node server.js"
}
```

The custom server uses the following import to connect the server with the Next.js application:

```
const next = require('next')
const app = next({})
```

The above `next` import is a function that receives an object with the following options:

- `dev` : `Boolean` - Whether or not to launch Next.js in dev mode. Defaults to `false`
- `dir` : `String` - Location of the Next.js project. Defaults to `'.'`
- `quiet` : `Boolean` - Hide error messages containing server information. Defaults to `false`
- `conf` : `object` - The same object you would use in next.config.js. Defaults to `{}`

The returned `app` can then be used to let Next.js handle requests as required.

Disabling file-system routing

By default, `Next` will serve each file in the `pages` folder under a pathname matching the filename. If your project uses a custom server, this behavior may result in the same content being served from multiple paths, which can present problems with SEO and UX.

To disable this behavior and prevent routing based on files in `pages`, open `next.config.js` and disable the `useFileSystemPublicRoutes` config:

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

> *Note that `useFileSystemPublicRoutes` disables filename routes from SSR; client-side routing may still access those paths. When using this option, you should guard against navigation to routes you do not want programmatically.*

> *You may also wish to configure the client-side router to disallow client-side redirects to filename routes; for that refer to `router.beforePopState`.*