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

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
`native`	Local development machine
`linux-musl-openssl-3.0.x`	Alpine Linux x64 (Docker)
`linux-musl-arm64-openssl-3.0.x`	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
out
nextjs_space/out

# Environment files (provided at runtime)
.env
nextjs_space/.env
.env.*
!.env.example

# Git
.git
.gitignore

# IDE
.vscode
.idea
*.swp
*.swo

# Documentation
*.md
!README.md

# Miscellaneous
.DS_Store
Thumbs.db
*.log
npm-debug.log*
yarn-debug.log*
yarn-error.log*

# Deployment files for reference (not needed in container)
DEEP-AGENT-*.md
DEEP-AGENT-*.pdf
docker-compose*.yml

# Test and coverage
coverage
.nyc_output

# Abacus.AI specific
.abacus.donotdelete
opt/
```

4. TypeScript Configuration

File: `nextjs_space/tsconfig.json`

No special changes needed. Default Next.js configuration works.

---

5. Dockerfile Requirements (v11)

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

STATUS: ✅ **v11** - Compatible with deployment platforms that run `node server.js`

Key Architecture Changes

The Dockerfile has been restructured to:
1. Use separate working directories (Builder: `/build`, Runner: `/srv/app`)
2. Copy full `node_modules` (not standalone traced deps)
3. Include a `server.js` that **spawns** `next start` (for platform compatibility)
4. Use `next.config.docker.js` to prevent standalone mode

Why server.js (v11 Approach)

Many deployment platforms (Coolify, Dokploy, Railway) are configured to run `node server.js` by default. Instead of fighting this:

- We provide a `server.js` that **doesn't require('next')** directly
- It spawns `node ./node_modules/next/dist/bin/next start` as a child process
- This avoids module resolution issues while satisfying the platform's expectations

Complete Dockerfile Template (v11)

```
FROM node:20-alpine AS base

# ============================================
# v11 - 2026-02-12 - WORKING SERVER.JS
# ============================================

FROM base AS deps
RUN apk add --no-cache libc6-compat wget openssl
WORKDIR /build
COPY nextjs_space/package.json ./
RUN yarn install --frozen-lockfile || yarn install

FROM base AS builder
RUN apk add --no-cache wget openssl
WORKDIR /build
COPY --from=deps /build/node_modules ./node_modules
COPY nextjs_space/ ./

# Use clean next.config.js (no standalone)
RUN cp next.config.docker.js next.config.js && \
    echo "=== next.config.js (v11) ===" && cat next.config.js

RUN npx prisma generate

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

RUN rm -rf .next

ENV NEXT_TELEMETRY_DISABLED=1
ENV NEXT_OUTPUT_MODE=""
RUN yarn build

# Verify NO standalone was created
RUN ! test -d .next/standalone || (echo "Removing standalone" && rm -rf .next/stan-
dalone)

# ============================================
# PRODUCTION IMAGE v11
# ============================================
FROM base AS runner
RUN apk add --no-cache wget openssl bash

WORKDIR /srv/app

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

ENV NODE_ENV=production
ENV NEXT_TELEMETRY_DISABLED=1
ENV PORT=3000
ENV HOSTNAME="0.0.0.0"

RUN mkdir -p ./uploads/public ./uploads/private && \
    chown -R nextjs:nodejs ./uploads

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

# Copy server.js that spawns next start (works with deployment platforms)
COPY --from=builder --chown=nextjs:nodejs /build/server.js ./server.js

# Verify
RUN echo "=== v11 VERIFICATION ===" && \
    ls -la ./server.js && \
    ls -la ./node_modules/next/ > /dev/null && \
    echo "OK: server.js and next module exist"

# Create startup script (embedded in Dockerfile)
RUN cat > /srv/app/start.sh << 'STARTSCRIPT'
#!/bin/bash
set -e

echo ""
echo "========================================"
echo "  TCP v11 - $(date -u +%Y-%m-%dT%H:%M:%SZ)"
echo "========================================"
echo ""

# Wait for database
echo "Connecting to database..."
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
  sleep 2
done
echo "OK: Database connected"

