PatchMon PATCHMON DOCS

PatchMon Admin Guide

Day-to-day usage guide for PatchMon administrators and operators working in the web UI.

LAST UPDATED

22 August 2026

CHAPTERS

27

READ ONLINE

https://patchmon.net/docs/patchmon-admin-guide

SOURCE

github.com/PatchMon/PatchMon

Contents

1. Table of Contents
2. Chapter 1: Welcome to PatchMon
3. Chapter 2: Settings in the Web UI
4. Chapter 3: Adding a Host
5. Chapter 4: Host Detail Page
6. Chapter 5: Managing Host Groups
7. Chapter 6: Package Inventory
8. Chapter 7: Repository Tracking
9. Chapter 8: Patching Overview
10. Chapter 9: Running a Patch
11. Chapter 10: Patch Policies and Scheduling
12. Chapter 11: Patch History and Live Logs
13. Chapter 12: Enabling Docker Integration
14. Chapter 13: Docker Inventory Tour
15. Chapter 14: Compliance Overview
16. Chapter 15: Running Compliance Scans
17. Chapter 16: Compliance Results and Remediation
18. Chapter 17: Alerts Overview
19. Chapter 18: Notification Destinations
20. Chapter 19: Notification Routes and Delivery Log
21. Chapter 20: Scheduled Reports
22. Chapter 21: Web SSH Terminal
23. Chapter 22: RDP via Guacamole
24. Chapter 23: AI Terminal Assistant
25. Chapter 24: Users, Roles, and RBAC
26. Chapter 25: Two-Factor Authentication
27. Chapter 26: Metrics and Telemetry

This is the day-to-day usage guide for PatchMon administrators and operators working in the web UI. For installing or running the PatchMon server itself, see the **Operator Guide**. For integrations and the REST API, see the **API & Integrations Guide**.

Table of Contents

- Chapter 1: Welcome to PatchMon
- Chapter 2: Settings in the Web UI
- Chapter 3: Adding a Host
- Chapter 4: Host Detail Page
- Chapter 5: Managing Host Groups
- Chapter 6: Package Inventory
- Chapter 7: Repository Tracking
- Chapter 8: Patching Overview
- Chapter 9: Running a Patch
- Chapter 10: Patch Policies and Scheduling
- Chapter 11: Patch History and Live Logs
- Chapter 12: Enabling Docker Integration
- Chapter 13: Docker Inventory Tour
- Chapter 14: Compliance Overview
- Chapter 15: Running Compliance Scans
- Chapter 16: Compliance Results and Remediation
- Chapter 17: Alerts Overview
- Chapter 18: Notification Destinations
- Chapter 19: Notification Routes and Delivery Log
- Chapter 20: Scheduled Reports
- Chapter 21: Web SSH Terminal
- Chapter 22: RDP via Guacamole
- Chapter 23: AI Terminal Assistant
- Chapter 24: Users, Roles, and RBAC
- Chapter 25: Two-Factor Authentication
- Chapter 26: Metrics and Telemetry

Chapter 1: Welcome to PatchMon

PatchMon (https://patchmon.net) is an open-source patch management and infrastructure monitoring platform that gives sysadmins and IT teams centralised visibility over patches, packages, compliance, and remote access across their entire server fleet.

It works with standard Linux package managers (**apt**, **yum**, and **dnf**) and requires no inbound ports on your monitored hosts.

How It Works

PatchMon uses a lightweight agent model:

Deploy the Server. Self-host PatchMon using Docker or the native installer, or use the managed PatchMon Cloud (https://patchmon.net).

Install the Agent. Add a host in the dashboard and run the one-liner install command on your Linux server.

Monitor. The agent sends system and package data outbound to PatchMon on a schedule. No inbound ports need to be opened on your servers.

> ***Network requirements:*** *Agents only need outbound access on port 443 (HTTPS). If your systems are behind firewalls that inspect SSL/DNS traffic or are air-gapped, adjust your rules accordingly.*

Key Features

Area	Details
Dashboard	Customisable per-user card layout with fleet-wide overview
Host Management	Host inventory, grouping, and OS detail tracking
Package Tracking	Package inventory, outdated package counts, and repository tracking per host
Compliance Scanning	OpenSCAP CIS Benchmark scans and Docker Bench for Security (scheduled or on-demand)
Docker Monitoring	Container discovery and status tracking across your hosts
Agent System	Lightweight agents with outbound-only communication. No attack surface on your servers.
Remote Access	In-browser RDP via Guacamole and SSH terminal with AI-assisted analysis
AI Analysis	AI-powered assistance inside the SSH terminal
Users & Auth	Multi-user accounts with roles, permissions, and RBAC
OIDC SSO	Single Sign-On via external identity providers (e.g. Authentik, Keycloak, Entra ID)
TOTP 2FA	Time-based one-time password two-factor authentication
Auto-Enrollment	Automatic agent enrollment for Proxmox LXC containers
API	REST API with JWT authentication under `/api/v1`
Rate Limiting	Configurable rate limits for general, auth, and agent endpoints

