

Deep Agent VPS Deployment Guide

This guide documents all modifications needed to deploy the Traffic Control Plane Next.js app on a self-hosted VPS using Docker/Coolify.

Table of Contents





1. [Environment Independence](#)
 2. [Prisma Schema Configuration](#)
 3. [Dockerignore Setup](#)
 4. [TypeScript Configuration](#)
 5. [Dockerfile Requirements](#)
 6. [Next.js Standalone Output](#)
 7. [Local File Storage](#)
 8. [Health Check Endpoint](#)
 9. [Prisma CLI in Runner Stage](#)
 10. [Database Seeding](#)
 11. [Docker Entrypoint](#)
 12. [Docker Compose](#)
 13. [Environment Variables](#)
 14. [Pre-Push Checklist](#)
 15. [Common Errors & Fixes](#)
 16. [Troubleshooting 502 Errors](#)
 17. [Coolify-Specific Deployment](#)
 18. [Verifying Successful Deployment](#)
-

1. Environment Independence

STATUS:  **COMPLETED**

The app has been updated to remove all Abacus AI specific dependencies:

Removed Dependencies

-  Removed `https://apps.abacus.ai/chatllm/apllm-lib.js` script from `app/layout.tsx`
-  Removed hardcoded `*.abacusai.app` domain references
-  Removed AWS S3 storage dependency
-  Removed hardcoded Abacus AI LLM API endpoints

Added Environment Variables

```
# Platform configuration (replaces hardcoded domains)
PLATFORM_DOMAIN='your-domain.com'
PLATFORM_IP='your-server-ip' # Optional, for A record DNS

# LLM API configuration (replaces Abacus AI endpoints)
LLM_API_BASE_URL='https://api.openai.com/v1' # Or any OpenAI-compatible API
LLM_API_KEY='your-api-key'
LLM_MODEL='gpt-4o-mini' # Or your preferred model
```


2. Prisma Schema Configuration

File: `nextjs_space/prisma/schema.prisma`

Required Changes

REMOVE any hardcoded output line and **ADD** Alpine binary targets:

```
generator client {
  provider = "prisma-client-js"
  binaryTargets = ["native", "linux-musl-openssl-3.0.x", "linux-musl-arm64-openssl-3.0.x"]
  // NO output line - Abacus.AI adds this automatically, it must be removed!
}
```

 **WARNING:** The system auto-adds `output = /home/ubuntu/...` after every checkpoint. ALWAYS verify and remove this line before every git push!

Binary Targets Explained

Target	Purpose
<code>native</code>	Local development machine
<code>linux-musl-openssl-3.0.x</code>	Alpine Linux x64 (Docker)
<code>linux-musl-arm64-openssl-3.0.x</code>	Alpine Linux ARM64 (Docker on ARM)

3. Dockerignore Setup

File: `.dockerignore` (at **PROJECT ROOT**, not in `nextjs_space/`)

```

# Dependencies
node_modules
nextjs_space/node_modules
.pnp
.pnp.js

# Lock files (symlinked in dev environment)
yarn.lock
nextjs_space/yarn.lock
package-lock.json
nextjs_space/package-lock.json

# Build outputs
.next
nextjs_space/.next
.build
nextjs_space/.build
out
build
dist

# Testing
coverage

# Misc
.DS_Store
*.pem

# Debug
npm-debug.log*
yarn-debug.log*
yarn-error.log*

# Local env files
.env
.env.local
.env.development.local
.env.test.local
.env.production.local

# IDE
.vscode
.idea

# Git
.git
.gitignore

# TypeScript cache
tsconfig.tsbuildinfo
nextjs_space/tsconfig.tsbuildinfo

```

⚠ CRITICAL: The `.dockerignore` must be at the PROJECT ROOT (same level as `Dockerfile`), NOT inside `nextjs_space/` !