# Sync schema
npx prisma db push --skip-generate --accept-data-loss 2>&1 || \
  npx prisma db push --skip-generate 2>&1 || true
echo "OK: Schema synced"

# Seed if needed
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 "Seeding database..."
  node scripts/seed.js
else
  echo "Database has data, syncing..."
  node scripts/seed.js || true
fi

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

RUN chmod +x /srv/app/start.sh
```

```
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

CMD ["/srv/app/start.sh"]
```

---

6. Next.js Deployment Strategy

STATUS: ✅ v11 - Using server.js wrapper with `next start`

The Problem

Many deployment platforms (Coolify, Dokploy, etc.) have a hardcoded start command that runs `node server.js`. Previous attempts to:
- Remove server.js and use `next start` directly
- Embed the entrypoint in the Dockerfile
- Clear Docker cache

...all failed because the platform **always** runs `node server.js` regardless of Dockerfile CMD.

The Solution: A Working server.js

Instead of fighting the platform, we provide a `server.js` that works:

File: `nextjs_space/server.js`

```
#!/usr/bin/env node
// server.js v11 - Works even when deployment platform runs "node server.js"
const { spawn } = require('child_process');
const path = require('path');

console.log('');
console.log('========================================');
console.log('  TCP server.js v11 - 2026-02-12');
console.log('========================================');
console.log('');

const nextBin = path.join(__dirname, 'node_modules', 'next', 'dist', 'bin', 'next');

console.log('Starting Next.js via:', nextBin);

const child = spawn('node', [nextBin, 'start', '-p', process.env.PORT || '3000', '-
H', '0.0.0.0'], {
  stdio: 'inherit',
  cwd: __dirname
});

child.on('error', (err) => {
  console.error('Failed to start:', err);
  process.exit(1);
});

child.on('exit', (code) => {
  process.exit(code || 0);
});
```

Why This Works

Approach	Problem	v11 Solution
`require('next')`	Module resolution fails in Docker	Don't require - use `spawn()`
Standalone `server.js`	Hardcoded paths don't work	Not using standalone mode
`npx next start`	Platform ignores Dockerfile CMD	Platform runs our server.js which spawns next

next.config.docker.js

File: `nextjs_space/next.config.docker.js`

A clean config file that prevents standalone mode:

```
/** @type {import('next').NextConfig} */
const nextConfig = {
  eslint: { ignoreDuringBuilds: true },
  typescript: { ignoreBuildErrors: false },
  images: { unoptimized: true },
};
module.exports = nextConfig;
```

The Dockerfile copies this over `next.config.js` during build to ensure no standalone mode.

---

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`:

```yaml
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

With full node_modules copy, Prisma CLI is automatically available.

Verify in Dockerfile:

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

---

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:

```
COPY --from=builder /build/scripts/compiled ./scripts
```

D) In startup script:

```
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 / Startup

STATUS: ✅ **v11** - Startup script embedded in Dockerfile

The startup logic is now embedded directly in the Dockerfile as `start.sh`. This ensures the correct startup process is always used.

What start.sh Does

1. **Prints version banner** - `TCP v11 - <timestamp>` confirms correct image
2. **Waits for database** - Retries connection until PostgreSQL is ready
3. **Syncs schema** - Runs `prisma db push` to apply any schema changes
4. **Seeds if needed** - Checks user count, runs seed.js if empty
5. **Starts Next.js** - Runs `node server.js` which spawns `next start`

Expected Startup Sequence

```
==========================================
  TCP v11 - 2026-02-12T03:00:00Z
==========================================

Connecting to database...
OK: Database connected
OK: Schema synced
Seeding database...
... (seed output) ...

Starting Next.js...