---

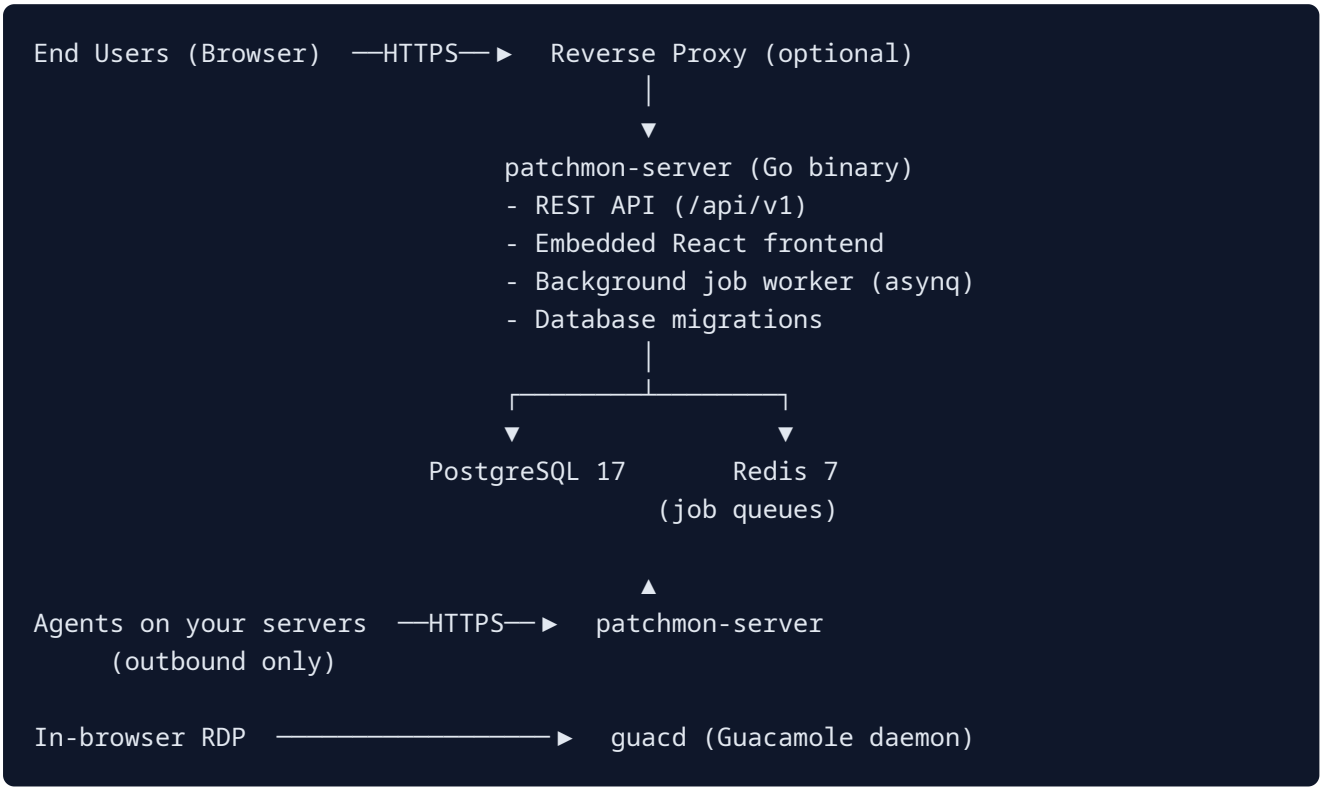
Quick Links

- Installing PatchMon Server on Docker
- Installing the PatchMon Agent
- Proxmox LXC Auto-Enrollment Guide
- PatchMon Environment Variables Reference
- Metrics and Telemetry
- Feature Roadmap (https://feedback.patchmon.net/roadmap) (request and vote on features)
- Report a Bug (https://github.com/PatchMon/PatchMon/issues) (bugs are tracked in GitHub, not the feedback portal)
- YouTube (https://www.youtube.com/@patchmonTV)
- Discord Community (https://patchmon.net/discord)

GitHub Repository (https://github.com/PatchMon/PatchMon)

Architecture

PatchMon is a **single Go binary** that serves both the API and the embedded React frontend. There is no separate frontend container or web server. The binary also runs database migrations automatically on startup.



Component	Technology
Server	Go single binary (API + embedded frontend + migrations)
Frontend	React + Vite (embedded in the server binary)
Database	PostgreSQL 17
Job Queue	Redis 7 (via asynq)
RDP Gateway	guacd (Apache Guacamole daemon), optional (required for RDP)

Support

Discord: patchmon.net/discord (https://patchmon.net/discord)

Email: support@patchmon.net

GitHub Issues: Report a bug (https://github.com/PatchMon/PatchMon/issues)

License

PatchMon is licensed under AGPLv3 (https://github.com/PatchMon/PatchMon/blob/main/LICENSE).

Chapter 2: Settings in the Web UI

Overview

PatchMon 2.0 moves most day-to-day tuning out of the container's `.env` and into the **Settings** area of the web UI. From here you manage users and roles, host groups, agent update cadence, server-level toggles, branding, integrations, and authentication providers. Settings are stored in the database and the server re-reads them on every request (with a brief in-memory cache for hot paths), so most changes take effect without restarting the container.

> ***Env vars beat DB values.*** *When the same setting is present both as an environment variable and as a Settings UI value, the environment variable wins. The UI shows a small yellow "env" badge on values that are being overridden by `.env`, so you can tell at a glance why your change "didn't save". See PatchMon Environment Variables Reference for the full priority model.*

This page is the map of the Settings area: what each page does, which permission unlocks it, and which deeper chapter to read if you need more detail.

How to reach Settings

Click the cog icon in the top navigation bar, or go directly to `/settings`. You land on whatever your highest-priority settings page is (users, for people with `can_view_users`; branding, for everyone else with settings permissions).

