



Table of Contents >

Deployment

A Nuxt application can be deployed on a Node.js server, pre-rendered for static hosting, or deployed to serverless or edge (CDN) environments.

If you are looking for a list of cloud providers that support Nuxt 3, see the [list below](#).

Node.js Server

Discover the Node.js server preset with Nitro to deploy on any Node hosting.

- ✔ **Default output format** if none is specified or auto-detected
- ✔ Loads only the required chunks to render the request for optimal cold start timing
- ✔ Useful for deploying Nuxt apps to any Node.js hosting

Entry Point

When running `nuxt build` with the Node server preset, the result will be an entry point that launches a ready-to-run Node server.

```
> node .output/server/index.mjs
```

Example

```
> $ node .output/server/index.mjs
> Listening on http://localhost:3000
```

Configuring Defaults at Runtime

This preset will respect the following runtime environment variables:

- `NITRO_PORT` or `PORT` (defaults to `3000`)
- `NITRO_HOST` or `HOST` (defaults to `'0.0.0.0'`)
- `NITRO_SSL_CERT` and `NITRO_SSL_KEY` - if both are present, this will launch the server in HTTPS mode. In the vast majority of cases, this should not be used other than for testing, and the Nitro server should be run behind a reverse proxy like nginx or Cloudflare which terminates SSL.

Using PM2

To use `pm2` , use an `ecosystem.config.js` :

```
module.exports = {
  apps: [
    {
      name: 'NuxtAppName',
      port: '3000',
      exec_mode: 'cluster',
      instances: 'max',
      script: './.output/server/index.mjs'
    }
  ]
}
```

`ecosystem.config.js`

Using Cluster Mode

You can use `NITRO_PRESET=node_cluster` in order to leverage multi-process performance using Node.js `cluster` module.

By default, the workload gets distributed to the workers with the round robin strategy.

Learn More

👉 [Read more in the Nitro documentation for node-server preset.](#)

Static Hosting

There are two ways to deploy a Nuxt application to any static hosting services:

- Static site generation (SSG) with `ssr: true` pre-renders routes of your application at build time. (This is the default behavior when running `nuxt generate`.) It will also generate `/200.html` and `/404.html` single-page app fallback pages, which can render dynamic routes or 404 errors on the client (though you may need to configure this on your static host).
- Alternatively, you can prerender your site with `ssr: false` (static single-page app). This will produce HTML pages with an empty `<div id="__nuxt"></div>` where your Vue app would normally be rendered. You will lose many of the benefits of prerendering your site, so it is suggested instead to use `<ClientOnly>` to wrap the portions of your site that cannot be server rendered (if any).

Crawl-based Pre-rendering

Use the `nuxt generate` command to build and pre-render your application using the Nitro crawler. This command is similar to `nuxt build` with the `nitro.static` option set to `true`, or running `nuxt build --prerender`.

```
> npx nuxt generate
```

That's it! You can now deploy the `.output/public` directory to any static hosting service or preview it locally with `npx serve .output/public`.

Working of the Nitro crawler:

1. Load the HTML of your application's root route (`/`), any non-dynamic pages in your `~/pages` directory, and any other routes in the `nitro.prerender.routes` array.
2. Save the HTML and `payload.json` to the `~/output/public/` directory to be served statically.
3. Find all anchor tags (``) in the HTML to navigate to other routes.

4. Repeat steps 1-3 for each anchor tag found until there are no more anchor tags to crawl.

This is important to understand since pages that are not linked to a discoverable page can't be pre-rendered automatically.

Read more about the `nuxt generate` command.

Selective Pre-rendering

You can manually specify routes that Nitro will fetch and pre-render during the build or ignore routes that you don't want to pre-render like `/dynamic` in the `nuxt.config` file:

```
defineNuxtConfig({nuxt.config.ts|js  
  nitro: {  
    prerender: {  
      routes: ['/user/1', '/user/2']  
      ignore: ['/dynamic']  
    }  
  }  
})
```

You can combine this with the `crawlLinks` option to pre-render a set of routes that the crawler can't discover like your `/sitemap.xml` or `/robots.txt` :

```
defineNuxtConfig({nuxt.config.ts|js  
  nitro: {  
    prerender: {  
      crawlLinks: true,  
      routes: ['/sitemap.xml', '/robots.txt']  
    }  
  }  
})
```

Setting `nitro.prerender` to `true` is similar to `nitro.prerender.crawlLinks` to `true` .

Read more about pre-rendering in the Nitro documentation.

Client-side Only Rendering

If you don't want to pre-render your routes, another way of using static hosting is to set the `ssr` property to `false` in the `nuxt.config` file. The `nuxi generate` command will then output an `.output/public/index.html` entrypoint and JavaScript bundles like a classic client-side Vue.js application.

```
defineNuxtConfig({nuxt.config.ts|js  
  ssr: false  
})
```

Presets


In addition to Node.js servers and static hosting services, a Nuxt 3 project can be deployed with several well-tested presets and minimal amount of configuration.

You can explicitly set the desired preset in the `nuxt.config` file:

```
export default {nuxt.config.js|ts  
  nitro: {  
    preset: 'node-server'  
  }  
}
```

... or use the `NITRO_PRESET` environment variable when running `nuxt build` :












```
> NITRO_PRESET=node-server nuxt build
```

 Check the Nitro deployment for all possible deployment presets and providers.

Supported Hosting Providers

Nuxt 3 can be deployed to several cloud providers with a minimal amount of configuration:

-  AWS
-  Azure

-  [Cleavr](#)
-  [Cloudflare](#)
-  [DigitalOcean](#)
-  [Edgio](#)
-  [Firebase](#)
-  [Heroku](#)
-  [Lagon](#)
-  [Netlify](#)
-  [Render](#)
-  [Stormkit](#)
-  [Vercel](#)

CDN Proxy

In most cases, Nuxt can work with third-party content that is not generated or created by Nuxt itself. But sometimes such content can cause problems, especially Cloudflare's "Minification and Security Options".

Accordingly, you should make sure that the following options are unchecked / disabled in Cloudflare. Otherwise, unnecessary re-rendering or hydration errors could impact your production application.

1. Speed > Optimization > Auto Minify: Uncheck JavaScript, CSS and HTML
2. Speed > Optimization > Disable "Rocket Loader™"
3. Speed > Optimization > Disable "Mirage"
4. Scrape Shield > Disable "Email Address Obfuscation"
5. Scrape Shield > Disable "Server-side Excludes"

With these settings, you can be sure that Cloudflare won't inject scripts into your Nuxt application that may cause unwanted side effects.



© 2016-2023 Nuxt - MIT
License

Enterprise

Design Kit
Nuxt Studio

NuxtLabs

