Sonde — Design Document

Sonde Design Document

AI-Powered Infrastructure Diagnostics Agent System

Version 1.0 — February 2026

TL;DR

Sonde is a **hub-and-spoke agent system** that gives AI assistants (Claude, ChatGPT, Gemini, or any MCP/REST-compatible model) **secure, read-only access** to infrastructure for troubleshooting and diagnostics.

Today, an engineer troubleshooting a server with AI spends 15+ minutes copying commands, pasting into terminals, copying output back, and repeating. Sonde eliminates that friction entirely. The engineer describes the problem in natural language, and the AI gathers all diagnostic data itself through Sonde in seconds.

Key properties:

- ⚡ **Fast** — Full diagnostic battery in one request (~30 seconds vs ~15 minutes manual)
- 🔒 **Secure** — 9-layer defense-in-depth model, read-only by default, no raw shell access
- 🌐 **Model-agnostic** — Works with any AI via MCP protocol or REST API
- 📦 **Extensible** — Plugin "packs" for Docker, systemd, Postgres, nginx, Splunk, and more
- 🏢 **Enterprise-ready** — mTLS, OAuth 2.0, per-key policy scoping, tamper-evident audit logs

---

Problem Statement

The Copy-Paste Loop

Engineers increasingly use AI assistants for infrastructure troubleshooting. The current workflow is:

1. Describe problem to AI
2. AI suggests a diagnostic command
3. Engineer copies command, pastes into terminal
4. Engineer copies output, pastes back to AI
5. AI analyzes, suggests another command
6. Repeat 5–15 times

This loop is **slow** (10–20 minutes per issue), **error-prone** (truncated output, wrong terminal, stale data), and **unscalable** (one server at a time, no fleet-wide view).

The Gap

AI assistants have no direct visibility into infrastructure. They are limited to what the user manually feeds them. There is no secure, standardized way for an AI to query a server’s state.

What Sonde Solves

Sonde bridges this gap by providing a **secure, structured, read-only** channel between AI assistants and infrastructure. The AI describes what it needs to know, and Sonde handles the rest — executing diagnostics across one or many servers simultaneously and returning consolidated, structured results.

Solution Overview

How It Works

```
Engineer: "Docker is broken on server-01"
     │
     ▼
```

```
AI Assistant (Claude, GPT, Gemini, etc.)
     │  calls diagnose tool via MCP or REST
     ▼
Sonde Hub (central server)
     │  dispatches probes to target agent
     ▼
Sonde Agent (on server-01)
     │  runs read-only diagnostics locally:
     │  - Docker daemon status
     │  - Container list + states
     │  - Recent Docker logs
     │  - Disk usage
     │  - Memory usage
     ▼
Results flow back through hub to AI
     │
     ▼
AI: "Your disk is 98% full on /var/lib/docker.
     Run: docker system prune -a
     This will free approximately 45GB."
```

Total time: **~30 seconds**, zero copy-paste.

Core Components

Sonde Hub — The central coordination server. Receives requests from AI clients, dispatches probe commands to agents, assembles results, enforces policies, and maintains audit logs. Deployed as a single Docker container.

Sonde Agents — Lightweight daemons installed on target machines. Connect outbound to the hub via WebSocket (no inbound ports required). Execute structured diagnostic probes and return sanitized results. Run as an unprivileged system user.

Packs — Modular capability plugins that define what an agent can inspect. Each pack covers a specific technology (Docker, Postgres, nginx, etc.) and declares exactly what system access it requires. Packs are versioned, signed, and independently installable.

Runbooks — Pre-defined diagnostic workflows that orchestrate multiple probes for a given problem category. When an AI says "diagnose Docker," the runbook fires a battery of relevant probes in parallel and returns a consolidated report.

Architecture

Hub-and-Spoke Topology

```
                  ┌──────────────────────────────┐
                  │        Sonde Hub             │
                  │                              │
AI Clients ───▶  │  MCP Endpoint  (/mcp)        │
(Claude,          │  REST API     (/api/v1)      │
 GPT,             │  Dashboard    (/dashboard)   │
 Gemini,          │  Agent WS     (/ws/agent)    │
 Custom)          │                              │
                  └──────┬───────┬───────┬───────┘
                         │       │       │
                    WSS  │  WSS  │  WSS  │  (outbound)
                         │       │       │
                    ┌────┴┐  ┌───┴─┐  ┌──┴──┐
                    │Agent│  │Agent│  │Agent│
                    │  01 │  │  02 │  │  03 │
                    └─────┘  └─────┘  └─────┘
```

Critical design decision: Agents initiate all connections outbound to the hub. Agents never listen on a port. This makes deployment NAT-friendly, firewall-friendly, and eliminates an entire class of attack surface.