==========================================
  TCP server.js v11 - 2026-02-12
==========================================

Starting Next.js via: /srv/app/node_modules/next/dist/bin/next
  ▲ Next.js 14.x.x
  - Local:        http://0.0.0.0:3000
 ✓ Ready in Xs
```

---

12. Docker Compose

STATUS: ✅ **CREATED**

Production (`docker-compose.yml`)

```yaml
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/)
- [ ] `server.js` exists in `nextjs_space/` directory
- [ ] `next.config.docker.js` exists in `nextjs_space/` directory

server.js

- [ ] Uses `spawn()` to run next start (NOT `require('next')`)
- [ ] Logs version banner: `TCP server.js v11`

Dockerfile (v11)

- [ ] All `COPY` commands use `nextjs_space/` prefix for source files
- [ ] Builder stage uses `WORKDIR /build`
- [ ] Runner stage uses `WORKDIR /srv/app`
- [ ] Copies `next.config.docker.js` over `next.config.js`

- [ ] Sets `ENV NEXT_OUTPUT_MODE=""`
- [ ] Copies `server.js` to runner
- [ ] Embeds `start.sh` inline
- [ ] `CMD ["/srv/app/start.sh"]`

Seeding

- [ ] Pre-compiled `scripts/compiled/seed.js` exists
- [ ] Seed script upsert includes password in update clause

---

15. Common Errors & Fixes

Error	Fix
`Cannot find module 'next'` from `server.js`	Use v11 server.js with `spawn()` instead of `require()`
overlay2 filesystem conflict	Use `/srv/app` as runner WORKDIR, not `/app`
Standalone mode still being used	Ensure `next.config.docker.js` copied, `NEXT_OUTPUT_MODE=""` set
node_modules symlink issues	Copy full node_modules from builder
`sh: prisma: not found`	Set `ENV PATH="/srv/app/node_modules/.bin:$PATH"`
`Cannot find module 'get-tsconfig'`	Don’t use tsx at runtime. Pre-compile seed.ts to JS
401 Unauthorized on login	Ensure seed.ts upsert includes password in update clause
Container starts but seed doesn’t run	Check if seed.js exists in `/srv/app/scripts/`
New database columns not appearing	Ensure start.sh runs `prisma db push` before queries
COPY failed: file not found	Ensure `nextjs_space/` prefix on all COPY source paths
502 but container is running	App binding to wrong host - ensure `HOSTNAME="0.0.0.0"`
Prisma generates to wrong path	Remove `output` line from prisma schema

15.1 Platform Runs `node server.js` Regardless of Dockerfile CMD

Symptom: You’ve changed the Dockerfile to use `npx next start`, but logs show:

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

Root Cause: Deployment platforms (Coolify, Dokploy, etc.) have a hardcoded start command that runs `node server.js`, ignoring the Dockerfile's CMD.

Solution: Use v11 approach - provide a `server.js` that works:

```
const { spawn } = require('child_process');
const path = require('path');

const nextBin = path.join(__dirname, 'node_modules', 'next', 'dist', 'bin', 'next');

spawn('node', [nextBin, 'start', '-p', process.env.PORT || '3000', '-H', '0.0.0.0'], {
  stdio: 'inherit',
  cwd: __dirname
});
```

This server.js:
- ✅ Exists (platform is happy)
- ✅ Doesn't `require('next')` directly (no module resolution issues)
- ✅ Spawns Next.js correctly

15.2 Docker Cache Issues

If you're seeing old code running after deploying changes:

Solution 1: Force Rebuild in Coolify/Dokploy
1. Go to your app in the dashboard
2. Click **Deployments → Rebuild**
3. Enable **"Force rebuild"** or **"No cache"** checkbox
4. Click Deploy

Solution 2: Clear Docker Cache on VPS

```
ssh user@your-vps
docker system prune -a --volumes -f
docker builder prune -a -f
```

Verification: Look for version banners in logs:

```
========================================
  TCP v11 - 2026-02-12T...
========================================

...

========================================
  TCP server.js v11 - 2026-02-12
========================================
```

---

16. Troubleshooting 502 Errors

A **502 Bad Gateway** error means the reverse proxy cannot connect to your application.