The left sidebar groups settings into four sections:

User Management: users, roles, your own profile, and social/SSO authentication

Hosts Management: host groups and agent update behaviour

Integrations: API integrations (auto-enrolment tokens) and AI Terminal

Server: server URL, environment variables, branding, server version, and metrics

Some items only appear depending on your deployment or your edition. For example, **Server URL** and **Metrics** are only shown on the self-hosted version, and features like **Roles** (custom RBAC), **Branding**, and **AI Terminal** are gated by the corresponding capability modules on paid tiers.

Settings Pages: Quick Reference

Page	Path	Purpose	Required permission
Users	`/settings/users`	Create, edit, and disable accounts	`can_view_users` / `can_manage_users`
Roles	`/settings/roles`	Create and edit custom RBAC roles (Plus tier)	`can_manage_settings` + `rbac_custom` module
My Profile	`/settings/profile`	Your own name, email, password, MFA, trusted devices	Any authenticated user
Discord Auth	`/settings/discord-auth`	Configure Discord OAuth sign-in	`can_manage_settings`
OIDC / SSO	`/settings/oidc-auth`	Configure OpenID Connect single sign-on	`can_manage_settings`
Host Groups	`/settings/host-groups`	Organise hosts into groups for policy and visibility	`can_manage_settings`
Agent Updates	`/settings/agent-config`	Global auto-update behaviour, update interval	`can_manage_settings`
Agent Version	`/settings/agent-version`	Check and manage bundled agent binary versions	`can_manage_settings`
API integrations	`/settings/integrations`	Auto-enrolment tokens, Proxmox LXC, getHomepage, etc.	`can_manage_settings`
AI Terminal	`/settings/ai-terminal`	Configure AI provider for SSH terminal assist (Max tier)	`can_manage_settings` + `ai` module
Server URL	`/settings/server-url`	Protocol, host, and port agents use to connect back	`can_manage_settings`
Environment	`/settings/environment`	Read and edit server environment variables from the UI	`can_manage_settings`

Page	Path	Purpose	Required permission
Branding	`/settings/branding`	Upload custom logo and favicon (Plus tier)	`can_manage_settings` + `custom_branding` module
Server Version	`/settings/server-version`	Show the running version; check for updates	`can_manage_settings`
Metrics	`/settings/metrics`	Control the optional telemetry opt-in	`can_manage_settings`

Notifications, alert channels, alert settings, and patch management policies live outside the Settings area in 2.0, see Where alerts and patch policies live below.

User Management

Users

Path: `/settings/users`

Central directory of all PatchMon accounts. From here you can:

- Create new users (local username/password or OIDC-matched)
- Assign a role (`superadmin`, `admin`, `host_manager`, `user`, `readonly`, or any custom role you've created)
- Reset a user's password (admin-initiated reset, not self-serve)
- Enable, disable, or delete an account
- See when each user last logged in

Users also get a one-click button to create an auto-enrolment-style API token scoped to themselves, useful for integrations that need to act on behalf of a specific human operator.

Roles

Path: `/settings/roles` **Requires:** `rbac_custom` module (Plus tier)

The Roles editor is where custom roles are authored. A role is a named bundle of permission flags:

`can_view_dashboard`, `can_view_hosts`, `can_view_users`, `can_view_packages`, `can_view_reports`, `can_view_notification_logs`
`can_manage_hosts`, `can_manage_users`, `can_manage_settings`, `can_manage_alerts`, `can_manage_notifications`, `can_manage_compliance`, `can_manage_patching`, `can_manage_automation`, `can_manage_docker`

`can_use_remote_access` (SSH terminal and RDP)

The built-in roles (`superadmin` , `admin` , `user` , `readonly`) are immutable; the editor lets you create and edit additional roles alongside them and assign any user to any custom role.

My Profile

Path: `/settings/profile`

Your own account settings. Every authenticated user has access. Covers:

- First name, last name, email
- Password change (rate-limited; by default 5 attempts per 15-minute window)
- **Two-Factor Authentication**: enable or disable TOTP, regenerate backup codes
- **Trusted Devices**: list and revoke "remember this device" exemptions from MFA challenges
- **Dashboard preferences**: light/dark mode, card layout, default landing tab

Password policy rules are live: you cannot save a password that fails the server's policy. See PatchMon Environment Variables Reference: Password Policy.

Discord Auth

Path: `/settings/discord-auth`

Configure a Discord application as a sign-in provider. Each user can link their Discord identity from their profile page; once linked, they can sign in via the Discord button on the login page instead of typing a password.

Discord Auth is intentionally less feature-rich than OIDC SSO. There is no group-to-role mapping, no enforced-SSO mode, and no user auto-provisioning. Use it for communities and small teams; use OIDC SSO for everything else.

OIDC / SSO

Path: `/settings/oidc-auth`

Full OpenID Connect configuration: issuer URL, client ID and secret, redirect URI, scopes, button text, auto-provisioning, group-to-role mapping, and enforced-SSO toggle. A dedicated **Import from environment** button pulls existing `OIDC_*` values from `.env` into the database so you can migrate from file-based config without retyping anything.

For a step-by-step walk-through (Authentik, Keycloak, Entra ID, Okta), see Setting up OIDC SSO.

Hosts Management

Host Groups

Path: `/settings/host-groups`

Groups are the primary way to organise hosts for patching policies, alert routing, and dashboard filtering. Each host can belong to many groups; groups are purely organisational (no hierarchy, no nesting) and are referenced by name from policies, scheduled reports, and notification routes.

Agent Updates

Path: `/settings/agent-config`

Controls how and when PatchMon agents talk to the server and update themselves:

- **Update interval**: how often agents perform a check-in (default: 60 minutes); hosts with the WebSocket channel open pick up interval changes live. From v2.0.3 each tick is a hash-gated check-in: the agent ships content hashes per section, and the server only requests full content for sections whose hash has changed. Steady-state cycles are a few KB instead of a few MB.
- **Auto-update behaviour**: global on/off for automatic agent binary updates. Per-host overrides live on the host detail page.
- **Signup enabled**: whether the first-time setup wizard still serves the initial-admin endpoint.

