🧭 UPRISE RUNBOOK — Developer & Agent Operations Manual

Repository: `music-community-platform`
Last Updated: November 12, 2025 (America/Chicago)
Applies To: All UPRISE Agents (DeepAgent, Claude Code, Codex CLI, Cursor, etc.)

🏗️ Core Directives

File	Purpose	Status
STRATEGY_CRITICAL_INFRA_NOTE.md (./STRATEGY_CRITICAL_INFRA_NOTE.md)	Defines “DeepAgent = Foundry only” rule; production targets (Vercel, Fly, AWS, Neon).	🔴 Critical
PHASE1_COMPLETION_REPORT.md (./PHASE1_COMPLETION_REPORT.md)	Certifies monorepo foundation; establishes readiness for Phase 2.	✅ Complete

> All agents MUST read the two files above before running any task.

⚙️ Development Environment

See ENVIRONMENTS.md (./ENVIRONMENTS.md) for full setup.

Summary:
- **Web / API / Socket / Workers:** WSL 2 (Ubuntu 22.04+ recommended)
- **Mobile (RN 0.66.x):** Windows non-admin PowerShell (Hermes, Gradle 7.0.2, JDK 11)
- **Node:** v22.x (fnm or nvm)
- **Package Manager:** pnpm 9.x (corepack)
- **Database:** Postgres with **PostGIS** (dev: DeepAgent container; prod: Neon or AWS RDS)
- **Optional:** Docker for local workers/DB

🧱 Monorepo Structure

See PROJECT_STRUCTURE.md (./PROJECT_STRUCTURE.md) for details and conventions.

```
apps/
  web/        → Next.js 15 (Vercel)
  api/        → NestJS (Fly.io / App Runner)
  socket/     → Socket.IO (Fly.io / App Runner)
  workers/
    transcoder/ → FFmpeg Node worker (AWS Fargate / Fly.io)
packages/
  ui/         → Shared components
  types/      → Zod schemas → OpenAPI
  sdk/        → Generated client for web/api
infra/
  prisma/     → Prisma schema + PostGIS migrations + seeds
```

Strict Web-Tier Contract

- No DB access, no secrets, no server actions that mutate state in `apps/web`.
- All mutations go through `apps/api`.
- Realtime is subscribe-only via `apps/socket`.

🚀 Deployment Flow

Default pipeline:

DeepAgent (dev/CI) → GitHub PR → External Deploy (Vercel / Fly / AWS) → Production (Neon Postgres)

Each PR MUST include:

```
Deployment Target: [Vercel|Fly|AppRunner|Fargate|Neon]
Phase: [1|2|3]
Specs: [IDs of affected specs, e.g., 04 Community, 07 Discovery]
```

CI runs on PR:

- Lint / Typecheck / Build
- Web-tier contract guard (fail on boundary violations)
- Unit & integration tests
- Prisma PostGIS migrations on ephemeral DB
- Socket realtime smoke test

🧪 Testing Matrix

Test Type	Tool	Location	Schedule
Unit Tests	Jest/Vitest	apps/api, apps/web, apps/socket	On commit
E2E (web)	Playwright	apps/web	Nightly + on release
Realtime	Vitest + Socket test client	apps/socket	On PR
Migrations	Prisma Migrate	infra/prisma	On deploy
Contract Guard	ESLint + custom CI rule	apps/web	On PR

🔄 CI/CD Pipeline (T8 Implementation)

Overview

The UPRISE monorepo uses GitHub Actions for comprehensive continuous integration and deployment. The CI/CD pipeline ensures code quality, security, and architectural compliance before any code reaches production.

