

# 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](#)
  19. [Application Features & Architecture](#)
- 

## 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/appllm-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 module resolution issues:

Stage	Working Directory	Purpose
deps	/build	Install dependencies
builder	/build	Build the Next.js app
runner	/srv/app	Fresh path, runs next start

### Why This Matters

We originally attempted to use Next.js's `output: 'standalone'` mode but encountered persistent `Cannot find module 'next'` errors. The standalone `server.js` has hardcoded module resolution paths that don't work reliably when copying between Docker stages. The new approach:

1. **Uses separate working directories** - Builder uses `/build`, runner uses `/srv/app`
2. **Full node\_modules copy** - Copies the complete `node_modules` directory (not traced deps)
3. **Uses `next start`** - Standard Next.js production server instead of standalone's `node server.js`
4. **No standalone output** - `next.config.js` does NOT include `output: 'standalone'`

### Complete Dockerfile Template

**⚠ UPDATE:** We no longer use `output: 'standalone'` due to persistent module resolution issues. Instead, we use `next start` with the full `node_modules` directory. This results in a larger image (~500MB vs ~150MB) but is much more reliable.

```

FROM node:20-alpine AS base

# Cache bust: 2026-02-11-v3 - CRITICAL: Remove server.js, use next start only
# ====
# 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"

# Clean any previous build artifacts to prevent standalone remnants
RUN rm -rf .next

# Create a clean next.config.js for Docker (NO standalone - use next start instead)
RUN cat > next.config.js << 'NEXTCONFIG'
/** @type {import('next').NextConfig} */
const nextConfig = {
  eslint: {
    ignoreDuringBuilds: true,
  },
  typescript: {
    ignoreBuildErrors: false,
  },
  images: { unoptimized: true },
};
module.exports = nextConfig;
NEXTCONFIG

# Verify next.config.js has NO standalone
RUN echo "=== next.config.js ===" && cat next.config.js && \
  echo "=== Verifying no standalone in config ===" && \
  ! grep -q "standalone" next.config.js && echo "✓ No standalone in config"

# Build the application
ENV NEXT_TELEMETRY_DISABLED=1
RUN yarn build

```

```

# Verify build output exists and NO standalone folder
RUN ls -la .next/ && \
    echo "==== Checking for standalone (should NOT exist) ===" && \
    ! test -d .next/standalone && echo "\ No standalone folder - correct!" || \
    (echo "ERROR: standalone folder exists!" && rm -rf .next/standalone && echo "Re-\
moved it")

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

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

# ====
# Copy full app with node_modules (no standalone)
# ====
# We use `next start` instead of standalone's `node server.js`
# This is more reliable but results in a larger image

COPY --from=builder --chown=nextjs:nodejs /build/package.json ./
COPY --from=builder --chown=nextjs:nodejs /build/next.config.js ./
COPY --from=builder --chown=nextjs:nodejs /build/node_modules ./node_modules
COPY --from=builder --chown=nextjs:nodejs /build/.next ./next
COPY --from=builder --chown=nextjs:nodejs /build/public ./public
COPY --from=builder --chown=nextjs:nodejs /build/prisma ./prisma

# CRITICAL: Remove any server.js if it exists (from cached standalone builds)
RUN rm -f ./server.js && \
    rm -rf ./next/standalone

# Verify structure - NO server.js should exist
RUN echo "==== Final structure ===" && ls -la ./ && \
    echo "==== .next contents ===" && ls -la ./next/ && \
    echo "==== Verifying next module ===" && ls -la ./node_modules/next/ && \
    echo "==== Verifying NO server.js ===" && \
    ! test -f ./server.js && echo "\ No server.js - will use next start" || \
    (echo "ERROR: server.js exists!" && exit 1)

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

# 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
Cannot find module 'next'	<b>Don't use standalone</b> - use <code>next start</code> with full <code>node_modules</code>
Module resolution issues	Copy full <code>node_modules</code> from builder, use <code>npx next start</code>
Missing Prisma CLI	Full <code>node_modules</code> includes all CLI tools
tsx silent failures	Pre-compile <code>seed.ts</code> in builder stage

## 6. Next.js Deployment Strategy

STATUS:  USING `next start` (NOT STANDALONE)

### Why We Don't Use Standalone

The `output: 'standalone'` option in Next.js creates a minimal deployment bundle with traced dependencies. However, we encountered persistent `Cannot find module 'next'` errors due to:

- Module resolution conflicts** - The standalone `server.js` has hardcoded module paths that don't work when copying between Docker stages
- Symlink issues** - Docker's overlay2 filesystem doesn't handle symlinks in `node_modules` well
- Incomplete traced dependencies** - Next.js's dependency tracing sometimes misses required modules

### The Solution: Use `next start` with Full `node_modules`

Instead of standalone, we use the standard `next start` command with the complete `node_modules` directory:

