

Getting Started > Deployment

Deployment

Environment Variables



For consistency, we recommend prefixing all Auth.js environment variables with `AUTH_`. This way we can better autodetect them, and they can also be distinguished from other environment variables more easily.

Auth.js libraries require you to set an `AUTH_SECRET` environment variable. This is used to encrypt cookies and tokens. It should be a cryptographically secure random string of at least 32 characters:

`npm` `pnpm` `yarn` `bun`

```
npm exec auth secret
```

If you are using an [OAuth Provider](#), your provider will provide you with a **Client ID** and **Client Secret** that you will need to set as environment variables as well (in the case of an OIDC provider, like Auth0, a third `issuer` value might be also required, refer to the provider's specific documentation).



Auth.js supports **environment variable inference**, meaning that if you name your provider environment variables following a specific syntax, you won't need to explicitly pass them to the providers in your configuration.

Client ID's and client secrets should be named `AUTH_[PROVIDER]_ID` and `AUTH_[PROVIDER]_SECRET`. If your provider requires an issuer, that should be named `AUTH_[PROVIDER]_ISSUER`. For example:

```
AUTH_OKTA_ID=abc  
AUTH_OKTA_SECRET=abc  
AUTH_OKTA_ISSUER=abc
```

For more information, check out our [environment variables](#) page.

AUTH_SECRET

This is the only strictly required environment variable. It is the secret used to encode the JWT and encrypt things in transit. As mentioned above, we recommend at least a 32 character random string. This can be generated via the CLI with `npm exec auth secret` or via openssl with `openssl rand -base64 33`.

AUTH_TRUST_HOST

When deploying your application behind a reverse proxy, you'll need to set `AUTH_TRUST_HOST` equal to `true`. This tells Auth.js to trust the `X-Forwarded-Host` header from the reverse proxy. Auth.js will automatically infer this to be true if we detect the environment variable indicating that your application is running on one of the supported hosting providers. Currently `VERCEL` and `CF_PAGES` (Cloudflare Pages) are supported.

AUTH_URL

This environment variable is mostly unnecessary with v5 as the host is inferred from the request headers. However, if you are using a different base path, you can set this environment variable as well. For example, `AUTH_URL=http://localhost:3000/web/auth` or `AUTH_URL=https://company.com/app1/auth`


AUTH_REDIRECT_PROXY_URL

NOTE: Some providers (eg Apple) do not support [redirect proxy](#) usage.

This environment variable is designed for advanced use-cases only, when using Auth.js as a proxy for preview deploys, for example. For more details, see the [securing preview deploys](#) section below.

Serverless

1. Create the required [environment variables](#) for your desired environments. Don't forget to also add the required environment variables for your provider(s) of choice (i.e. OAuth `clientId` / `clientSecret`, etc.).
2. When using an OAuth provider, make sure the callback URL for your production URL is setup correctly. Many OAuth providers will only allow you to set 1 `callbackUrl` per OAuth application. In which case, you'll need to create separate applications for each environment (development, production, etc.). Other providers, like Google, allow you to add many `callbackUrl`s to one application.
 - By default, the callbackUrl for `next-auth` (Next.js) applications should look something like this: `https://company.com/api/auth/callback/[provider]` (replace `company.com` with your domain and `provider` with the provider name, i.e. `github`).
 - All other frameworks (`@auth/sveltekit`, `@auth/express`, etc.), by default, will use the path `/auth/callback/[provider]`.
3. Deploy! After having setup those two prerequisites, you should be able to deploy and run your Auth.js application on Netlify, Vercel, etc.

 If you are storing users in a [database](#), we recommend using a different OAuth app for development/production so that you don't mix your test and production user base.

Observability

To pass on your current user's details on to your observability tools, you can use the callbacks provided by Auth.js. For example, in the `session` callback, you could pass the `user.id` on to Sentry.

```
import * as Sentry from "@sentry/browser"
import NextAuth from "next-auth"

export const { handlers, auth, signIn, signOut } = NextAuth({
  callbacks: {
    session({ session, user }) {
      const scope = Sentry.getCurrentScope()

      scope.setUser({
        id: user.id,
        email: user.email,
      })

      return session
    },
  },
})
```

Self-hosted

Auth.js can also be deployed anywhere you can deploy your framework of choice. Check out the framework's documentation on self-hosting.

Docker

In a Docker environment, make sure to set either `trustHost: true` in your Auth.js configuration or the `AUTH_TRUST_HOST` environment variable to `true`.

