Detailed Master Technical Documentation: Phase 4 – Integrated EMsync Hub

Project: EMsync (Emergency Coordination & Management System)

Architecture: Cloud-Native Microservices

Environment: Docker / GitHub Actions / Docker Hub

Date: January 5, 2026

---

1. Architectural Evolution

The project has evolved from **Phase 3 (Disconnected Modules)** to **Phase 4 (Orchestrated Infrastructure)**. Instead of manually starting three different Python scripts, the entire backend is now managed as a single unit using **Docker Compose**.

Key Architecture Shifts:

- **Isolation**: Each module (ML, Routing, Transfer) now has its own dedicated OS environment (Alpine/Ubuntu-based containers).
- **Decoupling**: The "Build" process is separated from the "Run" process to protect local disk space.
- **Networking**: Services share a private virtual bridge (emsync-network), allowing them to communicate without exposing all ports to the public internet.

---

2. Deep Dive: Service Configurations

Service	Port	Dependency	Optimization
Phase 1: Severity Sim	8501	xgboost, scikit-learn	**Layered Cache**: Heavy ML libraries are cached in the cloud to speed up deployment.
Phase 2: Routing API	6005	flask-cors, openrouteservice	**API Gateway**: Configured to handle CORS requests from the upcoming React

			frontend.
Phase 3: Transfer Hub	5000	flask-socketio, pycryptodome	**Security Bypass**: Implemented allow_unsafe_werkzeug=True for containerized SocketIO stability.

---

3. Detailed Implementation Steps (The "How-To")

Step 1: Container Hardening & Port Bridging

We modified the Python source code for each backend to ensure compatibility with Docker's networking.

- **Fix**: Replaced app.run() and socketio.run() parameters.
- **Code Change**: Added host="0.0.0.0". Without this, the container listens only to itself. With this, it listens to the Docker bridge, allowing your Windows browser to see the app.

Step 2: The Storage Mitigation Strategy (Anti-Rubbish Protocol)

The primary goal was to prevent the G: drive from filling up with Docker build artifacts.

- **Action**: Created a .dockerignore file.
- **Logic**: This tells Docker to ignore __pycache__, .venv, and .git folders during the build. This reduces image size and prevents local files from corrupting the clean container environment.

Step 3: CI/CD Pipeline Construction (GitHub Actions)

We automated the build process using a "Cloud Runner."

- **Workflow**: .github/workflows/cloud-build.yml.
- **Automation**: Every time you git push, GitHub spawns a virtual Ubuntu server, installs Docker, builds your images, and pushes them to **Docker Hub**.
- **Secrets**: Used DOCKERHUB_TOKEN for secure, password-less authentication between GitHub and Docker.

Step 4: The "Thin Client" Compose Setup

Your local docker-compose.yml was stripped of build: commands.

- **Logic**: By specifying image: alanjosephpv/emsync-xxxx, you tell Docker to **pull** rather than **make**. This is the secret to keeping your PC storage clean.

---

4. Operational Procedures & Maintenance

The "Zero-Trash" Daily Workflow:

1. **Code & Push**: Edit code $\rightarrow$ git push origin main.
2. **Cloud Build**: Monitor the **Actions** tab on GitHub until the checkmark is green.
3. **Sync Local**: docker compose pull.
4. **Launch**: docker compose up -d.

Storage Recovery (Manual):

If the Windows VHDX (ext4.vhdx) grows too large:

1. Run docker system prune -af --volumes.
2. If still large, manually delete the %LOCALAPPDATA%/Docker/wsl folder and restart Docker (Factory Reset).

---

5. Current Progress Tracking

Task	Status	Note
Microservice Containerization	**100%**	All 3 phases packaged.
Cross-Service Networking	**100%**	Internal bridge active.
CI/CD Automation	**100%**	GitHub Actions + Docker Hub linked.
Local Storage Optimization	**100%**	Pull-only workflow established.
Unified UI Integration	*0%*	**NEXT STEP: React Shell App.**

> **Phase 4 Summary**: You now have a professional-grade, automated backend infrastructure. The system is scalable, storage-efficient, and ready to be connected to a final user interface.

Would you like me to start providing the code for the React Shell App to unify these

modules into one dashboard?