

Repo: <https://github.com/DevArtech/cartographer>

What Was Accomplished / Goal Met

The Goal: Consolidate multiple network monitoring tools (like Grafana and its dependencies) into a single, self-hosted application.

What Was Built: Cartographer delivers all the intended features:

Feature	Status
Automatic Network Mapping	<input checked="" type="checkbox"/> ARP scanning + ICMP ping
History Management	<input checked="" type="checkbox"/> JSON persistence
Device Uptime Recording	<input checked="" type="checkbox"/> Health service tracks device states
LAN Mapping	<input checked="" type="checkbox"/> Interactive D3.js visualization
User Sharing + Management	<input checked="" type="checkbox"/> JWT auth with roles (owner/admin/editor/viewer)
Embeddable Maps	<input checked="" type="checkbox"/> Public shareable embed IDs
AI Assistant	<input checked="" type="checkbox"/> Multi-provider (OpenAI, Anthropic, Gemini, Ollama)
Email + Discord Notifications	<input checked="" type="checkbox"/> Resend + Discord Bot integration
ML Anomaly Detection	<input checked="" type="checkbox"/> scikit-learn isolation forest
Metrics Publishing	<input checked="" type="checkbox"/> Redis Pub/Sub + WebSocket streaming

The microservices architecture was fully realized with 5 services behind an API gateway:

- **Backend** (8000) - Gateway + Vue.js SPA
- **Health** (8001) - Device monitoring
- **Auth** (8002) - JWT authentication
- **Metrics** (8003) - Real-time topology
- **Assistant** (8004) - AI chat
- **Notification** (8005) - Alerts + ML

Post-Mortem Summary

What Went Well - Technology Stack

Layer	Technologies
Frontend	Vue.js 3, TypeScript, Vite, Pinia, D3.js, TailwindCSS
API Gateway	FastAPI, HTTPX, WebSocket Proxy
Microservices	FastAPI (Python), async/await patterns
Event Streaming	Redis Pub/Sub

Layer	Technologies
Auth	JWT, bcrypt
ML/AI	OpenAI, Anthropic, Gemini, scikit-learn
Containers	Docker, Docker Compose
Testing	Pytest, Locust, Storybook
CI/CD	GitHub Actions (7 workflow files)

Future Improvements Identified

Design Decisions:

1. Implement Circuit Breakers (partially done via `http_client.py`)
2. Centralized Logging (ELK/Loki stack)
3. Kubernetes Deployment
4. Earlier Unit + E2E Tests

App Improvements:

1. Custom Alert Rules
2. Mobile App
3. Kubernetes Helm Chart
4. Plugin System

How to Deploy, Test, and Monitor

Local Deployment

```
# Option 1: Helper script
./deploy.sh up

# Then visit http://localhost:8000
```

Running Load Tests

```
cd load-tests
pip install -r requirements.txt

# Headless baseline test (10 users, 60 seconds)
python run_load_tests.py -s all -u 10 -r 2 -t 60 --username YOUR_USERNAME --
password YOUR_PASSWORD
```

Endpoint Status Endpoints

Endpoint	Description
GET /healthz	Internal health check - shows circuit breaker states for all services
GET /api/health/monitoring/status	Device health monitoring status
GET /api/metrics/redis/status	Redis connection status
GET /api/metrics/usage/stats	Aggregated endpoint usage statistics
GET /api/notifications/status	Notification service status + available channels
GET /api/notifications/ml/status	ML anomaly detection status
GET /api/assistant/context/status	AI assistant context status

The `/healthz` endpoint is particularly useful as it returns circuit breaker states:

```
{
  "status": "healthy",
  "services": {
    "health": {"circuit_state": "closed", "failure_count": 0},
    "auth": {"circuit_state": "closed", "failure_count": 0},
    "metrics": {"circuit_state": "closed", "failure_count": 0},
    "assistant": {"circuit_state": "closed", "failure_count": 0},
    "notification": {"circuit_state": "closed", "failure_count": 0}
  }
}
```

Feedback on the Assignment

The freeform nature of this assignment was genuinely enjoyable! It allowed for creative freedom while still requiring rigorous engineering. Building Cartographer allowed for me to create a **real, production-grade tool** that:

- **I'll actually use** to monitor my home server infrastructure
- **Demonstrates modern microservices patterns** (API gateway, event streaming, circuit breakers)
- **Showcases full-stack skills** (Vue.js frontend, FastAPI backend, Redis, Docker)
- **Includes enterprise concerns** (CI/CD, load testing, ML integration, multi-provider AI)

The ability to submit something personally meaningful—rather than a contrived demo project—made the work feel purposeful. It's rare to get course credit for building your own infrastructure tooling!