EcomLabs

STEP-BY-STEP GUIDE

A smaller Docker image for Next.js

The full, commented multi-stage Dockerfile for Next.js with standalone output, the .dockerignore, how to pick the base image, a script that checks static files are served, and how to measure before and after. With a checklist.

FOR	Next.js apps (App Router or Pages) deployed as a Docker image
TESTED WITH	Next.js 16.4.6 and Node 22.22.3: standalone output, server.js and the check script. The Dockerfile was not built on this machine
UPDATED	November 19, 2026
DOCS	nextjs.org/docs/app/api-reference/config/next-config-js/output · docs.docker.com/build/building/multi-stage
ARTICLE	ecomlabs.dev/g/docker-en
AUTHOR	Luis Morales · @luismoralesdev

Contents

Before you start	What you need and what was tested
Step 01	Turn on standalone
Step 02	The .dockerignore
Step 03	The full Dockerfile, commented
Step 04	Build, run and check
Step 05	Pick the base and measure
When you finish	Checklist

BEFORE YOU START

What you need and what was tested

You need Next.js 12.2 or later (15 or later for `next.config.ts`; tested with 16.4.6, the latest on npm on Nov 19, 2026), a `package-lock.json`, a `public` folder even if empty, and Docker with BuildKit.

We ran `next build` with standalone, `server.js` and the step 04 script. Not `docker build`: the Docker engine did not start on the test machine. The Dockerfile follows the official Next.js `with-docker` example.

STEP 01

Turn on standalone

```
// next.config.ts
import type { NextConfig } from "next";
const nextConfig: NextConfig = {
  output: "standalone",
};
export default nextConfig;
```

Run `npm run build` locally and look at `.next/standalone`: it should hold `server.js`, a `package.json` and a trimmed `node_modules`. It has no `public` and no `.next/static`; the Dockerfile copies those.

Files read through paths built at runtime: `outputFileTracingIncludes`. Monorepo: `outputFileTracingRoot` at the root.

STEP 02

The .dockerignore

```
# The container builds its own; yours has your OS's binaries
node_modules
.next
.git
# Next.js copies .env and .env.production into .next/standalone
.env*
!.env.example
npm-debug.log*
coverage
test-results
playwright-report
*.md
Dockerfile*
.dockerignore
```

It lives at the root of the build context, next to the Dockerfile. If you keep non-secret build values in `.env.production`, add the line `!.env.production`; the last matching rule wins.

STEP 03

The full Dockerfile, commented

```
# syntax=docker/dockerfile:1
# Exact version for all three stages (24.22.3: Nov 10, 2026 security release).
ARG NODE_VERSION=24.22.3-bookworm-slim
# ---------- 1) deps: install every dependency ----------
FROM node:${NODE_VERSION} AS deps
WORKDIR /app
# Manifest and lockfile only: this layer stays cached until deps change.
COPY package.json package-lock.json ./
# npm ci follows the lockfile and includes devDependencies (to build).
RUN npm ci --no-audit --no-fund
# ---------- 2) build: compile with output "standalone" ----------
FROM node:${NODE_VERSION} AS build
WORKDIR /app
ENV NEXT_TELEMETRY_DISABLED=1
COPY --from=deps /app/node_modules ./node_modules
# .dockerignore keeps your local node_modules from overwriting this one.
COPY . .
# NEXT_PUBLIC_ variables are inlined into the JS, so they go here.
# ARG NEXT_PUBLIC_SITE_URL
# ENV NEXT_PUBLIC_SITE_URL=$NEXT_PUBLIC_SITE_URL
RUN npm run build
# ---------- 3) runner: the only stage you ship ----------
FROM node:${NODE_VERSION} AS runner
WORKDIR /app
# HOSTNAME=0.0.0.0: Docker sets HOSTNAME to the container's hostname
# and server.js would listen on that address only.
ENV NODE_ENV=production \
    NEXT_TELEMETRY_DISABLED=1 \
    PORT=3000 \
    HOSTNAME=0.0.0.0
# Minimal server + traced node_modules.
COPY --from=build --chown=node:node /app/.next/standalone ./
# Standalone leaves these two out: without them, 404 on JS, CSS and images.
COPY --from=build --chown=node:node /app/.next/static ./.next/static
COPY --from=build --chown=node:node /app/public ./public
# No root. The node user already exists in the official image.
USER node
EXPOSE 3000
CMD ["node", "server.js"]
```

> **CAUTION**
>
> Secrets do not belong in the Dockerfile or in a `.env` inside the context: pass them when the container starts (`docker run -e`, your platform's settings or your secret manager). An `ARG` is no place for them either: Docker warns that build arguments are visible in `docker history`.