4. TypeScript Configuration

File: `nextjs_space/tsconfig.json`

Add explicit type control to prevent “Cannot find type definition” errors:

```
{
  "compilerOptions": {
    "typeRoots": ["../node_modules/@types"],
    "types": ["node", "react", "react-dom"]
  }
}
```

TypeScript Strict Mode

Docker builds use strict TypeScript. Fix all implicit any errors:

```
// Before (error in Docker):
items.forEach(item => { ... })

// After (fixed):
items.forEach((item: typeof items[number]) => { ... })
```

5. Dockerfile Requirements (Restructured Build)

File: `Dockerfile` (at **PROJECT ROOT**, not in `nextjs_space/`)

STATUS:  **RESTRUCTURED** - Fixes overlay2 filesystem conflicts

Key Architecture Changes

The Dockerfile has been completely restructured to avoid Docker layer caching conflicts:

Stage	Working Directory	Purpose
deps	/build	Install dependencies
builder	/build	Build the Next.js app
runner	/srv/app	Fresh path, no cached layers

Why This Matters

The original approach copied the entire `standalone` directory which included a problematic `node_modules` symlink. This caused overlay2 filesystem conflicts in Docker. The new approach:

1. **Uses separate working directories** - Builder uses `/build`, runner uses `/srv/app`
2. **Selective file copying** - Only copies specific needed files, not the entire standalone directory
3. **Pre-creates node_modules** - Creates directory structure BEFORE copying to avoid symlink issues
4. **Forces standalone output** - Uses `sed` to ensure standalone mode is enabled

Complete Dockerfile Template

```

FROM node:20-alpine AS base

# ====
# Stage 1: Dependencies
# ====
FROM base AS deps
RUN apk add --no-cache libc6-compat wget openssl
WORKDIR /build

# Copy package.json (note: nextjs_space/ prefix)
COPY nextjs_space/package.json ./

# Generate fresh yarn.lock (don't copy from host - it's symlinked)
RUN yarn install --frozen-lockfile || yarn install

# ====
# Stage 2: Builder
# ====
FROM base AS builder
RUN apk add --no-cache wget openssl
WORKDIR /build

# Copy dependencies from deps stage
COPY --from=deps /build/node_modules ./node_modules
COPY nextjs_space/ ./

# Generate Prisma client
RUN npx prisma generate

# Pre-compile seed script (tsx fails silently in Docker)
RUN npx tsc scripts/seed.ts --outDir scripts/compiled --esModuleInterop \
    --module commonjs --target es2020 --skipLibCheck --types node \
    || echo "Using pre-compiled seed.js"

# Create a clean next.config.js for Docker (removes problematic experimental settings)
RUN cat > next.config.js << 'NEXTCONFIG
/** @type {import('next').NextConfig} */
const nextConfig = {
  output: 'standalone',
  eslint: { ignoreDuringBuilds: true },
  typescript: { ignoreBuildErrors: false },
  images: { unoptimized: true },
};
module.exports = nextConfig;
NEXTCONFIG

# Verify next.config.js
RUN echo "=== next.config.js ===" && cat next.config.js

# Build the application
ENV NEXT_TELEMETRY_DISABLED=1
RUN yarn build

# Verify standalone output exists
RUN ls -la .next/standalone/ && ls -la .next/standalone/.next/

# ====
# Stage 3: Runner (Fresh path - /srv/app)
# ====
FROM base AS runner
RUN apk add --no-cache wget openssl