Communication Flow

1. Agent starts and connects outbound to hub via persistent WebSocket (WSS)

2. AI client sends tool call to hub via MCP protocol or REST API
3. Hub authenticates request, evaluates policies, selects target agent
4. Hub sends structured probe descriptor to agent over existing WebSocket
5. Agent executes mapped read-only command locally as unprivileged user
6. Agent scrubs sensitive data from output (keys, passwords, tokens)
7. Agent signs response payload and returns to hub
8. Hub assembles results and returns to AI client

Hub Interfaces

The hub exposes four interfaces from a single process:

Interface	Endpoint	Protocol	Purpose
MCP	`/mcp`	StreamableHTTP	AI assistants via MCP protocol
REST API	`/api/v1/*`	HTTPS	Custom integrations, bots, scripts
Agent WebSocket	`/ws/agent`	WSS	Agent connections
Dashboard	`/dashboard`	HTTPS	Web UI for human management

AI Client Connectivity

Sonde is **model-agnostic**. Any AI platform can connect:

Client	Connection Method	Auth
Claude.ai	MCP (StreamableHTTP)	OAuth 2.0
Claude Code	MCP (stdio bridge)	API key
ChatGPT / GPT API	REST API or MCP (when supported)	API key
Gemini / Google AI	REST API or MCP (when supported)	API key
Open-source models	REST API	API key
Custom applications	REST API	API key
n8n / automation	REST API	API key

The OAuth 2.0 flow exists specifically because Claude.ai requires it for remote MCP connections. All other clients use API keys. Both auth methods provide the same access — OAuth is not a gating mechanism for functionality.

MCP Tool Design

Sonde exposes high-level, intent-driven tools to AI assistants — not low-level command wrappers. The AI doesn't need to know system commands; it describes what it wants to understand.

Tool	Purpose	Example Input
`diagnose`	Run a full diagnostic battery for a problem category	`{ agent: "server-01", category: "docker" }`
`probe`	Run a single targeted probe	`{ agent: "server-01", probe: "docker.containers.list" }`
`list_agents`	Show all connected agents and their capabilities	`{}`
`agent_overview`	Detailed health snapshot of one agent	`{ agent: "server-01" }`
`query_logs`	Search logs across agents or external platforms	`{ query: "error", source: "docker" }`

The `diagnose` tool is the most powerful — a single call fires a runbook of parallel probes and returns a consolidated report. This is how a 15-minute troubleshooting session becomes 30 seconds.

Pack System

Packs are modular, versioned plugins that define an agent's capabilities. Each pack covers a specific technology and contains probe definitions, a permission manifest, and optional runbook definitions.

Available Packs

Pack	Probes	Required Access
System	disk.usage, memory.usage, cpu.usage, network.interfaces	Standard user

Pack	Probes	Required Access
Docker	containers.list, logs.tail, images.list, daemon.info	`docker` group
systemd	services.list, service.status, journal.query	`systemd-journal` group
Postgres	databases.list, connections.active, query.slow	Read-only DB role
nginx	config.test, access.log.tail, error.log.tail	`adm` group
Redis	info, keys.count, memory.usage	Redis AUTH
Splunk	Search via SPL, query indexes	API token
Elasticsearch	Query indices, cluster.health	API token
Loki	Query via LogQL, stream tailing	API token
Datadog	Query logs, list monitors	API key

Permission Model

Every pack explicitly declares what access it needs in its manifest:

```
Pack: docker
Requires:
  Groups:   [docker]
  Commands: [docker]
  Files:    [/var/run/docker.sock]
```

Packs enter a **pending** state until an administrator explicitly approves the required permissions. No pack activates silently. This is audited and visible in the management interface.