```
# Create a clean next.config.js for Docker (NO standalone)
RUN cat > next.config.js << 'NEXTCONFIG'
/** @type {import('next').NextConfig} */
const nextConfig = {
  eslint: {
    ignoreDuringBuilds: true,
  },
  typescript: {
    ignoreBuildErrors: false,
  },
  images: { unoptimized: true },
};
module.exports = nextConfig;
NEXTCONFIG
```

### Entrypoint uses:

```
exec npx next start -p 3000 -H 0.0.0.0
```

### Trade-offs

Aspect	Standalone	next start (Current)
Image Size	~150MB	~500MB
Reliability	✗ Module resolution issues	✓ Works reliably
Complexity	High (traced deps)	Low (standard deployment)
Startup Time	Faster	Slightly slower

We chose reliability over image size. The larger image is acceptable for a self-hosted VPS deployment.

## 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`

**STATUS:**  **UPDATED (v4)** - Now includes runtime server.js removal to handle cached Docker images

```

#!/bin/sh
set -e
echo "==== ENTRYPPOINT v4 - 2026-02-12 ==="
echo "Starting application..."

# CRITICAL: Remove server.js if it exists (from cached standalone builds)
if [ -f "/srv/app/server.js" ]; then
    echo "WARNING: Found cached server.js - removing it!"
    rm -f /srv/app/server.js
fi
if [ -d "/srv/app/.next/standalone" ]; then
    echo "WARNING: Found cached standalone folder - removing it!"
    rm -rf /srv/app/.next/standalone
fi

# 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..."

# Final check - absolutely ensure no server.js
if [ -f "/srv/app/server.js" ]; then
    echo "FATAL: server.js still exists after cleanup! Removing..."
    rm -f /srv/app/server.js
fi