```

```

WORKDIR /srv/app

# Create non-root user
RUN addgroup --system --gid 1001 nodejs && \
    adduser --system --uid 1001 nextjs

# Set environment
ENV NODE_ENV=production
ENV NEXT_TELEMETRY_DISABLED=1
ENV PORT=3000
ENV HOSTNAME="0.0.0.0"
ENV PATH="/srv/app/node_modules/.bin:$PATH"

# Create uploads directory for local file storage
RUN mkdir -p ./uploads/public ./uploads/private && \
    chown -R nextjs:nodejs ./uploads

# ====
# CRITICAL: Selective File Copying
# ====
# Instead of copying entire standalone directory (which has symlink issues),
# copy specific files individually:

# Copy server files
COPY --from=builder --chown=nextjs:nodejs /build/.next/standalone/server.js ./
COPY --from=builder --chown=nextjs:nodejs /build/.next/standalone/package.json ./

# Copy .next build output
COPY --from=builder --chown=nextjs:nodejs /build/.next/standalone/.next ./next

# Copy static assets
COPY --from=builder --chown=nextjs:nodejs /build/public ./public
COPY --from=builder --chown=nextjs:nodejs /build/.next/static ./next/static

# Copy Prisma schema
COPY --from=builder --chown=nextjs:nodejs /build/prisma ./prisma

# Copy compiled seed script
COPY --from=builder --chown=nextjs:nodejs /build/scripts/compiled ./scripts

# ====
# CRITICAL: Pre-create node_modules structure
# ====
# Create directories BEFORE copying to avoid symlink conflicts
RUN mkdir -p ./node_modules/.bin \
    ./node_modules/.prisma \
    ./node_modules/@prisma \
    ./node_modules/prisma \
    ./node_modules/bcryptjs

# Copy required node_modules from builder
COPY --from=builder /build/node_modules/.bin ./node_modules/.bin
COPY --from=builder /build/node_modules/.prisma ./node_modules/.prisma
COPY --from=builder /build/node_modules/@prisma ./node_modules/@prisma
COPY --from=builder /build/node_modules/prisma ./node_modules/prisma
COPY --from=builder /build/node_modules/bcryptjs ./node_modules/bcryptjs

# Copy entrypoint script (note: nextjs_space/ prefix)
COPY nextjs_space/docker-entrypoint.sh ./
RUN chmod +x docker-entrypoint.sh

# Switch to non-root user
USER nextjs

```

```
EXPOSE 3000
```

```
HEALTHCHECK --interval=30s --timeout=10s --start-period=60s --retries=3 \
  CMD wget --no-verbose --tries=1 --spider http://localhost:3000/api/health || exit
1
```

```
ENTRYPOINT ["/docker-entrypoint.sh"]
```

Key Points Summary

Issue	Solution
overlay2 symlink conflict	Use <code>/srv/app</code> as fresh runner path
Standalone node_modules symlink	Selective file copying, not full directory
Standalone output not generated	Use <code>sed</code> to force <code>output: 'standalone'</code>
Missing Prisma CLI	Pre-create dirs + copy specific modules
tsx silent failures	Pre-compile seed.ts in builder stage

6. Next.js Standalone Output

STATUS:  **DOCKERFILE CREATES CLEAN CONFIG**

The Problem

1. Using `output: process.env.NEXT_OUTPUT_MODE` with environment variables is unreliable
2. The `sed` workaround proved problematic across different environments
3. The `experimental.outputFileTracingRoot` setting (pointing to parent directory) can break standalone output in Docker

The Solution

The Dockerfile creates a clean `next.config.js` at build time:


```
# Create a clean next.config.js for Docker (removes problematic experimental settings)
RUN cat > next.config.js << 'NEXTCONFIG'
/** @type {import('next').NextConfig} */
const nextConfig = {
  output: 'standalone',
  eslint: {
    ignoreDuringBuilds: true,
  },
  typescript: {
    ignoreBuildErrors: false,
  },
  images: { unoptimized: true },
};
module.exports = nextConfig;
NEXTCONFIG

# Verify next.config.js
RUN echo "=== next.config.js ===" && cat next.config.js

# Build the application
ENV NEXT_TELEMETRY_DISABLED=1
RUN yarn build

# Verify standalone output exists
RUN ls -la .next/standalone/ && ls -la .next/standalone/.next/
```

This approach:

- Removes problematic `experimental.outputFileTracingRoot` setting
- Ensures `output: 'standalone'` is always set correctly
- Works regardless of what's in the source `next.config.js`
- Provides verification that the config is correct before building

7. Local File Storage

STATUS:  **COMPLETED**

The app uses **local filesystem storage** instead of AWS S3.

Configuration

```
# Directory for uploaded files
UPLOAD_DIR='./uploads'

# Maximum file size in bytes (default: 50MB)
MAX_FILE_SIZE=52428800
```

Directory Structure

```
uploads/
├── public/           # Publicly accessible files
│   └── 1234567890-abc123-filename.pdf
└── private/         # Authenticated access only
    └── 1234567890-def456-document.docx
```

Docker Volume

Must mount a persistent volume for uploads in `docker-compose.yml` :

```
services:
  app:
    volumes:
      - uploads-data:/srv/app/uploads # Note: /srv/app path
    environment:
      - UPLOAD_DIR=/srv/app/uploads

volumes:
  uploads-data:
    driver: local
```

