NetPulse – UpGuardX

Smart Uptime Monitoring & Security Visibility Platform

Complete Project Documentation

README • Architecture Blueprint • Developer Guide

Version 1.0.0 — Planning Stage | License: MIT | Open Source

📋 PART 1: Project Overview (README)

About NetPulse UpGuardX

NetPulse UpGuardX is an open-source, self-hostable uptime monitoring and security visibility platform. Monitor websites, servers, APIs, ports, DNS, and SSL certificates in real time — with smart alerting, role-based access, public status pages, and deep infrastructure insights.

Designed for teams who want full data ownership, zero vendor lock-in, and enterprise-grade observability without enterprise pricing.

Feature Overview

Category	Key Capabilities
🟢 Uptime Monitoring	HTTP/HTTPS, TCP, ICMP, DNS, SSL, custom intervals (10s–1h)
🔐 Security Monitoring	Brute-force detection, port change alerts, WAF integration, SSL misconfig
🖥️ Infrastructure	CPU, RAM, Disk, Network, Docker containers, VM health
📋 Log Monitoring	Centralized logs, error detection, failed login alert, SIEM-ready
🔔 Smart Alerts	Email, Telegram, Slack, Discord, SMS (Twilio), Webhooks
🌐 Status Page	Public page, custom domain, branding, incident timeline
👥 RBAC	Super Admin, Admin, Viewer, API keys, organization-based access
🏢 Multi-Organization	Per-company dashboards, MSP mode, client separation
📊 Reporting & Analytics	PDF/CSV reports, uptime %, SLA tracking, downtime root cause

Quick Start

Docker (Recommended)

```
git clone https://github.com/your-org/netpulse-upguardx.git
cd netpulse-upguardx
cp .env.example .env
docker-compose up -d
```

Then visit http://localhost:3000 and log in with the default admin credentials.

Standalone Linux

```
curl -sSL https://install.netpulse.io | bash
```

Tech Stack

Layer	Technology
Frontend	Next.js 14, React, TailwindCSS, TypeScript
Backend API	Node.js 20, Fastify, TypeScript, Prisma ORM
Worker / Agent	Go 1.22 (high-performance concurrent probing)
Database	PostgreSQL 15 (primary), Redis (cache + queues)
Queue	BullMQ + Redis
Authentication	JWT + 2FA (TOTP), OAuth2 optional
Notifications	Nodemailer, Twilio, custom webhooks
Deployment	Docker, Kubernetes, standalone Linux binary

Security Features

- 2FA Login (TOTP — Google Authenticator compatible)
- IP whitelist login per user
- Audit logs for all state-changing actions
- API rate limiting via Redis sliding window
- AES-256-GCM encrypted credentials storage
- Secure agent communication over TLS 1.3
- Passwords hashed with bcrypt (cost 12)
- JWT with short-lived access tokens (15 min) + secure refresh tokens

Deployment Options

- Docker Compose (recommended for small/medium deployments)
- Kubernetes with Helm charts (production scale)
- Standalone Linux installation (single server)
- Agent-based architecture for remote server monitoring
- REST API for full programmatic integration

🏗️ PART 2: Architecture Blueprint

1. Vision & Design Principles

Principle	Description
Privacy First	Self-hostable, no telemetry by default, all data stays on your infrastructure
Modular Architecture	Each service can be scaled or replaced independently
Developer Friendly	REST API, webhooks, open schemas, well-documented extension points
Security by Design	Encrypted secrets, audit logs, TLS everywhere, rate limiting
Minimal Ops Overhead	Docker-first, one-command deploy, sensible defaults

2. System Architecture

Service Topology

NetPulse UpGuardX is composed of five core subsystems working in concert:

- Web Frontend — Next.js dashboard served to end users
- Core API — Node.js/TypeScript service handling all business logic, RBAC, and data access
- Worker Service — Go-based high-concurrency monitoring engine
- Agent — Lightweight Go binary deployed on monitored servers
- Data Layer — PostgreSQL for persistence, Redis for queuing and caching

Request Flow

User Browser → CDN/Nginx → Next.js Frontend OR API Gateway → Auth Middleware → RBAC Check → Core API → PostgreSQL / Redis Queue → Worker Service → Probes → Alert Dispatcher → Notification Channels

