Incrementally Adopting Next.js

▶ **Examples**

Next.js has been designed for gradual adoption. With Next.js, you can continue using your existing code and add as much (or as little) React as you need. By starting small and incrementally adding more pages, you can prevent derailing feature work by avoiding a complete rewrite.

Strategies

Subpath

The first strategy is to configure your server or proxy such that, everything under a specific subpath points to a Next.js app. For example, your existing website might be at `example.com`, and you might configure your proxy such that `example.com/store` serves a Next.js e-commerce store.

Using `basePath`, you can configure your Next.js application's assets and links to automatically work with your new subpath `/store`. Since each page in Next.js is its own standalone route, pages like `pages/products.js` will route to `example.com/store/products` in your application.

```
// next.config.js

module.exports = {
  basePath: '/store',
}
```

To learn more about `basePath`, take a look at our documentation.

> *This feature was introduced in Next.js 9.5 and up. If you're using older versions of Next.js, please upgrade before trying it out.*

Rewrites

The second strategy is to create a new Next.js app that points to the root URL of your domain. Then, you can use `rewrites` inside `next.config.js` to have some subpaths to be proxied to your existing app.

For example, let's say you created a Next.js app to be served from `example.com` with the following `next.config.js`. Now, requests for the pages you've added to this Next.js app (e.g. `/about` if you've added `pages/about.js`) will be handled by Next.js, and requests for any other route (e.g. `/dashboard`) will be proxied to `proxy.example.com`.

```
// next.config.js

module.exports = {
  async rewrites() {
    return {
      // After checking all Next.js pages (including dynamic routes)
      // and static files we proxy any other requests
      fallback: [
        {
          source: '/:path*',
          destination: `https://proxy.example.com/:path*`,
```

```
        },
      ],
    }

    // For versions of Next.js < v10.1 you can use a no-op rewrite instead
    return [
      // we need to define a no-op rewrite to trigger checking
      // all pages/static files before we attempt proxying
      {
        source: '/:path*',
        destination: '/:path*',
      },
      {
        source: '/:path*',
        destination: `https://proxy.example.com/:path*`,
      },
    ]
  },
}
```

To learn more about rewrites, take a look at our documentation.

> *This feature was introduced in Next.js 9.5 and up. If you're using older versions of Next.js, please upgrade before trying it out.*

Micro-Frontends with Monorepos and Subdomains

Next.js and Vercel make it straightforward to adopt micro-frontends and deploy as a Monorepo. This allows you to use subdomains to adopt new applications incrementally. Some benefits of micro-frontends:

- Smaller, more cohesive and maintainable codebases.
- More scalable organizations with decoupled, autonomous teams.
- The ability to upgrade, update, or even rewrite parts of the frontend in a more incremental fashion.

Once your monorepo is set up, push changes to your Git repository as usual and you'll see the commits deployed to the Vercel projects you've connected.

Conclusion

To learn more, read about subpaths and rewrites or deploy an example with micro-frontends.