STEP 04

Build, run and check

```
docker build -t my-app:after .
docker run --rm -d --name my-app -p 3000:3000 my-app:after
node check-image.mjs http://localhost:3000 /logo.png
docker stop my-app
```

Replace `/logo.png` with a file that really is in your `public`. Save `check-image.mjs` (Node 18 or later, no dependencies):

```
// Usage: node check-image.mjs <base-url> <file-in-public>
// Example: node check-image.mjs http://localhost:3000 /logo.png
// Checks that the page, one /_next/static file and one public file return 200.
const base = process.argv[2] ?? "http://localhost:3000";
const publicFile = process.argv[3];
if (!publicFile) {
  console.error("Usage: node check-image.mjs <base-url> <file-in-public>");
  process.exit(2);
}
async function status(path) {
  try {
    const res = await fetch(new URL(path, base), { redirect: "manual" });
    return { code: res.status, body: await res.text() };
  } catch (err) {
    return { code: 0, body: "", error: err.cause?.code ?? err.message };
  }
}
const page = await status("/");
const asset = page.body.match(/\/_next\/static\/[^"'\s)]+/)?.[0];
const checks = [
  ["page /", page],
  [`static ${asset ?? "(none found in the HTML)"}`, asset ? await status(asset) : { code: 0 }],
  [`public ${publicFile}`, await status(publicFile)],
];
let failed = false;
for (const [name, r] of checks) {
  const ok = r.code === 200;
  if (!ok) failed = true;
  console.log(`${ok ? "OK  " : "FAIL"} ${r.code || r.error || "-"} ${name}`);
```

```
}
process.exit(failed ? 1 : 0);
```

It exits with code 1 if the page, the first `/_next/static` file in the HTML or the `public` file does not return 200. Against `server.js` without the static files it printed `FAIL 404` for both files; after copying them, `OK 200` for all three.

STEP 05

Pick the base and measure

TAG	LIBC	PULL	WHEN
`node:24.22.3-bookworm`	glibc	410 MB	Build stages that compile native modules
`node:24.22.3-bookworm-slim`	glibc	81 MB	Starting point for all three stages
`node:24.22.3-alpine`	musl	63 MB	When size is critical and your native deps have musl builds

Compressed layer size, linux/amd64, from the Docker Hub manifests on November 19, 2026; on disk they take more. To try alpine: `docker build --build-arg NODE_VERSION=24.22.3-alpine -t my-app:alpine .` and repeat step 04.

To measure, save your previous Dockerfile as `Dockerfile.naive` and build both from the same app:

```
docker build -f Dockerfile.naive -t my-app:before .
docker build -t my-app:after .
docker image ls my-app
docker image history my-app:after
```

`SIZE` is the cumulative space of the image and its parents. `docker image history` shows each layer with the instruction that created it: that is where you spot the heavy one. Write down both `SIZE` values with the base and the date.

WHEN YOU FINISH

Checklist

- [ ] `output: "standalone"` in `next.config`, and `.next/standalone/server.js` exists after the build.
- [ ] `.dockerignore` at the root with `node_modules`, `.next`, `.git` and `.env*`.
- [ ] Three stages on one base pinned in an `ARG` to an exact version; lockfile copied before the source.
- [ ] The final stage copies `.next/standalone`, `.next/static` and `public`.
- [ ] `PORT` and `HOSTNAME=0.0.0.0` are set; `CMD ["node", "server.js"]`.
- [ ] It runs as `USER node`, with no secrets in the image.
- [ ] `check-image.mjs` passes and you wrote down before and after `SIZE`.