echo "Contents of /srv/app:"

```

```
ls -la /srv/app/
echo "Using next start (NOT node server.js)..."
exec node ./node_modules/next/dist/bin/next start -p 3000 -H 0.0.0.0
```

## Key Features of v4 Entrypoint

Feature	Purpose
<b>Version identifier</b>	==== ENTRYPPOINT v4 === in logs confirms new code is running
<b>Runtime server.js removal</b>	Deletes server.js at startup even if cached image has it
<b>Standalone folder cleanup</b>	Removes .next/standalone if it exists
<b>Final check before start</b>	Double-checks no server.js remains
<b>Directory listing</b>	Shows /srv/app/ contents for debugging
<b>Direct node path</b>	Uses node ./node_modules/next/dist/bin/next instead of npx

**⚠ Note:** The runtime cleanup handles the case where Docker/Coolify uses a cached image from a previous standalone build. Even if the Dockerfile changes aren't picked up, the entrypoint will fix it at runtime.

---

## 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` **WITHOUT** `output: 'standalone'`
- [ ] No `experimental.outputFileTracingRoot` in Docker build

### Dockerfile (Using `next start`)

- [ ] 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/`
- [ ] `ENV PATH="/srv/app/node_modules/.bin:$PATH"` set in runner

- [ ] Copy full node\_modules: `COPY --from=builder /build/node_modules ./node_modules`
- [ ] Copy .next build: `COPY --from=builder /build/.next ./next`
- [ ] Verification step: `RUN ls -la ./node_modules/next/`
- [ ] Creates uploads directories
- [ ] Installs `wget`, `openssl`, and `bash` in runner stage
- [ ] Entrypoint uses: `COPY nextjs_space/docker-entrypoint.sh ./`

## Entrypoint

- [ ] Uses `npx next start -p 3000 -H 0.0.0.0` (NOT `node server.js`)

## Docker Cache (Important After Major Changes)

- [ ] If switching from standalone to `next start`, force a full rebuild (see Section 15.1)
- [ ] Update cache bust comment in Dockerfile if needed: `# Cache bust: YYYY-MM-DD-vX`

## 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
<b>overlay2 filesystem conflict</b>	Use <code>/srv/app</code> as runner WORKDIR, not <code>/app</code>
<b>Cannot find module 'next' with <code>/srv/app/server.js</code></b>	<b>Docker cache issue</b> - see Section 15.1 below
<b>node_modules symlink issues</b>	Copy full node_modules from builder (not traced deps from standalone)
<code>sh: prisma: not found</code>	Set ENV <code>PATH="/srv/app/node_modules/.bin:\$PATH"</code> - full node_modules includes CLI
<code>Cannot find module 'get-tsconfig'</code>	Don't use tsx at runtime. Pre-compile seed.ts to JS
<code>Cannot find type definition file for 'minimatch'</code>	Add <code>--types</code> node flag to tsc command
401 Unauthorized on login	Ensure seed.ts 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 seed.js 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>
<code>bash: executable file not found (code 127)</code>	Alpine Linux doesn't have bash by default
<code>XX002 PostgreSQL index error</code>	Database index corruption

## 15.1 Docker Cache Issues: Cannot find module 'next' with server.js

**Symptom:** Container logs show:

```
Error: Cannot find module 'next'
Require stack:
- /srv/app/server.js
```

**Root Cause:** Docker/Coolify is using a **cached image** from a previous build that used `output: 'standalone'`. The cached runner stage still contains the old `server.js` file even though the source code has been updated.

### Solution 1: Force Rebuild in Coolify

1. Go to your app in Coolify dashboard
2. Click **Deployments** → **Rebuild**
3. Enable “**Force rebuild**” or “**No cache**” checkbox
4. Click Deploy

### Solution 2: Clear Docker Cache on VPS

```
# SSH into your VPS
ssh user@your-vps

# Clear all Docker cache and images
docker system prune -a --volumes -f
docker builder prune -a -f

# Then trigger a new deployment from Coolify
```

### Solution 3: Add Build Command Options

In Coolify’s build settings, add to Docker build options:

```
--no-cache --pull
```

**Verification:** After rebuild, container logs should show:

```
Starting Next.js server...
  ▲ Next.js 14.X.X
    - Local:      http://0.0.0.0:3000
  ✓ Ready in Xs
```

**NOT** `/srv/app/server.js` being executed.

## 15.2 Dockerfile Safeguards Against Cached Standalone

The current Dockerfile includes safeguards to prevent this issue:

```
# In builder stage - clean any previous build artifacts
RUN rm -rf .next

# Verify no standalone in next.config.js
RUN ! grep -q "standalone" next.config.js && echo "✓ No standalone in config"

# Verify no standalone folder after build
RUN ! test -d .next/standalone && echo "✓ No standalone folder"

# In runner stage - forcefully remove any server.js
RUN rm -f ./server.js && rm -rf ./next/standalone

# Fail build if server.js still exists
RUN ! test -f ./server.js && echo "✗ No server.js - will use next start"
```

The Dockerfile also includes a **cache bust comment** that can be updated to force rebuilds:

```
# Cache bust: 2026-02-11-v3 - CRITICAL: Remove server.js, use next start only
```

To force a rebuild, increment the version or change the date in this comment.

## 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:

