Containerization Requirements Report (Automaspec)

Artifacts (Current Repo)

- `Dockerfile` (multi-stage, Next.js standalone runtime)
- `.dockerignore`
- `docker-compose.dev.yml` (development run profile)
- `docker-compose.prod.yml` (production run profile)
- `.env.example` (documented env vars)

Image Design

Base and stages

- Base image: `node:24-alpine`
- Stages:
  - `deps`: installs dependencies via pnpm (Corepack) with frozen lockfile
  - `builder`: builds Next.js (`pnpm build`)
  - `runner`: minimal runtime, runs `server.js` from the standalone output

Security

- Runtime runs as a non-root user (`nextjs`)
- Secrets are provided at runtime via env, not baked into the image

Health and ports

- `PORT` is configurable (default `3000`)
- Container exposes `${PORT}` and defines an HTTP healthcheck (`curl http://localhost:${PORT}/`)

Compose (Dev/Prod)

Both `docker-compose.dev.yml` and `docker-compose.prod.yml`:

- Run a single `app` service on `${PORT:-3000}`
- Load environment from `.env` (`env_file`)
- Include an HTTP healthcheck
- Use a dedicated bridge network (`automaspec-network`)

Differences:

- Resource limits/reservations are higher in the production compose file.

Environment Variables

Required:

- `NEXT_PUBLIC_DATABASE_URL`
- `DATABASE_AUTH_TOKEN`

Optional (feature-dependent):

- `OPENROUTER_API_KEY`
- `GEMINI_API_KEY`
- `VERCEL_URL`

How to Build and Run

1. Copy `.env.example` to `.env` and fill required values.
2. Development-style run: `pnpm docker:dev:up`
3. Production-style run: `pnpm docker:prod:up`
4. Stop: `pnpm docker:dev:down` / `pnpm docker:prod:down`

Validation Checklist

- `docker compose -f docker-compose.dev.yml up --build` succeeds
- App serves `http://localhost:3000`
- Healthcheck passes
- App can connect to the configured libSQL endpoint
- Container runs as non-root (`nextjs`)