Agent Version

Path: `/settings/agent-version`

Inspect the bundled agent binary versions (one per OS/architecture), check for newer releases upstream, and force a fresh download of the bundled binaries. Useful after you upgrade the server. Agents pick up the new binaries via the auto-update flow. No manual distribution required.

See Managing the PatchMon Agent for how agents consume this information.

Integrations

API integrations

Path: `/settings/integrations`

Auto-enrolment tokens and per-integration API credentials:

- **Auto-enrolment tokens**: one-shot or long-lived tokens that let enrolment scripts register new hosts without a human in the loop. Each token can be scoped to specific host groups and flagged for integrations like Proxmox LXC or getHomepage.
- **Integration-type tokens**: the scoped token model used by the integration `/api/*` routes, including `gethomepage` for the dashboard widget.

AI Terminal

Path: `/settings/ai-terminal` **Requires:** `ai` module (Max tier)

Configure the AI provider used by the in-browser SSH terminal's assist feature. Supported providers: OpenAI, Anthropic, Google Gemini, OpenRouter. Credentials are encrypted at rest using `AI_ENCRYPTION_KEY` (see Environment Variables Reference). The page includes a "Test connection" button so you can confirm the key works before saving.

Server

Server URL

Path: `/settings/server-url` **Hidden on:** PatchMon Cloud

Three fields (protocol, host, port) that together define the base URL agents use to reach the server. This is the same URL the first-time setup wizard asked you to confirm, persisted in the database so the UI can generate correct install commands for every new host you add.

If you change the URL later, existing agents keep using whatever URL they were installed with; only new agents pick up the change. Rerun the install command on any host you want to retarget.

Environment

Path: `/settings/environment` **Requires:** `can_manage_settings`

New in 2.0: every tunable environment variable that can be safely changed at runtime is listed here with its **effective value**, **source** (env / database / default), **default**, and a one-line description. Editable variables have an edit button; sensitive or bootstrap-only variables (like `DATABASE_URL`, `JWT_SECRET`, `REDIS_PASSWORD`, `AI_ENCRYPTION_KEY`, `SESSION_SECRET`) show as read-only and must still be changed in `.env`.

Variables are grouped by category: Database, Server, Logging, Authentication, Password policy, Server performance, Rate limits, Redis, Encryption, Deployment.

When you edit a value, the UI writes it to the database and immediately flashes a "Restart the application for changes to take effect" toast. Some settings take effect on the next request (CORS origin, log level, rate limits); others need a restart. The UI doesn't always know which is which, so the safe rule is: change, then `docker compose restart server`.

> ***Tip:*** *If you've been managing PatchMon from `.env` files for a long time and want to move configuration into the database, clear a variable from `.env` first, then change it here. Otherwise the env value keeps winning.*

Full reference: PatchMon Environment Variables Reference.

Branding

Path: `/settings/branding` **Requires:** `custom_branding` module (Plus tier)

Upload a custom logo and favicon. The assets are stored in the database and served via `GET /api/v1/settings/logos/{type}`, so they live through container restarts without a persistent volume. Both dark-mode and light-mode variants can be uploaded separately. The read path is public (so the login page can show your branding before the user authenticates) but upload/reset are gated behind the `custom_branding` module.

Server Version

Path: `/settings/server-version` **Hidden on:** PatchMon Cloud

Shows the running PatchMon server version, the latest upstream version (checked daily by the `version-update-check` automation job, see Background jobs and automation), and a manual "Check for updates" button. It does not perform the upgrade. To upgrade, PatchMon is a container-image swap (see Installing PatchMon Server on Docker).

Metrics

Path: `/settings/metrics` **Hidden on:** PatchMon Cloud

Opt-in anonymous telemetry. PatchMon sends a small heartbeat (server version, number of hosts, rough OS distribution) to the upstream metrics endpoint once a day. You can turn this off, regenerate your anonymous instance ID, or send a one-off payload immediately. See Metrics and telemetry for exactly what is sent.

Where Alerts and Patch Policies Live

In 1.4.x these lived inside Settings. In 2.0 they've moved to more natural homes:

- **Alerts (Open alerts, History)**: `/reporting` → **Alerts** tab
- **Alert Lifecycle** `/reporting` → **Alert Lifecycle** tab (retention, auto-resolve, cleanup jobs) → **Alert Lifecycle** tab (requires `alerts_advanced` module, Plus tier)
- **Destinations** (SMTP, webhook, ntfy): `/reporting` → **Destinations** tab
- **Event Rules** (routing alerts to destinations): `/reporting` → **Event Rules** tab
- **Delivery Log**: `/reporting` → **Delivery Log** tab
- **Scheduled Reports**: `/reporting` → **Scheduled Reports** tab
- **Patch Policies** (scheduling, approval rules, exclusions): `/patching?tab=policies`

The Settings sidebar doesn't list them because the pages where you actually use them (Reporting and Patching) are the right home for them. The permissions are unchanged: `can_manage_notifications`, `can_manage_alerts`, `can_manage_patching` still control who sees each area.

Troubleshooting

"I changed a setting but nothing happened"

Check the Environment page for the variable you changed. If the "Source" column says `env`, your change to the database value is being overridden by an environment variable set in `.env` or the container spec. Clear the env value and the DB value will take over.