```
# Bash is now installed in the container
docker exec -it <container_name> bash

# Or use sh as fallback
docker exec -it <container_name> sh

# Inside container:
ls -la /srv/app/

# Verify NO server.js exists (we use next start, not standalone)
test -f server.js && echo "ERROR: server.js exists - cached image!" || echo "✓ No server.js"

# Manually test starting the app
npx next start -p 3000 -H 0.0.0.0
```

**i Note:** The Dockerfile now installs `bash` in the runner stage (`apk add --no-cache wget openssl bash`). If you're using an older image without `bash`, use `sh` instead.

**⚠️ IMPORTANT:** If you see `server.js` in `/srv/app/`, you have a cached Docker image. Force a rebuild with `--no-cache` (see Section 15.1).

## 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 - v4 Entrypoint)

```
== ENTRYPPOINT v4 - 2026-02-12 ==
Starting application...
Waiting for database connection...
Database connected!
Running database migrations...
Database schema synchronized!
Checking database state...
Running seed script...
... (seed output) ...
Starting Next.js server...
Contents of /srv/app:
total XX
drwxr-xr-x  ... .next
drwxr-xr-x  ... node_modules
-rw-r--r--  ... package.json
drwxr-xr-x  ... prisma
drwxr-xr-x  ... public
drwxr-xr-x  ... scripts
drwxr-xr-x  ... uploads
Using next start (NOT node server.js)...
  ▲ Next.js 14.x.x
  - Local:      http://0.0.0.0:3000
  - Network:    http://0.0.0.0:3000
```

Ready in Xs

## What to Look For in Logs

Log Message	Meaning
== ENTRYPPOINT v4 ==	✓ New entrypoint is running
WARNING: Found cached server.js	⚠ Cached image detected, but cleaned up
Using next start (NOT node server.js)	✓ Correct startup method
✓ Ready in Xs	✓ App is running successfully
node server.js or /srv/app/server.js	✗ OLD IMAGE - force rebuild!

**Note:** The “Ready” message confirms the app is running with `next start`. If you don’t see `== ENTRYPPOINT v4 ==`, the image wasn’t rebuilt properly.

## 19. Application Features & Architecture

This section documents the key features implemented in the Traffic Control Plane application.

## Authentication System

### Standard Login

- Email/password authentication via NextAuth.js
- Session-based authentication with JWT tokens
- Password hashing using bcryptjs

### Multi-Factor Authentication (MFA)

The application supports TOTP-based MFA with backup codes.

#### MFA Flow:

1. **Setup** (`/api/auth/mfa/setup`): Generates MFA secret, QR code URL, and 10 backup codes
2. **Verify** (`/api/auth/mfa/verify`): Verifies TOTP token and enables MFA
3. **Login with MFA**: If MFA enabled, login requires 6-digit TOTP or 8-character backup code
4. **Disable** (`/api/auth/mfa/disable`): Requires MFA token or password to disable
5. **Regenerate Backup Codes** (`/api/auth/mfa/backup-codes`): Requires MFA token verification

#### Key Files:

File	Purpose
<code>lib/mfa.ts</code>	MFA utilities (secret generation, TOTP verification, backup codes)
<code>lib/auth-options.ts</code>	NextAuth configuration with MFA integration
<code>app/api/auth/mfa/*</code>	MFA API endpoints
<code>app/(auth)/login/page.tsx</code>	Login page with MFA step
<code>app/(dashboard)/profile/page.tsx</code>	MFA setup/management UI

#### Dependencies:

- `otplib` - TOTP generation and verification
- `qrcode` - QR code generation for authenticator apps

## Password Reset with MFA

- `GET /api/auth/reset-password?token=X` - Validates token and returns `mfaRequired` status
- `POST /api/auth/reset-password` - Accepts token, password, and optional `mfaToken`

## Traffic Management Features

### Routing Policies

- Priority-based routing rules with conditions and actions
- AI Assistant for generating routing policy JSON via natural language
- Manual JSON entry option

#### AI Assistant Integration:

- Endpoint: `/api/routing-policies/ai-assist`
- Configurable LLM via environment variables:

