lazyclaw

Linux TUI for Monitoring OpenClaw Instances

Document Type: **Design Document & Software Requirements Specification**
Version: 1.0
Date: February 2026
Status: Draft

Table of Contents

PART I

Design Document

1. Purpose and Scope

Goal: Build a terminal-first, keyboard-driven monitoring and operator console—similar in interaction style to lazygit and lazydocker—for observability and light operations across one or more OpenClaw Gateways ("instances").

1.1 In Scope (v1)

1. Discover and connect to multiple OpenClaw Gateways (local and remote).
2. Show live status: gateway reachability, protocol/auth status, channels readiness, agents, sessions, and health snapshot.
3. Stream and tail logs and system events.
4. Provide safe operator actions (non-destructive by default; destructive gated by confirmations).
5. Provide configuration and persistence (profiles, UI state, last selection, filters).

1.2 Out of Scope (v1)

- Building a full chat client rivaling OpenClaw Control UI.
- Deep device-pairing UX that replaces openclaw devices approve workflows. Integration may follow in a future release.

2. Target Users and Scenarios

Primary user: a developer or operator running one or more OpenClaw Gateways (local workstation, home server, VPS), wanting quick triage and confidence that everything is running correctly.

2.1 Typical Scenarios

- "Is the gateway up? Which channels are connected? Any errors in logs?"
- "Which agent has active sessions? Any stuck tool calls?"
- "Did a recent update break WhatsApp linking? What does health/doctor report?"
- "I have 3 hosts—show me a single dashboard view."

3. Product Principles

Borrowed from the lazygit and lazydocker design patterns that have proven effective for terminal-first workflows:

1. **One screen, minimal typing.** Navigation and actions are performed via keyboard shortcuts and menus, minimizing context switching.
2. **Context-sensitive help always visible.** Bottom bar hints show relevant shortcuts; ? reveals a full help overlay.
3. **Strong configuration UX.** YAML config in standard XDG-compliant location with schema support; open/edit config from inside the app.
4. **Persistent state.** Restore layout, last selected instance/panel, and filters across restarts (mirroring lazygit's state.yml).
5. **Safety rails.** Confirmations for destructive actions; the UI shows exactly what will happen before executing.

4. High-Level Architecture

4.1 Components

4.1.1 TUI Frontend

- Layout manager (panes, tabs)
- Input/keybinding router
- Modal system (help, actions, confirmations, errors)

4.1.2 Core Domain Layer

- Instance registry (profiles, groups/tags)
- "Selected context" state machine (selected instance → selected view → selected entity)
- Normalized domain models: GatewayStatus, ChannelStatus, Agent, Session, HealthSnapshot, LogEvent, Presence

4.1.3 Data/Transport Layer (OpenClaw Connectivity)

Gateway WS Client (primary): Implements OpenClaw Gateway protocol handshake (connect). Supports operator scopes and token authentication.

SSH Tunnel Helper (optional but recommended): Creates and maintains ssh -L forwards for remote gateways, allowing lazyclaw to connect to 127.0.0.1 which is auto-approved by OpenClaw.

CLI Adapter (fallback mode): Executes openclaw status, openclaw health --json, openclaw gateway status --json where available, and parses JSON output. Useful when the WS protocol evolves or WS access is blocked.

4.1.4 Storage

- config.yml — profiles and preferences
- state.yml — UI persistence
- Secrets store (preferred: OS keyring; fallback: encrypted file)

4.2 Why WebSocket-First

OpenClaw defines the Gateway WebSocket protocol as the single control plane for all clients (CLI, web UI, apps, nodes). Therefore lazyclaw treats WebSocket as canonical, with CLI as a fallback for environments where WS access is restricted.

5. OpenClaw Integration Design

5.1 Connection Modes

Mode	Transport	Notes
Local	ws://127.0.0.1:18789	Default. Auto-approved by OpenClaw.
SSH Tunnel	ssh -L to remote 127.0.0.1:18789	Recommended for remote operators. Remote gateway sees a localhost connection, bypassing pairing requirements.
Direct Remote	ws://<host>:18789 or wss://	For tailnet/LAN setups. Requires token and possibly device pairing approval.