"I saved a setting and the UI says 'Restart to take effect'"

Some settings (startup-only values like `PORT`, `DATABASE_URL`, pool sizes) are read once at boot and cached for the life of the process. Restart the `server` container:

```
docker compose restart server
```

A small number of settings (CORS origin, log level, rate-limit windows) are re-resolved on every request and don't require a restart. The UI doesn't always distinguish between them; when in doubt, restart.

"Branding / AI Terminal / Roles is greyed out"

These are paid-tier features. Self-hosted users on the free tier see them in the sidebar but can't click through; clicking redirects to an upgrade page. If you're on a paid tier and still see them as locked, click **Settings → My Profile → Subscription** to confirm the module is listed under your enabled modules.

See Also

- First-time admin setup
- PatchMon Environment Variables Reference
- Managing the PatchMon Agent
- Setting up OIDC SSO
- Background jobs and automation

Chapter 3: Adding a Host

Overview

"Adding a host" is a two-sided operation. On the server side, you pre-register the host in the web UI: give it a friendly name, pick its operating system family, optionally place it in host groups, and receive a unique API ID and API key. On the host side, run the one-line install

command that downloads and configures the agent using those credentials.

This page walks through the Add Host wizard, the install command, the Waiting for Connection screen, and what to do if the agent never shows up.

The server side is **UI-only**. You do not need shell access to the PatchMon server. Installing the agent on the target host is a separate job; see Installing the PatchMon Agent for distribution-specific prerequisites.

Permission required: `can_manage_hosts`. Users with only `can_view_hosts` see the Hosts list but not the **Add Host** button.

Before You Start

You will need:

- A **PatchMon user account** with `can_manage_hosts` (typically Admin or a custom role).
- **Console or SSH access** to the host you are adding, with root / `sudo` (Linux/FreeBSD) or Administrator (Windows).
- **Outbound HTTPS** from the host to the PatchMon server on port 443. No inbound ports are opened on the host.
- If your PatchMon server uses a **self-signed certificate**, decide up front whether to install the CA into the host's trust store or to bypass TLS verification in the install command.

Opening the Add Host Wizard

1. In the left navigation, click **Hosts**. The Hosts page loads with the **Total Hosts**, **Needs Updates**, **Needs Reboots**, and **Connection Status** summary cards at the top.
2. In the page header, click the blue **Add Host** button (icon: `+`). A modal titled **Add New Host** opens.

The wizard has four steps:

Step	What happens
1. Choose OS	Pick Linux, FreeBSD, or Windows
2. Host details	Name the host, pick groups, toggle integrations
3. Copy command	Copy the install one-liner, run it on the host
4. Connection	The wizard waits for the agent to connect and report

Step indicators at the top highlight where you are. You can go **Back** at any point before Step 3 is submitted.

Step 1: Choose OS

You pick one of three tiles:

Linux: Ubuntu, Debian, CentOS, RHEL, Rocky, Alma, Fedora, Alpine, etc.

FreeBSD: FreeBSD 13 / 14, including pfSense.

Windows: Windows 10/11 (amd64 or ARM64) and Windows Server 2019 / 2022 / 2025.

The choice controls which install command the wizard generates and which download URL the server uses. You do **not** pick an architecture (amd64 / arm64 / arm / 386) here. The install script detects it automatically on the target host and downloads the matching binary.

Click **Next** to continue.

Step 2: Host Details

This form creates the host record on the server. Three groups of fields:

Friendly Name (required)

A human-readable label such as `web-01.prod` or `billing-db`. It appears in the Hosts list, dashboards, alerts, and the URL bar (`/hosts/<id>`). It is editable later from the Host Detail page, so don't worry about getting it perfect.

The placeholder `server.example.com` is **not** used as the system hostname. The real hostname is detected when the agent first reports.

Host Groups (optional)

A checkbox list of existing groups with coloured dots next to each name. Tick any group you want the host to belong to; a host can belong to multiple groups. You can change membership later from the Hosts table or the Host Detail page.

If you have no groups yet, this section is empty. Create groups first from **Settings → Host Groups**, see Managing Host Groups.

Integrations (optional)

Two toggles:

Docker: enables container, image, volume, and network discovery via the Docker socket. Requires the `docker` module on your plan.

Compliance: enables OpenSCAP CIS benchmark scanning. Requires the `compliance` module on your plan.

These toggles write the initial `docker_enabled` / `compliance_enabled` state on the host record. The agent picks them up on its first connection and updates `config.yml` accordingly. If you're not sure, leave them off; you can switch them on later from the Host Detail → Integrations tab.

See Enabling Docker Integration.

Click **Next**. PatchMon creates the host in **Pending** state and generates a unique API ID and API key. The key is displayed **only once** (in the command on the next step). If you close the wizard without copying it, you will need to regenerate credentials from the Host Detail page.

Step 3: Copy the Install Command

The wizard now shows a read-only command tailored to:

- The chosen OS (Linux / FreeBSD / Windows).
- Your server's configured URL.
- The new host's API ID and API key.
- Your global TLS setting (`ignore_ssl_self_signed`): if it is on, the Linux command uses `curl -sk` and Windows toggles default to SSL bypass.

Linux / FreeBSD

```
curl -s "https://patchmon.example.com/api/v1/hosts/install" \
  -H "X-API-ID: patchmon_xxxxxxxx" \
  -H "X-API-KEY: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"
| sudo sh
```

On FreeBSD the script drops `sudo` , because root usually runs the command directly.