Workflows

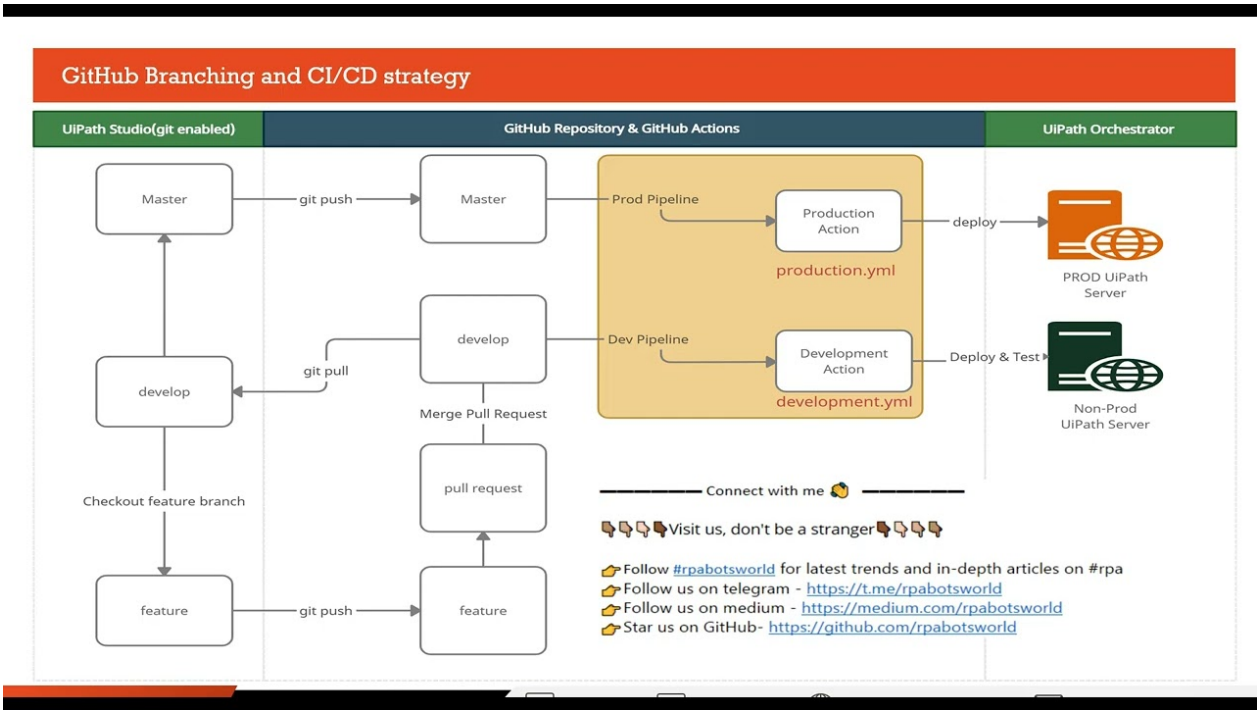
1. Main CI Pipeline (`ci.yml`)

Triggers: Pull requests and pushes to `main` and `develop` branches
Purpose: Comprehensive validation of all code changes

Jobs:
- **Install Dependencies** - Installs and caches pnpm dependencies
- **Lint** - Runs ESLint across all packages
- **Type Check** - Runs TypeScript compiler for type safety
- **Test** - Runs tests for all apps (web, api, socket) with matrix strategy
- **Build** - Builds all apps to ensure production readiness
- **Infrastructure Policy** - Enforces web-tier contract boundaries
- **CI Success** - Final validation that all checks passed

Optimizations:
- Concurrency groups to cancel outdated runs
- Multi-layer caching (pnpm store, node_modules, Turborepo cache)
- Matrix strategy for parallel test execution
- Timeout protection (10-15 minutes per job)
- Artifact uploads for test coverage and build outputs