5.2 Authentication and Handshake

lazyclaw implements the Gateway protocol handshake sequence:

1. Pre-connect challenge event (connect.challenge) is received from the gateway.
2. Client sends a connect request specifying: protocol range, role: operator, scopes (read-only by default; upgrade to write if user enables), auth.token, and device identity fields when required in direct-remote mode.
3. Gateway acknowledges connection or returns an error (e.g., pairing required, token rejected).

Design choice: Default to read-only scopes until the user explicitly enables “write actions” per instance.

5.3 Data Sources

The following data sources are used to populate the UI:

- **Health snapshot:** openclaw health --json provides a structured gateway health snapshot.
- **Status overview:** openclaw status provides a local summary including gateway reachability, mode, and session activity.
- **Channels and sessions:** First-class entities accessible via CLI (openclaw sessions) and Control UI references.
- **Logs:** Gateway WS logs with normal/verbose modes; CLI provides log follow.

6. TUI/UX Specification

6.1 Default Layout

The layout follows the proven lazydocker “left resources → right details (tabs)” model:

6.1.1 Left Pane: Instances

- List of configured Gateways (name, tags, connection mode)
- Status badges: OK / DEGRADED / DOWN
- Filter box activated by /

6.1.2 Main Pane: Details (Tabbed)

Tabs available for the selected instance:

#	Tab	Content
1	Overview	Reachability, protocol/auth status, versions, uptime, last error
2	Health	Parsed gateway health snapshot (probe durations, per-channel health summaries)
3	Channels	Channel readiness, auth age, last reconnect
4	Agents	Configured agents, workspace path, active turn indicator
5	Sessions	Active sessions, recent recipients, compaction status indicator
6	Logs	Follow/tail with search and level filters
7	Events	Ephemeral system events feed

6.1.3 Bottom Bar

Contextual key hints displayed at all times (e.g., “press ? for help”), following the lazydocker convention.

6.2 Keybindings (Proposed Defaults)

Key	Action
q	Quit
?	Help overlay (context-sensitive)
/	Filter/search (instances list, logs search)
Tab / Shift+Tab	Switch focus between panes
1–7	Switch tabs in the Details pane
x	Actions menu for selected item
Enter	Drill down / select
Esc	Back / close modal

6.3 Actions Menu (Safe by Default)

Actions are contextual to the selected resource:

6.3.1 Instance Actions

- Open Control UI URL
- Copy WS URL
- Reconnect
- Toggle verbose logs
- Run troubleshooting ladder (guided: status → gateway status → logs → doctor → channels probe)

6.3.2 Channel Actions

- Probe channel (where available)
- Show last error

6.3.3 Session Actions

- Inspect
- Set overrides (if supported; Control UI mentions per-session overrides)

6.3.4 Write/Destructive Actions

Gated behind the “Enable write actions” toggle and require explicit confirmation:

- Restart gateway service (only when instance has an SSH/host command configured)
- Logout/login channels (explicit warning; shows the exact commands it will run)

7. Configuration and Persistence

7.1 Config Files

Following lazygit and lazydocker conventions:

- ~/.config/lazyclaw/config.yml — Global configuration
- ~/.config/lazyclaw/state.yml — Persistent UI state (mirroring lazygit’s state.yml approach)

7.2 Config Schema (High-Level)

```
instances:
  - name: "home-gateway"
    tags: ["home", "stable"]
    mode: "ssh_tunnel"          # local | ssh_tunnel | direct
    ws_url: "ws://127.0.0.1:18789"
    auth:
      token_ref: "keyring:openclaw_home_token"
    ssh:
      host: "home-server"
```

```
      user: "tav"
      local_port: 18790
      remote_host: "127.0.0.1"
      remote_port: 18789

ui:
  theme: "auto"
  refresh_ms: 1000
  log_tail_lines: 5000
  confirm_destructive: true

security:
  default_scopes: ["operator.read"]
  allow_write_scopes: false
```

7.3 In-App Config Editing

Mimics lazydocker's ergonomics: o to open config directory, e to edit the config file in the user's preferred editor ($EDITOR).

8. Observability Model

This section defines what "monitoring" means in v1.

8.1 Health Levels

A simple tri-state is computed per instance:

Level	Criteria
OK	Gateway reachable, health snapshot succeeds, no channel failures.
DEGRADED	Gateway reachable but health indicates channel failures, frequent reconnects, or auth issues.
DOWN	Gateway unreachable or handshake/auth fails.