Windows

On Windows the command is a PowerShell snippet that downloads the installer via `Invoke-WebRequest` and executes it. Two checkboxes adjust it:

- **Self-signed certificate (SSL bypass)**: prepends code that forces TLS 1.2 and disables certificate validation for the download. Use only on lab or internal-CA environments.
- **Use curl instead of Invoke-WebRequest (if download fails)**: switches to `curl.exe` for the download, useful when corporate endpoint-protection tooling breaks `Invoke-WebRequest` .

The Windows command must be run in an **elevated PowerShell** (Run as Administrator).

Copy

Click the **Copy command** button (or the **Copy** icon). The command is placed in your clipboard and the wizard advances to Step 4. If your browser blocks clipboard access, the wizard falls back to a `prompt()` dialog with the command pre-selected; copy from there.

> ***Tip:*** *Store the command somewhere safe only if you need it for later re-use. The API key is not shown again from this screen. If you lose it, regenerate credentials from* ***Host Detail → Deploy Agent → API Credentials → Regenerate****.*

Step 4: Waiting for Connection

After you copy the command, the wizard flips to a progress screen. It polls the server every 2 seconds and walks through four stages:

Stage	Icon	What it means
Waiting for connection	Pulsing Wi-Fi	Host record exists; no agent has connected yet. Run the command now.
Connected	Green tick	The agent has opened a WebSocket to the server. The initial report is still in flight.
Receiving initial report	Animated download	The agent is sending its first system / package inventory.
Done	Green tick	Initial report received. The wizard redirects you to the new Host Detail page.

Run the copied command on the target host. Within a few seconds the status flips to **Connected**, and shortly after to **Done**. At **Done**, the modal closes and the URL changes to `/hosts/<hostId>`.

If you need to see the command again (for example because you pasted it into the wrong terminal), click **View command again** to jump back to Step 3. The command and credentials are preserved.

> ***Note:*** *Closing the wizard before the agent connects does* ***not*** *cancel the host. The host remains in* ***Pending*** *state and you can finish enrolment later from the Host Detail page. However, the* ***plaintext API key*** *is cleared from memory once the modal closes. If enrolment is not complete, open the host, click* ***Deploy Agent****, and regenerate credentials to get a fresh command.*

After Enrolment

Once the agent connects and sends its first report:

- The host moves from **Pending** to **Active**.
- The OS type, OS version, architecture, hostname, IP, kernel, and package list populate automatically.
- The **Connection** column on the Hosts page shows a green **WSS** badge (or **WS** if you are running without TLS).

Packages, repositories, and any enabled integrations (Docker, compliance) start reporting on the configured interval.

Troubleshooting: The Host Doesn't Check In

If the wizard sits on **Waiting for connection** for more than a minute or two, run through the checks below on the target host.

Check the install command actually ran

On Linux / FreeBSD the install script is verbose. Look for:

```
Downloading patchmon-agent-<os>-<arch>...
Installing to /usr/local/bin/patchmon-agent
Writing /etc/patchmon/config.yml
Writing /etc/patchmon/credentials.yml
Starting patchmon-agent service
```

If the script aborted early, re-run it. If `apt-get` fails because of broken packages, open the host again from the Hosts page, click **Deploy Agent**, tick **Force install (bypass broken packages)** on the **Quick Install** tab, and use the regenerated command.

Confirm the service is running

Linux (systemd):

```
sudo systemctl status patchmon-agent
```

Alpine (OpenRC):

```
sudo rc-service patchmon-agent status
```

Windows:

```
Get-Service -Name PatchMonAgent
```

For detailed service management, see Managing the PatchMon Agent.

Test the connection manually

On the host, run the agent's built-in connectivity and credential test:

```
sudo patchmon-agent ping
```

A successful response looks like:

```
API credentials are valid
Connectivity test successful
```

If this fails, the agent log is the next stop:

```
sudo tail -n 50 /etc/patchmon/logs/patchmon-agent.log
```

Common causes:

HTTP 401: the API key on the host does not match the one stored on the server. Usually this means the wizard was closed and the host was re-created, or credentials were rotated. Regenerate credentials from **Host Detail → Deploy Agent**.

TLS / certificate error: the host does not trust the server's TLS certificate. Either install the CA into the host trust store, or set `skip_ssl_verify: true` in `/etc/patchmon/config.yml` (lab only).

Connection refused / timeout: firewall, DNS, or reverse-proxy issue. From the host, `curl -I https://patchmon.example.com` should return an HTTP response.

See Managing the PatchMon Agent for a full diagnostic walkthrough and the `patchmon-agent diagnostics` command.

Nothing wrong on the host: still "Waiting"?

Open the host record regardless: from the Hosts page, click the host's friendly name (even while it's in **Pending**). The page shows a **Deploy Agent** button near the top right. Click it to reopen the install command and the waiting screen, and try again.

If the host has been **Pending** for a long time and you want to start over, delete the host from the Hosts page (trash icon or bulk-select → **Delete**), then add it again from scratch.

Bulk / Automated Enrolment

The Add Host wizard is designed for one host at a time. For automated enrolment:

Proxmox LXC: PatchMon can auto-enrol all containers on a Proxmox host. See Proxmox LXC Auto-Enrollment Guide.

Scripted deployments: Use the Integration API to create hosts, then deploy the install command via configuration management (Ansible, Salt, Chef, cloud-init). See Integration API Documentation.

Related Pages

Managing Host Groups: create groups before (or after) adding hosts.

Host Detail Page: guided tour of the page you land on after enrolment.

Installing the PatchMon Agent: full agent-installer documentation.

Managing the PatchMon Agent: post-install CLI, logs, diagnostics.

Agent Configuration Reference: every `config.yml` field explained.