View Status: [



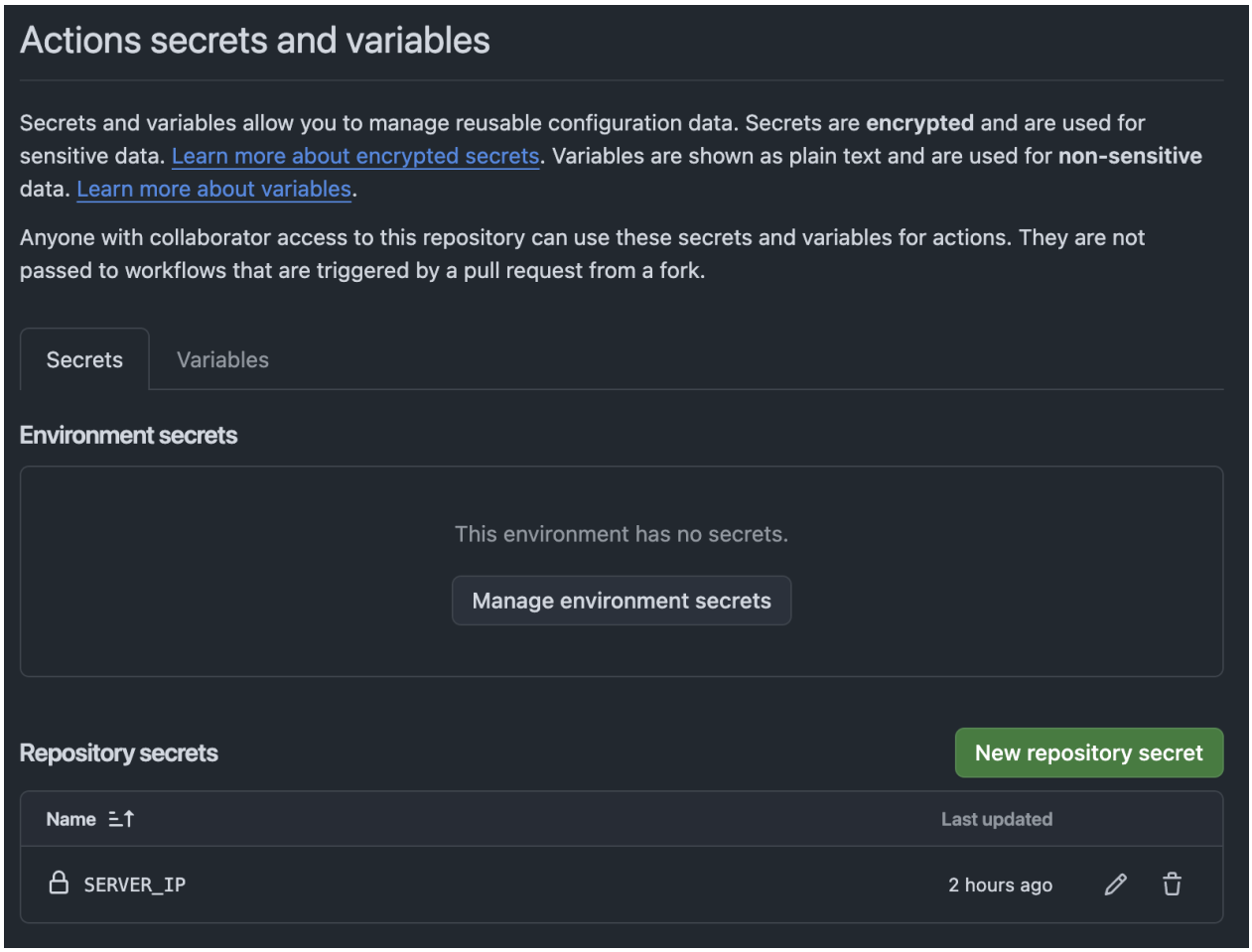
2. Secrets Scanning (`secrets-check.yml`)

Triggers: Pull requests, pushes to `main` / `develop` , weekly schedule (Sundays)
Purpose: Detect accidentally committed secrets and sensitive data

Scanners:
- **Gitleaks** - Industry-standard secret detection
- **TruffleHog** - Advanced pattern matching with verification
- **Custom Patterns** - UPRISE-specific secret detection:
- AWS access keys
- API keys and tokens
- Private keys (PEM, SSH)
- Database URLs with credentials
- JWT secrets (non-example values)
- Stripe live keys
- GitHub tokens
- Slack tokens
- Sentry DSN

Protection:
- Scans full git history (not just diffs)
- Validates `.env.example` files exist and contain no real secrets
- Fails PR if any secrets detected
- Weekly scheduled scans for ongoing monitoring

View Status: [



3. Infrastructure Policy Check (`infra-policy-check.yml`)

Triggers: Changes to `apps/web/**` or policy scripts, manual dispatch
Purpose: Enforce web-tier contract boundaries (T5)

What It Checks:
- No direct database imports in web tier
- No Prisma Client usage in frontend
- No hardcoded secrets in client code
- Proper separation of web and data tiers

Integration: This workflow is also integrated into the main CI pipeline as a job, but can be run independently for quick validation.

View Status: [

Running CI Locally

Before pushing changes, run these commands locally to catch issues early:

```
# Install dependencies
pnpm install

# Run all CI checks locally
pnpm run lint              # ESLint
pnpm run typecheck         # TypeScript
pnpm run test              # All tests
pnpm run build             # Build all apps
pnpm run infra-policy-check # Web-tier boundary check

# Run checks for specific apps
pnpm --filter web test
pnpm --filter api test
pnpm --filter socket test

# Watch mode for development
pnpm --filter web test:watch
```

Debugging CI Failures

Lint Failures

```
# View lint errors
pnpm run lint

# Auto-fix lint issues
pnpm run lint:fix

# Lint specific app
pnpm --filter web lint
```

Type Check Failures

```
# View type errors
pnpm run typecheck

# Type check specific app
pnpm --filter api typecheck
```