8.2 Metrics (v1)

No heavy Prometheus infrastructure is required. The v1 metrics surface includes:

- Probe durations and last health timestamp (from health snapshot)
- Session counts and recent activity indicators (from status/sessions)
- Log error rate (count errors in last N lines; configurable threshold)

9. Security Design

9.1 Secrets Handling

- Prefer OS keyring for token storage.
- If file-based: encrypt at rest; never print tokens in UI (mask with ••••).
- Support token entry via modal; store as keyring reference.

9.2 Principle of Least Privilege

- Default scopes are read-only.
- Require explicit opt-in per instance to request write scopes in the connect handshake.

9.3 Device Pairing Strategy

Given OpenClaw's device pairing requirement for remote devices, the recommended default is SSH tunnel mode to minimize pairing complexity. Direct remote mode remains supported for tailnet setups where pairing can be managed separately.

10. Implementation Plan

10.1 Milestone 1 — Single Instance Monitor

- WS handshake and Overview tab
- Logs tail and search
- Config and state persistence

10.2 Milestone 2 — Multi-Instance Dashboard

- Instance list with health rollups
- Parallel WS connections with exponential backoff

10.3 Milestone 3 — Ops Helpers

- Guided troubleshooting ladder
- SSH tunnel management
- Optional write actions with safety confirmations

PART II

Software Requirements Specification

1. Introduction

1.1 Product Overview

lazyclaw is a Linux TUI for monitoring and operating OpenClaw Gateways. OpenClaw's Gateway is a long-running process controlling channels and state, and exposes a WebSocket control plane used by CLI and UIs.

1.2 Definitions

Term	Definition
Instance	An OpenClaw Gateway endpoint (local or remote).
Operator Client	A client connecting to Gateway WS with operator role and scopes.
Health Snapshot	Structured health information queried from gateway (openclaw health --json).

2. Overall Description

2.1 User Needs

- Quickly answer: "Is everything running and connected?"
- See failures fast (logs and health)
- Switch between multiple hosts without reconfiguring OpenClaw CLI
- Safe, keyboard-first operations

2.2 Assumptions and Dependencies

- User has OpenClaw Gateways running and reachable (locally or via tunnel/direct transport).
- Gateway protocol supports the connect handshake and operator roles.
- For some diagnostic/ops tasks, the openclaw CLI may be installed (optional fallback).

2.3 Constraints

- Must run in a terminal (TTY), including over SSH.
- Must remain usable without a mouse.

3. External Interface Requirements

3.1 User Interface (TUI)

ID	Requirement
UI-1	The UI shall provide a split layout with: instances list pane, detail pane with tabs, and bottom bar showing contextual hints.
UI-2	The UI shall provide a help overlay accessible via ?, listing context-sensitive keybindings.
UI-3	The UI shall provide an actions menu (default key x) that shows available actions for the current selection.

3.2 Configuration Interface

ID	Requirement
CFG-1	The program shall load configuration from ~/.config/lazyclaw/config.yml.
CFG-2	The program shall persist UI state (layout/selection/filter) into ~/.config/lazyclaw/state.yml.
CFG-3	The program shall support multiple instances with per-instance connection mode (local, ssh_tunnel, direct).

3.3 Network/Protocol Interface

ID	Requirement
NET-1	The program shall connect to OpenClaw Gateway via WebSocket text frames using JSON payloads.
NET-2	The program shall implement the Gateway handshake: receive connect.challenge, then send a connect request with operator role and scopes.
NET-3	The program shall support token authentication via connect.params.auth.token.

4. Functional Requirements

4.1 Instance Management

ID	Requirement
FR-IM-1	The system shall display a list of configured instances with current connection status (OK / DEGRADED / DOWN).
FR-IM-2	The system shall allow adding/editing/removing instances by editing the config file (v1), with in-app “open/edit config” convenience.
FR-IM-3	The system shall support filtering instances by name/tag via / search.

4.2 Connection and Reconnection

ID	Requirement
FR-CN-1	For each instance, the system shall establish a WS connection and complete handshake within a configurable timeout.
FR-CN-2	The system shall implement exponential backoff reconnection on disconnect.
FR-CN-3	The system shall show handshake/auth failures clearly (e.g., “pairing required,” “token rejected”).