8. Health Check Endpoint

File: `app/api/health/route.ts`

Ensure this endpoint exists:

```
import { NextResponse } from 'next/server';

export async function GET() {
  return NextResponse.json({ status: 'healthy' });
}

export const dynamic = 'force-static';
```

9. Prisma CLI in Runner Stage

The standalone output doesn't include CLI binaries. To run prisma commands at container startup (`db push` , `seed`):

A) Add PATH to Dockerfile runner stage:

```
ENV PATH="/srv/app/node_modules/.bin:$PATH"
```

B) Pre-create directories, then copy from builder:

```
# Create directories BEFORE copying (avoids symlink conflicts)
RUN mkdir -p ./node_modules/.bin \
            ./node_modules/.prisma \
            ./node_modules/@prisma \
            ./node_modules/prisma

# Copy from builder (note: /build path in builder stage)
COPY --from=builder /build/node_modules/.bin ./node_modules/.bin
COPY --from=builder /build/node_modules/.prisma ./node_modules/.prisma
COPY --from=builder /build/node_modules/@prisma ./node_modules/@prisma
COPY --from=builder /build/node_modules/prisma ./node_modules/prisma
```

⚠ Without these, you'll see `sh: prisma: not found` at container startup!

10. Database Seeding

The `tsx` package (used by `prisma db seed`) has many transitive dependencies that **FAIL SILENTLY** in Docker builds.

Solution: Pre-compile seed.ts to JavaScript

A) In Dockerfile builder stage:

```
RUN npx tsc scripts/seed.ts --outDir scripts/compiled --esModuleInterop \
    --module commonjs --target es2020 --skipLibCheck --types node \
    || echo "TSC failed, using pre-compiled seed.js"
```

B) Locally compile and COMMIT:

```
cd nextjs_space
npx tsc scripts/seed.ts --outDir scripts/compiled --esModuleInterop \
    --module commonjs --target es2020 --skipLibCheck --types node
git add scripts/compiled/seed.js
```

C) In runner stage:

```
# Note: /build path in builder stage
COPY --from=builder /build/scripts/compiled ./scripts
COPY --from=builder /build/node_modules/bcryptjs ./node_modules/bcryptjs
```

D) In docker-entrypoint.sh:

```
node scripts/seed.js # NOT prisma db seed
```

Seed Script Upsert - Always Update Passwords!

```
const admin = await prisma.user.upsert({
  where: { email: 'admin@example.com' },
  update: {
    passwordHash: hashedPassword, // <-- CRITICAL!
  },
  create: { ... }
});
```

11. Docker Entrypoint

File: nextjs_space/docker-entrypoint.sh

```
#!/bin/sh
set -e
echo "Starting application..."

# Wait for database to be ready
echo "Waiting for database connection..."
until node -e "
const { PrismaClient } = require('@prisma/client');
const p = new PrismaClient();
p.\$connect().then(() => process.exit(0)).catch(() => process.exit(1));
" 2>/dev/null; do
  echo "Database not ready, waiting..."
  sleep 2
done
echo "Database connected!"

# ALWAYS run database migrations to sync schema changes
echo "Running database migrations..."
npx prisma db push --skip-generate --accept-data-loss 2>&1 || {
  echo "Warning: prisma db push failed, attempting without --accept-data-loss..."
  npx prisma db push --skip-generate 2>&1 || echo "Migration skipped (may already be
in sync)"
}
echo "Database schema synchronized!"

# Check if seeding is needed
echo "Checking database state..."
NEEDS_SEED=$(node -e "
const { PrismaClient } = require('@prisma/client');
const p = new PrismaClient();
p.user.count().then(c => console.log(c === 0 ? 'true' : 'false')).catch(() => con-
sole.log('true'));
" 2>/dev/null || echo "true")

if [ "$NEEDS_SEED" = "true" ]; then
  echo "Running seed script..."
  node scripts/seed.js
else
  echo "Database has users, syncing passwords..."
  node scripts/seed.js || echo "Seed sync completed"
fi

echo "Starting Next.js server..."
exec node server.js
```