Pack Lifecycle

Packs can be installed, updated, and removed without restarting the agent. Updates that require new permissions re-enter the pending state for approval. All pack operations are logged to the audit trail.

Security Model

Sonde implements defense-in-depth with **nine security layers**. No single layer is the entire defense; they reinforce each other.

Layer 1: Agent Privilege Isolation

Agents run as a dedicated unprivileged `sonde` system user — never root. Access to resources (Docker socket, log files, databases) is granted via OS group membership, reviewed and approved per-pack. The agent binary refuses to start as root.

Layer 2: No Raw Shell Execution

There is no code path from the AI to arbitrary shell commands. The hub sends **structured probe descriptors** (e.g., `{ probe: "docker.containers.list", params: { all: true } }`). The agent maps these to pre-defined, internally-implemented probe functions. An attacker who compromises the hub cannot execute arbitrary commands on agents.

Layer 3: Capability Ceilings

Three capability levels control what operations are possible:

Level	Scope	Example
observe	Read-only (default)	List containers, read logs, check disk
interact	Safe mutations	Restart a service, clear cache
manage	Full control	Delete containers, modify configs

Each agent has a configured ceiling. Probes above the ceiling cannot be loaded — the code path does not exist at runtime.

Layer 4: Agent-Hub Wire Security

- **TLS transport** — All WebSocket communication over WSS
- **Mutual TLS (mTLS)** — Hub issues client certificates during enrollment; both sides verify identity on every connection
- **Enrollment tokens** — Single-use, time-limited (15 minutes), burned after certificate exchange

- **Payload signing** — Every message signed with sender's private key; tampered messages rejected

Layer 5: Client Authentication

- **OAuth 2.0** for Claude.ai MCP connections
- **API keys** for all other clients (scoped per key)
- **Session tokens** with short TTL and rotating refresh
- **Client allowlisting** to restrict which origins can connect
- **Per-key policy scoping** — each API key is restricted to specific agents, tools, and capability levels

Layer 6: Output Sanitization

All probe output is scrubbed before leaving the agent:

- Environment variables matching `*_KEY`, `*_SECRET`, `*_TOKEN`, `*_PASSWORD`
- Connection strings (database URLs, API endpoints with credentials)
- Contents of `.env` files
- Custom patterns configurable per-deployment

Sanitization runs **before** payload signing, ensuring scrubbed output is what gets cryptographically attested.

Layer 7: Signed Packs

Official packs are code-signed during the CI/CD build pipeline. Agents verify signatures before loading. Unsigned or tampered packs are rejected by default. Operators can opt in to unsigned packs for custom/internal use.

Layer 8: Agent Attestation

On enrollment, each agent generates a fingerprint (OS version, binary hash, installed packs, configuration). The hub stores this attestation record. On subsequent connections, the agent re-attests. Unexpected changes (binary swap, config tampering) are flagged, and the agent can be quarantined pending investigation.

Layer 9: Tamper-Evident Audit Trail

Every operation is logged on both the agent and hub in an append-only, hash-chained audit log. Each entry includes a cryptographic hash of the previous entry, making tampering detectable. The log records: who requested the action, what probe was run, which agent executed it, the result, and the timestamp.

Deployment

Hub Deployment

The hub is distributed as a single Docker container. Four deployment paths are supported:

Path	Best For	Complexity
One-liner installer	Most users	Low — interactive script handles everything
Dokploy	Existing Dokploy users	Low — one-click from GitHub
Docker Compose	Power users with existing stacks	Medium — manual config
Cloud marketplace	Cloud-native teams	Low — pre-configured images

The hub supports three networking modes: public domain with automatic Let’s Encrypt TLS, Cloudflare Tunnel for zero-port-forwarding access, and local/Tailscale for internal-only deployments.

On first boot, the hub detects it has no database and serves a setup wizard that walks through admin account creation, AI tool connection, and first agent enrollment.

Agent Deployment

Agents install via a single command and an interactive terminal UI guides through pack selection and permission approval:

```
curl -fsSL https://sonde.dev/install | sh
```

The installer:

1. Creates the `sonde` system user
2. Connects to hub and exchanges enrollment credentials
3. Scans the system for installed software
4. Presents available packs for selection
5. Requests permission approval for selected packs
6. Registers as a systemd service and starts

Agents require no inbound port access. They work behind NAT, firewalls, and in cloud VPCs without networking changes.