4.3 Overview and Health

ID	Requirement
FR-HL-1	The system shall display an Overview tab showing: gateway reachability, last successful handshake time, selected scopes (read/write), and update/version metadata when available.
FR-HL-2	The system shall fetch and display a gateway health snapshot (Health tab).
FR-HL-3	The system shall mark instance health as DEGRADED if any channel probe/summary indicates failure or if logs show frequent errors (threshold configurable).

4.4 Channels View

ID	Requirement
FR-CH-1	The system shall list channels and their readiness/connected state.
FR-CH-2	The system shall allow “probe channel” (read-only action) when supported by the gateway/CLI.

4.5 Agents View

ID	Requirement
FR-AG-1	The system shall list configured agents (name/id) and show which agent is currently handling activity when that signal is available.

4.6 Sessions View

ID	Requirement
FR-SE-1	The system shall list stored sessions and indicate which are active/recent.
FR-SE-2	The system shall support filtering sessions and viewing per-session details.

4.7 Logs View

ID	Requirement
FR-LG-1	The system shall stream logs in a scrollable view with follow mode, and allow searching within logs via /.
FR-LG-2	The system shall support a “verbose protocol logging” toggle where available.

4.8 Guided Troubleshooting

ID	Requirement
FR-TS-1	The system shall provide a guided troubleshooting flow that runs the recommended command ladder (or WS equivalents): status, gateway status, logs --follow, doctor, channels status --probe.

5. Non-Functional Requirements

5.1 Performance

ID	Requirement
NFR-P-1	UI refresh shall be configurable (default 1s) and must not exceed 5% CPU on an average workstation when monitoring up to 3 instances.
NFR-P-2	WS reconnect attempts shall be rate-limited to avoid flooding (exponential backoff).

5.2 Reliability

ID	Requirement
NFR-R-1	If one instance connection fails, the UI shall remain responsive and other instances shall continue updating.
NFR-R-2	State persistence shall be crash-safe (atomic write of state.yml).

5.3 Security

ID	Requirement
NFR-S-1	Tokens shall never be rendered in plaintext in the UI.
NFR-S-2	Default scopes shall be read-only; write scopes must require explicit user opt-in per instance.
NFR-S-3	For direct remote connections, the system shall support device identity/pairing flows or clearly instruct the user when pairing approval is required.

5.4 Portability

ID	Requirement
NFR-O-1	The application shall run on Linux terminals commonly used over SSH (xterm-256color compatible).

6. Data Model (Logical)

6.1 Core Entities

InstanceProfile

- id, name, tags
- connection mode (local | ssh_tunnel | direct)
- ws_url
- auth token reference
- optional SSH configuration

ConnectionState

- connected (boolean), last_error, last_handshake_ts, scopes

HealthSnapshot

- probe_duration_ms, per-channel summaries, session summary

ChannelStatus / Agent / Session

- Minimal fields required for list, details, and filters

LogEvent

- ts, level, source, message, raw

7. Acceptance Criteria (v1 Definition of Done)

6. User can configure 2+ instances and see them in list with live status.
7. Selecting an instance shows Overview, Health, and Logs tabs.
8. Help overlay (?) and actions menu (x) work across panes.
9. SSH tunnel mode works for remote gateway monitoring (at least by invoking ssh -L).
10. Config and state persistence work; app restores last selection after restart.
11. No secrets leak in UI, logs, or crash output.

Appendix A: Design Rationale

A.1 Why This Feels Like a lazy* Tool

lazyclaw adopts lazydocker's interaction pattern: a sidebar of resources coupled with tabbed details (logs, stats, config analogs), a context-sensitive action menu, and always-available shortcuts and help. It also adopts lazygit's strengths: persistent UI state (state.yml), heavy configurability via YAML, and extensibility patterns that pave the way for custom commands and keybindings in later versions.

A.2 Patterns Borrowed from lazygit

- "One-screen workflow" with persistent user config and UI state file
- Configurable and extensible via YAML (keybindings, themes, command overrides)
- Power-user extensibility via custom command keybindings
- Terminal-first flow philosophy: minimal context switching, frictionless frequent operations

A.3 Patterns Borrowed from lazydocker

- Split layout: navigation sidebar (resources) + detail panel with tabs (logs/stats/config)
- Context-sensitive help: bottom line shows shortcuts; ? reveals help screen; x opens actions menu; / searches
- YAML config with known standard paths and JSON schema; in-app affordances to open/edit config