12. Docker Compose

STATUS:  CREATED

Production (docker-compose.yml)

```

version: '3.8'

services:
  app:
    build:
      context: .
      dockerfile: Dockerfile
    container_name: tcp-app
    ports:
      - "3000:3000"
    environment:
      - NODE_ENV=production
      - UPLOAD_DIR=/srv/app/uploads
    env_file:
      - nextjs_space/.env
    volumes:
      - uploads-data:/srv/app/uploads
    depends_on:
      db:
        condition: service_healthy
    restart: unless-stopped
    healthcheck:
      test: ["CMD", "wget", "--no-verbose", "--tries=1", "--spider", "http://localhost:3000/api/health"]
      interval: 30s
      timeout: 10s
      start_period: 60s
      retries: 3

  db:
    image: postgres:15-alpine
    container_name: tcp-db
    environment:
      POSTGRES_USER: ${POSTGRES_USER:-tcp}
      POSTGRES_PASSWORD: ${POSTGRES_PASSWORD:-tcp_secret_change_me}
      POSTGRES_DB: ${POSTGRES_DB:-tcp}
    volumes:
      - postgres-data:/var/lib/postgresql/data
    ports:
      - "5432:5432"
    restart: unless-stopped
    healthcheck:
      test: ["CMD-SHELL", "pg_isready -U ${POSTGRES_USER:-tcp}"]
      interval: 10s
      timeout: 5s
      retries: 5

volumes:
  postgres-data:
    driver: local
  uploads-data:
    driver: local

```

13. Environment Variables

File: nextjs_space/.env.example

Required Variables

```
# Database
DATABASE_URL='postgresql://user:password@localhost:5432/tcp'

# Authentication
NEXTAUTH_SECRET='generate-with-openssl-rand-base64-32'
NEXTAUTH_URL='https://your-domain.com'

# LLM API Provider (OpenAI-compatible)
LLM_API_BASE_URL='https://api.openai.com/v1'
LLM_API_KEY='sk-your-openai-key'
LLM_MODEL='gpt-4o-mini'
# OR use legacy env var:
# OPENAI_API_KEY='sk-your-openai-key'

# File Storage (Local)
UPLOAD_DIR='./uploads'
MAX_FILE_SIZE=52428800

# Platform Configuration
PLATFORM_DOMAIN='your-domain.com'
PLATFORM_IP='' # Optional
```

Generate Secrets

```
openssl rand -base64 32
```

14. Pre-Push Checklist

Before every git push, verify:

Prisma Schema

- [] Prisma schema has NO `output = ...` line
- [] Prisma schema HAS both linux-musl binary targets (arm64 AND x64)

File Locations

- [] `Dockerfile` is at PROJECT ROOT (not in `nextjs_space/`)
- [] `.dockerignore` is at PROJECT ROOT (not in `nextjs_space/`)

Next.js Config (handled by Dockerfile)

- [] Dockerfile creates clean `next.config.js` with `output: 'standalone'`
- [] No `experimental.outputFileTracingRoot` in Docker build

Dockerfile (Restructured Build)

- [] All `COPY` commands use `nextjs_space/` prefix for source files
- [] Builder stage uses `WORKDIR /build`
- [] Runner stage uses `WORKDIR /srv/app` (NOT `/app`)
- [] Has verification step: `RUN ls -la .next/standalone/`
- [] `ENV PATH="/srv/app/node_modules/.bin:$PATH"` set in runner

- [] Selective file copying (NOT full standalone directory):
- [] `COPY --from=builder /build/.next/standalone/server.js ./`
- [] `COPY --from=builder /build/.next/standalone/package.json ./`
- [] `COPY --from=builder /build/.next/standalone/.next ./next`
- [] Pre-creates `node_modules` directories BEFORE copying
- [] Copies `node_modules/.bin`, `.prisma`, `@prisma`, `prisma`, `bcryptjs` from `/build/`
- [] Creates uploads directories
- [] Installs `wget` and `openssl` in runner stage
- [] Entrypoint uses: `COPY nextjs_space/docker-entrypoint.sh ./`

Seeding

- [] Pre-compiled `scripts/compiled/seed.js` exists and is committed
- [] Seed script upsert includes password in update clause
- [] `docker-entrypoint.sh` runs `npx prisma db push` BEFORE seed check
- [] `docker-entrypoint.sh` uses `node scripts/seed.js` (not prisma db seed)