3. Core Services

Service	Language	Responsibility
apps/web	TypeScript / Next.js	Dashboard UI, status page builder, report viewer
apps/api	TypeScript / Fastify	CRUD operations, auth, webhook delivery, PDF reports
apps/worker	Go	Scheduled probes, incident detection, alert triggering

apps/agent	Go	Host metrics collection, log tailing, API reporting

4. Monitor Types

Type	What It Checks	Key Config Options
HTTP/HTTPS	Status code, response time, body match	Method, headers, expected status, body pattern
TCP	Port open/closed, connect time	Host, port, timeout
ICMP (Ping)	Round-trip latency, packet loss	Host, packet count, timeout
DNS	Record resolution, value match	Record type (A/MX/CNAME), expected value
SSL	Certificate expiry, chain validity, misconfig	Alert days before expiry, chain check

5. Database Schema Overview

Primary tables in PostgreSQL (all include org_id for multi-tenancy isolation):

Table	Purpose
organizations	Company/team accounts with branding and plan info
users	User accounts with role, 2FA, and IP whitelist
monitors	Monitor definitions with type, target, interval, config
monitor_checks	Time-series probe results (status, response time, error)
incidents	Downtime incidents with timeline and root cause
alert_channels	Notification channel configs (encrypted credentials)
alert_rules	Rules linking monitors to channels with cooldown policy
agents	Registered server agents with API key hashes
agent_metrics	Time-series server metrics (CPU, RAM, Disk, Docker)
audit_logs	Immutable log of all admin actions
api_keys	Scoped API keys for programmatic access

6. API Design

Base URL & Auth

```
Base URL: https://api.yourdomain.com/v1
Auth: Authorization: Bearer <jwt>  OR  X-API-Key: <key>
```

Core Endpoint Groups

Resource	Endpoints
Auth	POST /auth/register, login, logout, refresh, 2fa/setup, 2fa/verify
Monitors	GET/POST /monitors, GET/PUT/DELETE /monitors/:id, pause, resume, checks, uptime
Incidents	GET/POST /incidents, GET/PATCH /incidents/:id, updates
Alert Channels	GET/POST /alert-channels, PUT/DELETE /:id, test
Alert Rules	GET/POST /alert-rules, PUT/DELETE /:id
Agents	GET/POST /agents, DELETE /:id, rotate-key
Organizations	GET/PUT /org, members CRUD, invite
Reporting	GET /reports/uptime, sla, export (PDF/CSV)
Status Pages	GET/POST /status-pages, PUT/DELETE /:id
Public (no auth)	GET /public/status/:slug, /public/status/:slug/incidents
API Keys	GET/POST /api-keys, DELETE /:id
Audit	GET /audit-logs

7. Alerting Pipeline

When a monitor changes status (UP → DOWN or DOWN → UP), the following pipeline executes:

- Incident Manager creates or resolves an incident in PostgreSQL
- Alert Rule Evaluator finds matching rules for the affected monitor
- Cooldown check via Redis TTL — suppresses duplicate alerts during cooldown period
- Alert Dispatcher enqueues delivery jobs for each matched channel
- Channel-specific dispatchers send to Email, Slack, Telegram, Discord, SMS, or Webhook
- Delivery Tracker logs status; failed deliveries retry 3 times with exponential backoff

8. Security Architecture

Layer	Implementation
Password Storage	bcrypt with cost factor 12

Sessions	JWT access tokens (15 min) + httpOnly refresh tokens (7 days)
2FA	TOTP (RFC 6238) — Google Authenticator / Authy compatible
Credential Encryption	AES-256-GCM, key from environment variable only
Agent Auth	Pre-shared API key (bcrypt hash stored), TLS 1.3 transport
Rate Limiting	Redis sliding window per IP and per API key
SQL Injection	Prisma parameterized queries — no raw SQL from user input
Audit Trail	Immutable logs: who, what, when, where, payload snapshot

9. Scalability Targets

Metric	v1 Target
API response time (p99)	< 200ms
Check scheduling lag	< 5 seconds
Alert delivery time	< 30 seconds from incident
Dashboard load time	< 2 seconds
Concurrent monitors (single node)	10,000+