Test Failures

```
# Run tests with verbose output
pnpm --filter web test --verbose

# Run specific test file
pnpm --filter api test apps/api/test/communities.test.ts

# Run tests in watch mode
pnpm --filter socket test:watch

# Run tests with coverage
pnpm --filter web test:coverage
```

Build Failures

```
# Build with verbose logging
pnpm run build --verbose

# Build specific app
pnpm --filter web build

# Clear cache and rebuild
pnpm clean
pnpm install
pnpm run build
```

Infrastructure Policy Failures

```
# Run policy check locally
pnpm run infra-policy-check

# Common issues:
# - Importing @prisma/client in apps/web
# - Importing from apps/api/src in apps/web
# - Using database functions in web tier

# Fix: Move database logic to apps/api and use API client
```

Secrets Scan Failures

```
# If secrets are detected:
# 1. Remove the secret from the file
# 2. Add it to .gitignore (if it's in a file like .env)
# 3. Use environment variables instead
# 4. Update .env.example with placeholder values
# 5. Rotate the compromised secret immediately

# Check what's being tracked by git
git ls-files | grep -E "\.env$"

# Remove accidentally committed .env file
git rm --cached .env
echo ".env" >> .gitignore
git add .gitignore
git commit -m "fix: Remove .env from git tracking"
```

CI/CD Best Practices

1. **Always run checks locally before pushing**
   - Saves CI time and catches issues early
   - Use `pnpm run lint && pnpm run typecheck && pnpm run test`

2. **Keep commits small and focused**
   - Easier to debug CI failures
   - Faster to identify the cause of issues

3. **Use conventional commit messages**
   - `feat:` for new features
   - `fix:` for bug fixes
   - `chore:` for maintenance
   - `docs:` for documentation
   - `test:` for test changes
   - `refactor:` for refactoring

4. **Monitor CI status badges**
   - Check README.md for overall project health
   - Red badges indicate failing workflows

5. **Review CI logs for failures**
   - GitHub Actions provides detailed logs
   - Click on failed jobs to see specific errors

6. **Use draft PRs for work-in-progress**
   - CI still runs but won't block merge
   - Useful for getting early feedback

CI Performance Metrics

- **Average CI time:** ~8-12 minutes (full pipeline)
- **Cache hit rate:** ~95% (with pnpm and Turborepo caching)
- **Test execution:** ~2-3 minutes (with matrix parallelization)
- **Build time:** ~3-4 minutes (all apps)

Future Enhancements (Phase 2-3)

- [ ] Automated deployment to staging environments
- [ ] Visual regression testing with Percy or Chromatic
- [ ] Performance testing with Lighthouse CI
- [ ] E2E tests with Playwright
- [ ] Automated dependency updates with Dependabot
- [ ] Code coverage tracking with Codecov
- [ ] Deploy previews for each PR (Vercel/Netlify)

🔒 Web-Tier Contract Guard (T5 Implementation)

Purpose

The Web-Tier Contract Guard enforces strict architectural boundaries to prevent direct database access, server-side imports, and secret leakage in the web tier (`apps/web`). This is a critical infrastructure policy that ensures the UPRISE platform maintains proper separation of concerns.

What It Does

The guard scans all TypeScript/JavaScript files in `apps/web` and detects:

❌ **PROHIBITED Patterns:**
- Direct database imports (`@prisma/client` , `pg` , `mongodb` , `mongoose` , etc.)
- Direct imports from `apps/api/src` or `apps/socket/src`
- Server-side environment variables (`DATABASE_URL` , `JWT_SECRET` , `AWS_SECRET_ACCESS_KEY` , etc.)
- Non- `NEXT_PUBLIC_` environment variables in client components
- AWS SDK imports (`aws-sdk` , `@aws-sdk/*`)
- File system access (`fs` module)
- Server-only Node.js modules (`child_process` , etc.)

✅ **ALLOWED Patterns:**
- API client (`import { api } from "@/lib/api"`)
- Socket.IO client (`import { io } from "socket.io-client"`)
- Shared packages (`@uprise/ui` , `@uprise/types` , etc.)
- `NEXT_PUBLIC_` environment variables
- Client-safe utilities and components

Running Locally

```
# Run the guard
pnpm run infra-policy-check

# Show help and documentation
pnpm run infra-policy-check --help

# Verbose output with file counts
pnpm run infra-policy-check --verbose
```

CI Integration

The guard runs automatically on every PR and push to `main` or `develop` branches via GitHub Actions. If violations are detected, the build will fail and must be fixed before merging.