Docker Compose

- [] `docker-compose.yml` uses `/srv/app/uploads` volume path
 - [] No Abacus AI specific references remain in code
-

15. Common Errors & Fixes

Error	Fix
overlay2 filesystem conflict	Use <code>/srv/app</code> as runner WORKDIR, not <code>/app</code>
node_modules symlink issues	Selective file copying instead of full standalone directory
standalone output not found	Dockerfile must create clean <code>next.config.js</code> with <code>output: 'standalone'</code>
<code>yarn build</code> succeeds but no standalone	Remove <code>experimental.outputFileTracingRoot</code> - it breaks standalone in Docker
<code>sh: prisma: not found</code>	Set <code>ENV PATH="/srv/app/node_modules/.bin:\$PATH"</code> and copy <code>node_modules/.bin</code>
Cannot find module 'get-tsconfig'	Don't use <code>tsx</code> at runtime. Pre-compile <code>seed.ts</code> to JS
Cannot find type definition file for 'minimatch'	Add <code>--types node</code> flag to <code>tsc</code> command
401 Unauthorized on login	Ensure <code>seed.ts</code> upsert includes password in update clause
Prisma client not found	Pre-create dirs, then copy <code>.prisma</code> and <code>@prisma</code> from <code>/build/</code>
Container starts but seed doesn't run	Check if <code>seed.js</code> exists in <code>/srv/app/scripts/</code>
New database columns not appearing	Ensure entrypoint runs <code>prisma db push</code> before queries
<code>The table does not exist</code>	Migration not running - verify entrypoint
File uploads failing	Ensure uploads volume mounted at <code>/srv/app/uploads</code>
Files not persisting after restart	Check <code>uploads-data</code> volume uses <code>/srv/app/uploads</code>
COPY failed: file not found	Ensure <code>nextjs_space/</code> prefix on all COPY source paths
502 but container is running	App binding to <code>127.0.0.1</code> instead of <code>0.0.0.0</code>

16. Troubleshooting 502 Errors

A **502 Bad Gateway** error means the reverse proxy (Coolify/Traefik/Nginx) cannot connect to your application. Here's how to diagnose:

Step 1: Check if Container is Running

```
docker ps -a | grep traffic
# OR for Coolify:
docker ps -a | grep tcp
```

Expected: Container should show status `Up` with port `3000` exposed.

If container is not running or keeps restarting:

```
docker logs <container_name> --tail 200
```

Step 2: Common Causes & Fixes

Symptom	Cause	Fix
Container immediately exits	Missing <code>DATABASE_URL</code>	Add database connection string to env vars
"Connection refused to localhost:5432"	Database not accessible	Check database is running, use container name not <code>localhost</code>
"prisma: not found" in logs	Missing Prisma CLI	Ensure Dockerfile copies <code>node_modules/.bin</code>
"Cannot find module '@prisma/client'"	Prisma client not generated	Ensure <code>npx prisma generate</code> runs in builder stage
Container runs but 502 persists	Wrong port or hostname binding	Ensure <code>ENV PORT=3000</code> and <code>ENV HOSTNAME="0.0.0.0"</code>
Health check fails	App not starting properly	Check entrypoint logs, verify <code>server.js</code> exists

Critical: Network Binding Configuration

The app **MUST** listen on `0.0.0.0`, not `127.0.0.1`. This is configured in the Dockerfile:

```
ENV PORT=3000
ENV HOSTNAME="0.0.0.0"
```

Why this matters:

- `0.0.0.0` binds to ALL network interfaces (accessible from outside the container)
- `127.0.0.1` only binds to localhost (only accessible from INSIDE the container)

If your app binds to `127.0.0.1`, the reverse proxy (Traefik/Nginx/Coolify) cannot reach it, resulting in 502 errors.

Verification: Check container logs for the startup message:

```
Starting Next.js server...
▲ Next.js 14.x.x
- Local: http://0.0.0.0:3000
```

If you see `http://127.0.0.1:3000` instead, the binding is incorrect.