```
env
  LLM_API_BASE_URL='https://api.openai.com/v1'
  LLM_API_KEY='your-key'
  LLM_MODEL='gpt-4o-mini'
```

## Backend Clusters

- Cluster management with health status tracking
- Cluster-level metrics and monitoring

## Read Replicas

- Lag-aware replica selection algorithm
- Status tracking: SYNCED, LAGGING, CATCHING\_UP, OFFLINE
- Manual and automatic replica selection

## Load Balancing

- Multiple algorithm support (round-robin, least-connections, weighted, etc.)
- Configuration management per cluster

## Canary/A/B Testing (Experiments)

### Experiment Lifecycle:

1. Create experiment with variants and traffic allocation
2. Start experiment (status: RUNNING)
3. Collect metrics per variant
4. Complete or abort experiment

### API Endpoints:

Endpoint	Methods	Purpose
/api/experiments	GET, POST	List/create experiments
/api/experiments/[id]	GET, PATCH, DELETE	Manage single experiment
/api/experiments/[id]/metrics	GET	Get experiment metrics

## Alerting System

- Alert rules with configurable thresholds
- Alert channels (email, webhook, etc.)
- Alert states: triggered, acknowledged, resolved, silenced

## Audit Logging

All significant actions are logged to the `AuditLog` table:

### Audit Action Categories:

- User actions: login, logout, password changes
- MFA actions: setup, enable, disable, backup code regeneration
- Experiment actions: created, updated, deleted
- Load balancer actions: config created/updated/deleted
- Alert actions: rule changes, acknowledgments, resolutions

**Key File:** `lib/audit.ts` - Defines `AuditAction` types and `createAuditLog()` function

## API Architecture

### Authentication Pattern

All protected API routes follow this pattern:

```

import { getServerSession } from 'next-auth';
import { authOptions } from '@/lib/auth-options';

export async function GET(request: NextRequest) {
  const session = await getServerSession(authOptions);
  if (!session?.user) {
    return NextResponse.json({ error: 'Unauthorized' }, { status: 401 });
  }
  // ... route logic
}

```

## BigInt Serialization

Some database fields (e.g., `totalRequests`, `totalErrors` in metrics) are BigInt. Convert to Number before JSON serialization:

```

const serializedSnapshot = {
  ...latestSnapshot,
  totalRequests: Number(latestSnapshot.totalRequests),
  totalErrors: Number(latestSnapshot.totalErrors),
};

```

## Database Schema Notes

### Prisma Configuration:

```

generator client {
  provider = "prisma-client-js"
  binaryTargets = ["native", "linux-musl-openssl-3.0.x", "linux-musl-arm64-
openssl-3.0.x"]
  // NO output line - system auto-adds this, must be removed before push
}

```

### Key Models:

- `User` - With MFA fields: `mfaSecret`, `mfaEnabled`, `mfaBackupCodes`, `mfaVerifiedAt`
- `Experiment` - Canary/A/B test definitions
- `ExperimentVariant` - Traffic variants within experiments
- `RoutingPolicy` - Traffic routing rules
- `ReadReplica` - Database replica configurations
- `AlertRule`, `AlertChannel`, `Alert` - Alerting system
- `AuditLog` - Action audit trail

## Environment Variables Summary

```
# Database
DATABASE_URL='postgresql://user:password@host:5432/db'

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

# LLM API (for AI routing assistant)
LLM_API_BASE_URL='https://api.openai.com/v1'
LLM_API_KEY='sk-your-key'
LLM_MODEL='gpt-4o-mini'

# File Storage
UPLOAD_DIR='./uploads'
MAX_FILE_SIZE=52428800
```

## Files Reference

### Project Structure for VPS Deployment

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

## 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
```