Our example application is also hosted via Docker [here](#) (see the [source code](#)). Below is an example `Dockerfile` for a Next.js application using Auth.js.

Dockerfile



```
# syntax=docker/dockerfile:1
# syntax=docker/dockerfile:1
FROM node:20-alpine AS base
```

```
# Install dependencies only when needed
FROM base AS deps
# Check https://github.com/nodejs/docker-node/tree/b4117f9333da4138b03a546ec926e
RUN apk add --no-cache libc6-compat
WORKDIR /app

# Install dependencies
COPY package.json pnpm-lock.yaml* ./
RUN corepack enable pnpm && pnpm i --frozen-lockfile

# Rebuild the source code only when needed
FROM base AS builder
WORKDIR /app
COPY --from=deps /app/node_modules ./node_modules
COPY . .

# Next.js collects completely anonymous telemetry data about general usage.
# Learn more here: https://nextjs.org/telemetry
# Uncomment the following line in case you want to disable telemetry during the build
# ENV NEXT_TELEMETRY_DISABLED 1

RUN corepack enable pnpm && pnpm build

# Production image, copy all the files and run next
FROM base AS runner
WORKDIR /app

ENV NODE_ENV production
# Uncomment the following line in case you want to disable telemetry during runtime
# ENV NEXT_TELEMETRY_DISABLED 1

RUN addgroup --system --gid 1001 nodejs
RUN adduser --system --uid 1001 nextjs

COPY --from=builder /app/public ./public

# Set the correct permission for prerender cache
RUN mkdir .next
RUN chown nextjs:nodejs .next

# Automatically leverage output traces to reduce image size
# https://nextjs.org/docs/advanced-features/output-file-tracing
COPY --from=builder --chown=nextjs:nodejs /app/.next/standalone ./
COPY --from=builder --chown=nextjs:nodejs /app/.next/static ./next/static

USER nextjs
```

```
EXPOSE 3000
```

```
ENV PORT 3000
```

```
ENV HOSTNAME "0.0.0.0"
```

```
# server.js is created by next build from the standalone output
```

```
# https://nextjs.org/docs/pages/api-reference/next-config-js/output
```

```
CMD ["node", "server.js"]
```

Securing a preview deployment

NOTE: Some providers (eg Apple) do not support [redirect proxy](#) usage.

Most OAuth providers cannot be configured with multiple callback URLs or using a wildcard.

However, Auth.js **supports Preview deployments**, even **with most OAuth providers**. The idea is to have one deployment which proxies authentication requests to the dynamic URLs of your main application. So you could have 1 stable deployment, like at `auth.company.com` where you would point all your OAuth provider's `callbackUrl`s, and this application would then, upon successful authentication, redirect the user back to the preview deploy URL, like `https://git-abc123-myapp.vercel.app`. Follow these steps to get started with securing preview deploys with Auth.js.

1. Determine a stable deployment URL. For example, a deployment whose URL does not change between builds, for example. `auth.yourdomain.com` (using a subdomain is not a requirement, this can be the main site's URL too, for example.)
2. In **both the preview and stable environment**, set `AUTH_REDIRECT_PROXY_URL` to that stable deployment URL, including the path from where Auth.js handles the routes. Eg.: (`https://auth.yourdomain.com/api/auth`). If the variable is not set in the stable environment, the proxy functionality will not be enabled!
3. Update the `callbackUrl` in your OAuth provider's configuration to use the stable deployment URL. For example, for GitHub it would be `https://auth.yourdomain.com/api/auth/callback/github`.

Fun fact: all of our example apps are using the proxy functionality!

i To support preview deployments, the `AUTH_SECRET` value needs to be the same for the stable deployment and deployments that will need OAuth support.

▼ How does this work?

To support preview deployments, Auth.js requires a deployment URL that is stable across deploys as a redirect proxy server.

It will redirect the OAuth callback request to the preview deployment URL, but only when the `AUTH_REDIRECT_PROXY_URL` environment variable is set.

When a user initiates an OAuth sign-in flow on a preview deployment, we save its URL in the `state` query parameter but set the `redirect_uri` to the stable deployment.

Then, the OAuth provider will redirect the user to the stable URL mentioned above, which will verify the `state` parameter and redirect the user to the preview deployment URL if the `state` is valid. This is secured by relying on the same server-side `AUTH_SECRET` for the stable deployment and the preview deployment.

Last updated on January 9, 2025

About Auth.js

Introduction

Security

Discord Community

Download

GitHub

NPM

Acknowledgements

Contributors

Sponsors