Step 3: Database Connection Issues

For Coolify with managed PostgreSQL:

```
# Use container name, not localhost
DATABASE_URL='postgresql://user:password@tcp-db:5432/tcp'
```

For external database:

```
# Use actual IP or domain
DATABASE_URL='postgresql://user:password@your-db-host.com:5432/tcp'
```

Step 4: Verify App Starts Correctly

Enter the container and check manually:

```
docker exec -it <container_name> sh
# Inside container:
ls -la /srv/app/
node server.js
```

Step 5: Coolify-Specific Checks

1. **Environment Variables:** In Coolify dashboard → Your App → Environment → Ensure all required vars are set
2. **Build Pack:** Use “Dockerfile” build pack, not “Nixpacks”
3. **Port Configuration:** Coolify should auto-detect port 3000 from `EXPOSE`
4. **Health Check:** Coolify uses the Dockerfile HEALTHCHECK - ensure `/api/health` endpoint works

Step 6: Full Rebuild

If all else fails, clear Docker cache completely:

```
docker system prune -a --volumes
docker builder prune -a
# Then rebuild
docker compose up -d --build
```

17. Coolify-Specific Deployment

Initial Setup

1. **Create New Resource** → Select “Docker Compose” or “Dockerfile”
2. **Connect Repository:** Link to `https://github.com/krocs2k/Traffic-Control-Plane`
3. **Build Configuration:**
 - Build Pack: `Dockerfile`
 - Dockerfile Location: `./Dockerfile`
 - Context: `.` (root)

Environment Variables in Coolify

Add these in Coolify’s Environment section:

```
DATABASE_URL=postgresql://tcp:your_password@tcp-db:5432/tcp
NEXTAUTH_SECRET=your-generated-secret
NEXTAUTH_URL=https://your-domain.com
LLM_API_KEY=your-openai-key
LLM_MODEL=gpt-4o-mini
```

Database Setup in Coolify

Option A: Coolify-Managed PostgreSQL

1. Create a PostgreSQL resource in Coolify
2. Use the internal connection URL provided
3. Container name format: `coolify-<project>-postgres`

Option B: External Database

1. Use your external database URL
2. Ensure firewall allows connection from VPS IP

Networking

Coolify automatically:

- Creates a network for your containers
- Configures Traefik reverse proxy
- Handles SSL certificates via Let’s Encrypt

Ensure containers are on the same network if using Coolify-managed database.

Persistent Storage

In Coolify’s Storage section, add:

- **Source:** Named volume `uploads-data`
- **Destination:** `/srv/app/uploads`

18. Verifying Successful Deployment

After deployment, verify these endpoints:

Health Check

```
curl https://your-domain.com/api/health
# Expected: {"status":"healthy"}
```

Login Page

```
curl -I https://your-domain.com/login
# Expected: HTTP 200
```

Container Logs (Healthy Start)

```
Starting application...
Waiting for database connection...
Database connected!
Running database migrations...
Database schema synchronized!
Checking database state...
Running seed script...
Starting Next.js server...
```

Files Reference

Project Structure for VPS Deployment

traffic-control-plane/	# Project root
├── Dockerfile	# <-- At root, NOT in nextjs_space/
├── .dockerignore	# <-- At root, NOT in nextjs_space/
├── docker-compose.yml	# Production compose
├── docker-compose.dev.yml	# Development compose (optional)
└── nextjs_space/	
├── package.json	
├── next.config.js	
├── docker-entrypoint.sh	# Entrypoint script
├── .env.example	# Environment template
├── prisma/	
├── schema.prisma	
├── scripts/	
├── seed.ts	
├── compiled/	
├── seed.js	# Pre-compiled seed
└── app/	
└── ...	

Default Credentials

After deployment, login with:

- **Email:** admin@tcp.local
- **Password:** admin123!

⚠ Change the default password immediately after first login!

Quick Deploy Commands

On VPS (Fresh Deploy)

```
git clone https://github.com/krocs2k/Traffic-Control-Plane.git
cd Traffic-Control-Plane

# Create .env file
cp nextjs_space/.env.example nextjs_space/.env
# Edit nextjs_space/.env with your values

# Build and run
docker compose up -d --build
```

On VPS (Update Existing)

```
cd Traffic-Control-Plane
git pull origin main

# Clear Docker cache (if having build issues)
docker system prune -a --volumes
docker builder prune -a

# Rebuild
docker compose up -d --build
```

View Logs

```
docker compose logs -f app
```