Workflow file: `.github/workflows/infra-policy-check.yml`

Error Codes

Each violation includes a specific error code for easy identification:

Code	Description
`WEB_TIER_DB_001` - `WEB_TIER_DB_007`	Database access violations
`WEB_TIER_IMPORT_001` - `WEB_TIER_IMPORT_004`	Server-side import violations
`WEB_TIER_SECRET_001` - `WEB_TIER_SECRET_006`	Environment variable/secret violations
`WEB_TIER_AWS_001` - `WEB_TIER_AWS_002`	AWS SDK violations
`WEB_TIER_FS_001` - `WEB_TIER_FS_002`	File system access violations
`WEB_TIER_SERVER_001` - `WEB_TIER_SERVER_002`	Server-only module violations

Example Violation Output

```
❌ Web-Tier Contract Violations Detected (ERRORS):
═══════════════════════════════════════════════════════════════

1. apps/web/src/lib/db.ts:5:1
   Code: WEB_TIER_DB_001
   Message: Direct Prisma Client import is prohibited in web tier. Use API client instead.
   Snippet: import { PrismaClient } from '@prisma/client';

2. apps/web/src/components/user-profile.tsx:10:15
   Code: WEB_TIER_SECRET_001
   Message: DATABASE_URL must not be accessed in web tier. This is a server-side secret.
   Snippet: const db = process.env.DATABASE_URL;

═══════════════════════════════════════════════════════════════

❌ Total Errors: 2
```

How to Fix Violations

Instead of direct database access:

```
// ❌ WRONG
import { PrismaClient } from '@prisma/client';
const prisma = new PrismaClient();
const users = await prisma.user.findMany();

// ✅ CORRECT
import { api } from '@/lib/api';
const users = await api.get('/users');
```

Instead of server-side secrets:

```
// ❌ WRONG
const secret = process.env.JWT_SECRET;

// ✅ CORRECT
const publicKey = process.env.NEXT_PUBLIC_API_KEY;
```

Instead of direct API imports:

```
// ❌ WRONG
import { UserService } from '../../api/src/users/user.service';

// ✅ CORRECT
import { api } from '@/lib/api';
// Or use shared types from @uprise/types
```

Related Documentation

- `STRATEGY_CRITICAL_INFRA_NOTE.md` (./STRATEGY_CRITICAL_INFRA_NOTE.md) - Infrastructure policy
- `PROJECT_STRUCTURE.md` (./PROJECT_STRUCTURE.md) - Architectural boundaries
- `apps/web/WEB_TIER_BOUNDARY.md` (../apps/web/WEB_TIER_BOUNDARY.md) - Web-tier contract details

Script Location

- **Source:** `scripts/infra-policy-check.ts`
- **Legacy (deprecated):** `scripts/infra_check_web.js`

📚 Documentation Index

Category	File	Description
Strategy	STRATEGY_CRITICAL_INFRA_NOTE.md (./STRATEGY_CRITICAL_INFRA_NOTE.md)	Infrastructure policy
Milestones	PHASE1_COMPLETION_REPORT.md (./PHASE1_COMPLETION_REPORT.md)	Phase 1 completion
Specs	Specifications/README.md (./Specifications/README.md)	Module-by-module technical docs
Environments	ENVIRONMENTS.md (./ENVIRONMENTS.md)	Windows/WSL setup rules
Structure	PROJECT_STRUCTURE.md (./PROJECT_STRUCTURE.md)	Folder map & conventions
Changelog	CHANGELOG.md (./CHANGELOG.md)	Auto-generated PR logs

---

🧩 Agent Rules

1. **Follow the Critical Infra Note** — DeepAgent may run tests, not production workloads.
2. **Keep Docs Current** — Every merged PR must update `CHANGELOG.md` and, if scope touches architecture or ops, update this `RUNBOOK.md`.
3. **Annotate PRs** — Link to affected specification(s) in `/docs/Specifications`.
4. **Blockers** — Any CI error tagged `infra-policy-check` halts merge until fixed.

🔄 Maintenance Schedule

Interval	Task	Owner
Daily	CI: lint/type/build pass	DeepAgent
Nightly	E2E (web) + socket smoke	DeepAgent
Weekly	Update CHANGELOG from merged PRs	DeepAgent
Per Phase	Publish phase completion report	PM/Lead Agent

🏁 Quickstart Commands

```
pnpm i
pnpm -r build
pnpm -r dev
```

See ENVIRONMENTS.md for full setup and service URLs.