10. Product Roadmap

Version	Status	Key Features
v1.0 — Foundation	Planning	Core monitoring, alerting, status page, RBAC, Docker deploy, REST API
v1.5 — Infrastructure	Future	Server agent, log monitoring, PDF reports, SMS, multi-org, 2FA, K8s
v2.0 — Security & Intelligence	Future	ML anomaly detection, brute-force alerting, WAF/SIEM integration, MSP mode
v3.0 — Enterprise	Future	SSO/SAML, global monitoring regions, white-label, advanced scripting

👩‍💻 PART 3: Developer Documentation

1. Prerequisites

Tool	Version	Purpose
Node.js	20 LTS	API + Web frontend
Go	1.22+	Worker + Agent
Docker	24+	Local infrastructure services
Docker Compose	v2	Orchestrate local services
pnpm	8+	Monorepo package manager

2. Getting Started

Clone and install

```
git clone https://github.com/your-org/netpulse-upguardx.git
cd netpulse-upguardx
pnpm install
cd apps/worker && go mod download
cd apps/agent && go mod download
```

Start infrastructure

```
docker-compose -f infrastructure/docker/docker-compose.dev.yml up -d
```

Configure environment

```
cp .env.example .env   # then edit with your values
```

Run migrations and seed

```
cd apps/api && pnpm prisma migrate dev && pnpm prisma db seed
```

Start all services

```
pnpm dev   # starts web + api in parallel via Turborepo
```

3. Project Structure

Path	Description
apps/web/	Next.js 14 frontend — dashboard, status pages, reports
apps/api/	Node.js TypeScript API — routes, services, Prisma ORM
apps/worker/	Go monitor worker — scheduler, probes, evaluator, alerter

apps/agent/	Go host agent — metric collectors, log watcher, reporter
packages/ui/	Shared React UI components (TailwindCSS)
packages/types/	Shared TypeScript types and Zod schemas
infrastructure/	Docker, Kubernetes, Caddy configs
docs/	All documentation files

4. Key Environment Variables

API Service

Variable	Description
DATABASE_URL	PostgreSQL connection string
REDIS_URL	Redis connection string
JWT_SECRET	Secret for signing access tokens (min 32 chars)
ENCRYPTION_KEY	32-byte hex key for AES-256 credential encryption
SMTP_HOST / PORT / USER / PASS	SMTP server for email alerts
TWILIO_ACCOUNT_SID / AUTH_TOKEN	Twilio credentials for SMS alerts
FRONTEND_URL	CORS allowed origin (your frontend URL)

5. Coding Standards

- TypeScript strict mode enabled — no implicit any
- Zod schemas for all API input validation — derive TypeScript types from schemas
- Conventional Commits for all commit messages (feat:, fix:, docs:, chore:)
- ESLint + Prettier enforced in CI — run 'pnpm lint' before committing
- Go: gofmt + golangci-lint, always propagate context.Context, wrap errors with fmt.Errorf
- Test coverage target: 70% minimum for core services

6. Extension Points

Adding a New Monitor Type

1. Implement the Probe interface in apps/worker/internal/probes/
2. Register in the probe registry (probes/registry.go)
3. Add Zod validation to the API route schema
4. Add form fields in the frontend MonitorForm component

Adding a New Alert Channel

1. Create a dispatcher function in apps/api/src/services/alerts/dispatchers/

2. Register in the dispatcher factory
3. Add a Zod config schema in packages/types/
4. Add a setup form in the frontend settings UI

7. Testing

Type	Tool	Command
Unit tests (API)	Vitest	cd apps/api && pnpm test
Integration tests (API)	Supertest + Vitest	pnpm test:integration
Unit tests (Worker)	Go test	cd apps/worker && go test ./...
End-to-end tests	Playwright	cd apps/web && pnpm test:e2e
Coverage report	Vitest / Go cover	pnpm test:coverage

8. Git Workflow

Branch	Purpose
main	Production-ready code — protected, requires PR
develop	Integration branch — all features merge here first
feature/*	New features (branch from develop)
fix/*	Bug fixes (branch from develop)
hotfix/*	Urgent production fixes (branch from main)
release/v*	Release preparation branches

9. Getting Help

- GitHub Discussions — questions, ideas, design proposals
- GitHub Issues — bug reports and feature requests
- Discord — community chat (link in README)

When reporting a bug, include: OS, Node/Go version, steps to reproduce, expected vs actual behavior, and sanitized logs.