Step 1: Check if Container is Running

```
docker ps -a | grep tcp
```

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 `DATABASE_URL`	Add database connection string to env vars
“Connection refused to localhost:5432”	Database not accessible	Use container name not `localhost`
“prisma: not found” in logs	Missing Prisma CLI	Ensure full node_modules copied
“Cannot find module ‘@prisma/client’“	Prisma client not generated	Ensure `npx prisma generate` runs in builder
Container runs but 502 persists	Wrong hostname binding	Ensure `ENV HOSTNAME="0.0.0.0"`
`Cannot find module 'next'`	Wrong server.js	Use v11 server.js with `spawn()`

Step 3: Test Locally

```
# Build and run locally
docker build -t tcp-test .
docker run --rm -p 3000:3000 \
  -e DATABASE_URL="postgresql://..." \
  -e NEXTAUTH_SECRET="test" \
  -e NEXTAUTH_URL="http://localhost:3000" \
  tcp-test

# Check if app responds
curl http://localhost:3000/api/health
```

---

17. Coolify-Specific Deployment

Application Settings

Setting	Value
Build Pack	Dockerfile
Dockerfile Location	`./Dockerfile`
Port	3000
Health Check Path	`/api/health`

Environment Variables

Add these in Coolify's Environment Variables section:

```
DATABASE_URL=postgresql://user:pass@db:5432/tcp
NEXTAUTH_SECRET=your-generated-secret
NEXTAUTH_URL=https://your-domain.com
LLM_API_BASE_URL=https://api.openai.com/v1
LLM_API_KEY=sk-your-key
LLM_MODEL=gpt-4o-mini
```

Force Rebuild After Changes

If changes aren't being picked up:
1. Go to Deployments tab
2. Click "Redeploy" with "No cache" option
3. Or SSH to VPS and run: `docker system prune -a -f`

---

18. Verifying Successful Deployment

After deployment, verify these:

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 - v11)

```
==========================================
  TCP v11 - 2026-02-12T03:00:00Z
==========================================

Connecting to database...
OK: Database connected
OK: Schema synced
Database has data, syncing...
... (seed output if needed) ...

Starting Next.js...

==========================================
  TCP server.js v11 - 2026-02-12
==========================================

Starting Next.js via: /srv/app/node_modules/next/dist/bin/next
  ▲ 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
`TCP v11 - <timestamp>`	✅ Correct start.sh is running
`TCP server.js v11`	✅ Correct server.js is running
`Starting Next.js via: /srv/app/node_modules/next/dist/bin/next`	✅ Correct startup method
`✓ Ready in Xs`	✅ App is running successfully
`Cannot find module 'next'` at `require`	❌ OLD server.js - update to v11
No version banner	❌ Image wasn't rebuilt - force rebuild

---

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
`lib/mfa.ts`	MFA utilities (secret generation, TOTP verification, backup codes)
`lib/auth-options.ts`	NextAuth configuration with MFA integration
`app/api/auth/mfa/*`	MFA API endpoints
`app/(auth)/login/page.tsx`	Login page with MFA step
`app/(dashboard)/profile/page.tsx`	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
- Load balancing configuration
- Health check monitoring

Read Replicas

- Lag-aware replica selection
- Manual and automatic replica selection
- Real-time lag monitoring

Experiments (A/B Testing & Canary)

- A/B test and canary deployment experiments
- Variant management with traffic allocation
- Metrics tracking and analysis

Alerting

- Alert rules with configurable thresholds
- Alert channels (email, Slack, webhook)
- Alert acknowledgment and resolution

Demo Credentials

Default users created by seed script (password: `password123`):

Email	Role	Organization
john@doe.com	OWNER	-
alice@acme.com	ADMIN	Acme Corp
bob@acme.com	OPERATOR	Acme Corp
carol@techstart.com	OWNER	TechStart Inc
dave@techstart.com	VIEWER	TechStart Inc
eve@external.com	AUDITOR	Acme Corp

Version History

Version	Date	Changes
v11	2026-02-12	Working server.js with spawn(), platform-compatible
v10	2026-02-12	Fixed Prisma schema output path
v9	2026-02-12	Added next.config.docker.js, ENV NEXT_OUTPUT_MODE
v8	2026-02-12	Removed server.js, npx next start
v7	2026-02-11	Custom server.js that spawns next
v4-v6	2026-02-11	Various standalone removal attempts
v3	2026-02-11	Switch from standalone to next start
v1-v2	2026-02-10	Initial standalone approach

Last Updated: 2026-02-12 (v11)
GitHub: https://github.com/krocs2k/Traffic-Control-Plane