Software Architecture

This chapter has documentation on the software architecture

- WebSockets Information and Design

WebSockets Information and Design

PatchMon WebSockets

This document describes how WebSockets are used in PatchMon: endpoints, authentication, WS vs WSS behaviour, and security.

Overview

PatchMon uses a single HTTP server with `noServer: true` WebSocket handling. All WebSocket upgrades are handled in one place (`server.on("upgrade")`) and routed by path:

Path pattern	Purpose	Clients
`/api/{v}/agents/ws`	Agent ↔ server persistent connection	PatchMon agent (Go)
`/api/{v}/ssh-terminal/:hostId`	Browser SSH terminal to a host	Frontend (browser)
`/bullboard*`	Bull Board queue UI real-time updates	Bull Board (browser)

Implementation lives in:

- **Backend:** `backend/src/services/agentWs.js` (agent + Bull Board), `backend/src/services/sshTerminalWs.js` (SSH terminal).

WebSocket Endpoints

Agent WebSocket

- **Path:** `/api/{version}/agents/ws` (e.g. `/api/v1/agents/ws`)
- **Auth:** HTTP headers on the upgrade request: `X-API-ID`, `X-API-KEY` (validated against `hosts` and API key utils).
- **Purpose:** Persistent connection from each agent to the server for:
  - Heartbeat / presence

- Commands from server (e.g. trigger update, compliance scan)
- Events from agent (e.g. Docker status)

SSH Terminal WebSocket

- **Path:** `/api/{version}/ssh-terminal/:hostId` (e.g. `/api/v1/ssh-terminal/abc-123`)
- **Auth:** One-time ticket (query `ticket=...`) or legacy JWT (`token=...` or `Authorization: Bearer ...`). User must have `can_manage_hosts` (or admin).
- **Purpose:** Browser opens a WebSocket to the backend; backend either connects directly to the host via SSH or proxies through the agent WebSocket to the host.

Bull Board WebSocket

- **Path:** `/bullboard` (prefix match)
- **Auth:** Session cookie `bull-board-session` or `Authorization` header.
- **Purpose:** Real-time updates for the Bull Board queue UI (echo-style).

WS vs WSS (Secure vs Insecure)

How each part of the system decides between **ws** (insecure) and **wss** (secure):

Server (backend)

The server **does not choose** the protocol; it **detects** whether the incoming connection is secure and records it for logging and metadata:

- **`socket.encrypted`** — `true` when the TCP socket is TLS (direct wss to Node).
- **`request.headers["x-forwarded-proto"] === "https"`** — `true` when TLS is terminated at a reverse proxy (e.g. Nginx) that sends the original protocol.

So: if the client connects over TLS, or the proxy sets `X-Forwarded-Proto: https`, the server treats the connection as secure (wss). Otherwise it is treated as ws.

Code: `backend/src/services/agentWs.js` (e.g. `isSecure`, `connectionMetadata`, log line with `protocol=wss|ws`).

Agent (Go)

The agent **converts** the configured server URL to a WebSocket URL and uses that to connect:

Configured URL	WebSocket URL
`https://...`	`wss://...`
`http://...`	`ws://...`
Already `wss://` or `ws://`	Used as-is
No protocol	Assumed HTTPS → `wss://...`

The path is then appended: `.../api/{version}/agents/ws`.

Code: `agent-source-code/cmd/patchmon-agent/commands/serve.go` (`connectOnce`, URL conversion and `wsURL`).

Frontend (browser)

The frontend (e.g. SSH terminal) builds the WebSocket URL so it **matches the page**:

- Page loaded over **HTTPS** → `wss:`
- Page loaded over **HTTP** → `ws:`

So the same app works on http and https without extra configuration.

Code: `frontend/src/components/SshTerminal.jsx` (e.g. `protocol = window.location.protocol === "https:" ? "wss:" : "ws:"`).

Authentication

- **Agent WS:** Credentials only on the **upgrade request** via `X-API-ID` and `X-API-KEY`. No auth in query or body. Keys are validated (including bcrypt/legacy) before the WebSocket is accepted.
- **SSH terminal:** Prefer **one-time ticket** in query (`ticket=...`); ticket is consumed once. Legacy: `token=...` or `Authorization: Bearer ...`. User must be authorized for the host (e.g. `can_manage_hosts`).
- **Bull Board:** Session cookie or `Authorization` header; required before accepting the upgrade.

All three paths reject the upgrade (e.g. 401) if auth fails; the WebSocket is never established.

Message Types and Flows

- **Agent → Backend:** e.g. `docker_status`, pings. JSON over the agent WebSocket.

- **Backend → Agent:** e.g. `update_agent`, `compliance_scan`, SSH proxy payloads. Sent over the same WebSocket by api_id.
- **SSH terminal:** Browser sends `connect` with SSH options; backend (or agent) establishes SSH and forwards terminal I/O over the WebSocket. Resize and other control messages are defined in `sshTerminalWs.js` and the frontend.

Security Notes

- **TLS in production:** Use HTTPS and WSS. The agent assumes WSS when the server URL is `https://` or has no scheme.
- **Proxy:** When behind Nginx (or similar), ensure `X-Forwarded-Proto: https` is set for HTTPS so the backend correctly detects secure connections.
- **Agent TLS verification:** `skip_ssl_verify` is blocked when `PATCHMON_ENV=production`; avoid disabling TLS verification in production.
- **SSH terminal:** Prefer one-time tickets; avoid putting long-lived tokens in URLs (logs, history). Permissions are enforced per user/role (`can_manage_hosts`).
- **Brute force / abuse:** Upgrade is rejected with 401/404 before the WebSocket is created; no WebSocket resource is exposed without valid auth.