Chapter 4: Host Detail Page

Overview

The **Host Detail** page is the single-host workbench in PatchMon. Reach it by clicking any host's friendly name from the **Hosts** page, or navigate directly to `/hosts/<hostId>`. Every action, statistic, and tab for a specific host lives here: connection status, package counts, repositories, integrations, patch runs, agent queue, credentials, and (when the relevant modules are enabled) Docker inventory and compliance results.

This page is a guided tour of the layout: what to click and what each tab is for.

Permission required: `can_view_hosts` to open the page. Mutating actions (trigger report, change groups, toggle integrations, delete host, run patches) need `can_manage_hosts`.

Page Header

The top of the page has four main areas:

Identity strip

Friendly name (large heading), editable inline.

Hostname and **IP** underneath, both editable inline (clicking shows a text field; Enter to save, Esc to cancel).

Status pills (four independent indicators, each with a hover tooltip explaining what it means):

WS — WebSocket control channel state. Green when connected, amber while disconnected within the configured grace window, red once the grace window elapses. The grace window is the `host_down` alert threshold (default 30 seconds, see Per-alert-type configuration).

Reporting — agent report freshness. Grey ("Awaiting report") until the agent sends its very first report, then green when the agent has reported within its update interval, amber when overdue but the WebSocket is still connected (agent is alive, just hasn't pushed yet), red ("Stale") when overdue *and* the WebSocket is also disconnected.

Reboot pending — only shown when the host has flagged a pending reboot (e.g. `/var/run/reboot-required`, kernel updates).

Updates — grey "No package data" until the first report arrives, then green "Up to date", amber "Updates pending" (non-security only), red "Security patches required" when one or more security updates are available.

Uptime and **Last updated** relative timestamps.

Suspended machines catch up on their own. A laptop or VM does not run its report timer while it is asleep, so it wakes up with an amber Reporting pill (red, while it was still asleep and its WebSocket was down). When an agent connects and the host has not reported within its update interval, the server asks it for a fresh report within the following minute, so the pill clears without anyone pressing **Fetch Report**. The same applies to a host that was powered off, or cut off by a network outage. Hosts that keep reporting on schedule are unaffected.

If there's a pending patch run awaiting a fresh post-patch report, you'll see an **Awaiting inventory report** chip that links to the run.

Action buttons (top right)

Button	What it does	Requires agent online?
Apply	Appears only when pending config changes (e.g. integration toggles) need to be pushed to the agent.	Yes
Fetch Report	Sends a WebSocket command asking the agent to collect and submit a fresh report now.	Yes
Patch all	Opens the Patching wizard pre-scoped to this host. Hidden on Windows hosts.	Yes
Deploy Agent (key icon)	Opens the Credentials modal with the install command and API credentials.	No
Refresh (circular arrow)	Re-fetches host data from the PatchMon server (UI only).	No
Delete host (trash icon)	Opens a confirmation dialog and then removes the host record.	No

Package statistics cards

Four clickable cards:

Total Installed → opens **Packages** filtered by this host.

Outdated Packages → **Packages** filtered to this host with only those needing updates.

Security Updates → **Packages** filtered to this host with only security updates.

Repos → opens **Repositories** filtered to this host.

Use these as quick jump-offs, with the host pre-selected. The **Packages** page you land on stays scoped to that host: its heading names the host, its own summary cards count only that host, and a **Clear filter** button widens it back out to the whole fleet.

The Tab Strip

Below the cards is a horizontal tab bar. On desktop, all content is inside tabs; on mobile, sections are stacked as cards and tabs are replaced by quick-jump links.

The tab strip is context-aware. Some tabs only appear under certain conditions:

Tab	Always visible?	Notes
Host Info	Yes	Default landing tab.
Network	Yes	
System	Yes	
Agent Activity	Yes	Unified timeline of agent comm cycles (ping, full report, partial report, Docker, compliance) and outbound queue jobs. Replaces the separate Package Reports and Agent Queue tabs from earlier releases.
Notes	Yes	Free-text notes.
Integrations	Yes	Per-host Docker / Compliance toggles.
Reporting	Conditional	Hidden when global alerts are off.
Docker	Conditional	Only when the host has reported a working Docker integration. Gated by the `docker` module; shows a PLUS badge if your plan doesn't have it.
Patching	Yes	Gated by the `patching` module; shows a tier badge if your plan doesn't have it.
Compliance	Conditional	Only when the host has reported the compliance integration. Gated by the `compliance` module.
Terminal	Yes on Linux/FreeBSD	Browser-based SSH. Gated by the `ssh_terminal` module.
RDP	Windows hosts only	Browser-based RDP via Guacamole. Gated by the `rdp` module.

Each tab below is described as what you see and what you can do.

Host Info

Quick reference panel for the host's identity and agent settings. Fields include:

Friendly Name: inline-editable.

IP Address: inline-editable; if the agent has picked a primary interface, this field is read-only and tagged *from eth0* (or whatever the interface is).

Hostname: inline-editable.

Machine ID: read-only unique hardware identifier.

Host Groups: coloured multi-select chips. Add or remove groups; changes save on blur.

Operating System: icon plus OS type and version (detected by the agent).

Agent Version: version of `patchmon-agent` currently reporting.

Agent Auto-update: per-host toggle. If global auto-update is disabled, a yellow warning badge is shown with a tooltip pointing to **Settings → Agent Updates**.

Force Agent Version Upgrade: the **Update Now** button sends an immediate upgrade command via WebSocket. Disabled when the agent is offline.

See Managing the PatchMon Agent for what the agent does when it receives an upgrade command.

