Deployment Risk Analysis: Configuration Inconsistencies

1. Summary of Findings

A thorough review of the repository has identified significant inconsistencies and contradictions in the configuration management for the Aegis Risk Management Platform. These issues present a high risk of deployment failures, create confusion for developers, and will impede debugging and maintenance.

The core problem is the lack of a single source of truth for network configuration, specifically the ports used by the frontend and backend services. We have identified at least three conflicting setup methods documented and implemented in the codebase.

2. Documented Configuration Conflicts

Method 1: Random Ports (from `CLAUDE.md`)

- **Source:** `CLAUDE.md`
- **Instruction:** Use random, high-numbered ports for both frontend and backend development servers to avoid conflicts.
- **Contradiction:** While this is a robust strategy, the documentation is incomplete. It mentions `export VITE_API_URL="http://localhost:$BACKEND_PORT"` but this is not integrated into a clear, step-by-step workflow, leaving it up to the developer to implement correctly.

Method 2: Hardcoded Production Ports in Dev (from `DEVELOPMENT_SPEC.md`)

- **Source:** `DEVELOPMENT_SPEC.md` (Section 3.3 Frontend .env)
- **Instruction:** The example frontend `.env` file specifies `VITE_API_URL=http://localhost:8000/api/v1`.
- **Contradiction:** This directly contradicts `CLAUDE.md`, which explicitly states: "**Never use ports 3000 and 8000 in development** - these are reserved for production."

Method 3: Docker Compose & `.env.local` (The *de facto* setup)

- **Source:** `docker/docker-compose.yml` and `frontend/aegis-frontend/.env.local`
- **Instruction:** The Docker setup exposes the backend on host port `30641` and the frontend on `58533`. The frontend's local environment file (`.env.local`) is configured to connect to port `30641`.

- **Contradiction:** This appears to be the actual working configuration for a containerized environment, but it is not clearly documented as the primary development method. The main markdown files (`CLAUDE.md`, `DEVELOPMENT_SPEC.md`) describe a non-containerized development workflow that is incompatible with this setup.

3. Risks to Production Deployment

These inconsistencies create the following risks:

- **Developer Environment Failure:** A new developer following the instructions in `CLAUDE.md` or `DEVELOPMENT_SPEC.md` will have a non-functional local development environment, as the frontend will not be able to connect to the backend.
- **CI/CD Pipeline Failures:** Continuous Integration pipelines that run tests (e.g., Playwright E2E tests) may be configured using one of the incorrect methods, leading to persistent test failures and blocked deployments.
- **Incorrect Builds:** If a build script for the frontend accidentally uses the wrong environment configuration (e.g., the one from `DEVELOPMENT_SPEC.md` with port 8000), the resulting static assets will be broken and will not work in a production environment that uses a different port mapping.
- **Debugging Complexity:** When production issues arise, the lack of clear and consistent configuration makes it difficult to replicate the production environment locally, significantly increasing the time required to diagnose and fix bugs.
- **Configuration Drift:** The presence of multiple conflicting configurations makes it likely that future changes will exacerbate the problem, leading to further "configuration drift" where documentation and reality are completely disconnected.

4. Proposed Unified Configuration Strategy (Recommendation)

To mitigate these risks, we strongly recommend adopting a **single, authoritative source of truth** for all environment configurations.

1. **Embrace Docker for Local Development:** Standardize on the `docker-compose.yml` setup as the primary method for local development. This ensures that developers are working in an environment that is as close to production as possible.
2. **Simplify and Unify `.env` files:**
    - Create a single `.env.example` file in the repository root that contains all necessary environment variables for both frontend and backend.
    - This file should have placeholders (e.g., `VITE_API_URL=http://localhost:30641/api/v1`).
    - Developers can copy this to a `.env` file (which is git-ignored) to run the `docker-compose` setup.
3. **Update All Documentation:**

- Rewrite the development setup sections of `CLAUDE.md` and `DEVELOPMENT_SPEC.md` to reflect the Docker-based workflow.
- Remove all references to random ports and the incorrect hardcoded port 8000 for development.
- Provide clear, step-by-step instructions: `git clone ...`, `cp .env.example .env`, `docker-compose up`.

4. **Align CI/CD:** Ensure that the CI/CD pipeline uses the same `docker-compose.yml` and environment variable configuration for running tests and building production artifacts.

By enforcing this consistency, you will improve developer onboarding, increase deployment reliability, and streamline the entire development and operations lifecycle.