Hub Resource Requirements

Scale	CPU	RAM	Storage	Notes
1–10 agents	1 vCPU	512MB	1GB	SQLite, single container
10–100 agents	2 vCPU	1GB	5GB	SQLite or Postgres
100+ agents	4 vCPU	2GB	10GB+	Postgres recommended

Operations

Hub Web Dashboard

A built-in web dashboard provides fleet management without command-line access:

- **Fleet view** — All agents with real-time status, packs, and last check-in

- **Agent detail** — Probe history, audit log, health metrics per agent
- **Enrollment** — Generate tokens, watch agents appear in real-time
- **API key management** — Create, scope, and revoke keys
- **Policy editor** — Configure per-key and per-agent restrictions
- **Audit log** — Searchable, filterable, with hash chain integrity verification
- **Dry-run panel** — Test diagnostics without AI; see what probes would fire

Agent Management

Agents provide an interactive terminal UI (TUI) for local management:

- Live pack status and probe activity
- Pack installation and removal
- Permission management
- Local audit log viewer
- Health status and hub connection info

Updates

Component	Update Mechanism
Hub	Docker image pull (Watchtower, Dokploy webhook, or manual)
Agent	`sonde update` command (downloads, verifies, restarts)
Packs	`sonde packs update` (signature-verified, hot-swap if no new permissions)

The hub automatically runs database migrations on startup. Agents report their version on connect; outdated agents are flagged in the dashboard.

Enterprise Considerations

Compliance and Auditability

- **Complete audit trail** of every diagnostic action across the fleet
- **Hash-chained logs** provide tamper evidence for compliance reviews
- **Per-key policy scoping** ensures least-privilege access for different teams or AI tools
- **No data leaves the network** — the hub is self-hosted; probe results flow hub → AI client, never to third parties (unless the AI client is cloud-hosted)

Multi-Team Access Control

API keys can be scoped to restrict access:

- **Agent scope** — Key can only query specific agents (e.g., "team-A servers only")
- **Tool scope** — Key can only use certain tools (e.g., "observe only, no interact")
- **Capability ceiling** — Key cannot exceed a capability level regardless of agent config

This enables safe multi-team usage where each team's AI assistant only sees their infrastructure.

Data Sovereignty

Sonde is **fully self-hosted**. The hub runs on your infrastructure. No telemetry, no phone-home, no third-party data sharing. Probe results never transit external servers (unless routed through a cloud-hosted AI client, which is the user's choice).

Network Security

- Agents connect outbound only — no inbound ports required
- All traffic encrypted with TLS
- mTLS provides mutual identity verification
- Works behind corporate firewalls, NAT, and VPNs
- Cloudflare Tunnel option eliminates all port exposure

AI Model Independence

Sonde does not lock you into a specific AI vendor. The MCP protocol and REST API are model-agnostic. You can use Claude today, switch to GPT tomorrow, or use an internally-hosted open-source model — without changing your Sonde deployment.

Technology Stack

Component	Technology	Rationale
Language	TypeScript / Node.js 22	Single language across hub, agent, dashboard
Hub HTTP	Hono	Lightweight, fast, SSE/streaming support
Hub WebSocket	ws	Battle-tested, minimal overhead
Hub Database	SQLite (default) / Postgres	Zero-config default, enterprise option
Dashboard	React 19 + Vite + Tailwind	Modern, fast build, widely known
Agent TUI	Ink v5	React for terminals, consistent developer experience
MCP SDK	@modelcontextprotocol/sdk	Official MCP implementation
Validation	Zod	Runtime type safety on all protocol boundaries
CI/CD	GitHub Actions + Turborepo	Monorepo-native, fast cached builds
Container	Docker (node:22-alpine)	Minimal image size, standard deployment

Summary

Sonde eliminates the copy-paste friction between AI assistants and infrastructure. It provides a secure, auditable, model-agnostic channel for AI-driven diagnostics across any fleet size. Self-hosted, extensible via packs, and built with enterprise security requirements from day one.

One sentence: Sonde lets any AI assistant troubleshoot your servers as fast as a senior engineer — without giving it the keys.

Repository: `github.com/physikal/sonde` *(private)*

Contact: Josh Owen