Network

Visible when the agent has reported network data. Two sections:

DNS Servers: grid of resolvers the host uses.

Network Interfaces: one card per NIC with:

- Name, type, and UP/DOWN status.
- MAC address, MTU, and link speed / duplex.
- All `inet` and `inet6` addresses with netmask and gateway.
- A **star** icon to mark one interface as the **primary**. PatchMon will use that interface's primary address as the host's IP everywhere in the UI, overriding auto-detection.

Clearing the primary flag re-enables auto-detection.

System

Hardware and OS specifics collected on each report:

Kernel version, **SELinux status**, **architecture**, **package manager**.

CPU model, socket/core counts, frequency.

Memory: total, used, free, swap (formatted in GiB).

Storage / disk layout.

Hardware vendor, product, serial number (where the agent can read it).

SSG (OpenSCAP) version when the compliance integration is enabled.

This tab is read-only. All values come from the agent report.

Agent Activity

Unified timeline of every agent comm cycle for this host. Each row is one of:

Ping: hash-gated check-in (every cycle, even when nothing changed).

Full / Partial: `/hosts/update` with the full inventory or just the sections the server flagged as stale.

Docker / Compliance: integration-specific submissions.

Job: outbound queue jobs the server sent to the agent (fetch report, patch run, integration setup, etc.).

The four queue stat cards (Waiting / Active / Delayed / Failed) sit above the table and reflect in-flight server-to-agent jobs. The auto-refresh interval is 30 seconds.

Each report row also shows section chips: green "Updated" chips for sections the agent shipped fresh data for this cycle, and muted "Skipped" chips for sections the server already had a matching hash for. Use this tab to:

Confirm a host is actively checking in (look for recent `Ping` rows).

Audit when a package, repo, network interface, or hostname change last propagated.

Trace a "my Fetch Report click didn't do anything" complaint by following the job row through `Waiting → Active → Completed` (or `Failed` with the error message).

Retention is governed by the `AGENT_REPORTS_RETENTION_DAYS` environment variable (default 30 days, range 7..365). The daily cleanup sweep at 02:00 deletes anything older. See the operator guide for tuning details.

> *Earlier releases split this view across two tabs (Package Reports and Agent Queue). Bookmarks against the old `?tab=history` and `?tab=queue` query params redirect to `?tab=activity`.*

Notes

A free-text area for operator notes: change windows, ownership, special configuration, support contacts. Click into the text area to edit, then **Save**.

Notes are visible to any user with `can_view_hosts` on this PatchMon deployment.

Integrations

Per-host toggles and setup status for optional agent integrations. Two primary panels:

Docker

Toggle: enables Docker discovery for this host. When off, no containers / images / volumes / networks are collected.

The change is staged as a **pending configuration**; a yellow banner appears at the top of the tab until the change is applied.

Click **Apply** in the page header to push the change to the agent via WebSocket. The agent updates its `config.yml` and reports back.

See Enabling Docker Integration for prerequisites and troubleshooting.

Compliance Scanning

A three-state selector: **Disabled**, **On-Demand**, **Enabled**.

Disabled: no scans.

On-Demand: only run when triggered from the UI; not included in scheduled reports.

Enabled: scans run on the agent's normal reporting interval.

Setup status indicator showing Installing / Ready / Partial / Error, with per-component status (OpenSCAP, Docker Bench).

Scanner Types section: individual toggles for **OpenSCAP (CIS Benchmarks)** and **Docker Bench** (the Docker Bench toggle is disabled if the Docker integration is off on this host).

Changes require the agent to be connected via WebSocket.

Refresh Status

The **Refresh Status** button at the top right of the tab asks the agent to report its current integration readiness immediately. Useful after installing OpenSCAP manually on the host.

Reporting

Host-scoped overrides for alerting. The tab is hidden when global alerts are disabled in **Settings**.

Primary feature: **Host Agent Down Alerts** with three states:

Inherit from global settings (default).

Enabled: always create alerts when this host's agent goes down, regardless of global defaults.

Disabled: never create alerts for this host even if the global setting is on.

Use the Disabled override for hosts that are expected to be intermittent (dev laptops, ephemeral CI runners) so they don't spam your alert channels.

Docker (conditional)

Appears only when the host has actually reported a working Docker integration on at least one report. The tab is a compact per-host version of the fleet-wide Docker Inventory Tour.

Sub-tabs: **Stacks**, **Containers**, **Images**, **Volumes**, **Networks**. Each sub-tab shows counts in a badge next to its name.

Requires the `docker` module to be enabled on your plan. Plans without the module show a tier badge on the tab and an upgrade prompt inside.

Patching

Per-host patch run history and trigger point. Shows:

- A filterable, sortable, paginated list of runs for this host.
- Status chips (queued, running, completed, failed, approval pending).
- Inline output for completed runs.
- The **Patch all** button at the top of the Host Detail page is the quick way to start a new run scoped to this host.

Requires the `patching` module. The tab displays a tier badge when unavailable.

Compliance (conditional)

Appears when the compliance integration has been set up and at least one scan has run. Shows the latest benchmark results, failed rules with remediation guidance, and history. The agent installs OpenSCAP automatically when the integration is enabled; this tab surfaces that install's progress plus scan output.

Requires the `compliance` module.

Terminal

Browser-based SSH session to the host, proxied through the agent. No inbound port is needed on the host; the connection is routed over the existing agent WebSocket. Typical workflow:

- Open the **Terminal** tab.
- PatchMon fetches a short-lived SSH ticket and opens a WebSocket to the agent.
- The agent connects to localhost SSH (or the configured target) and relays the session.
- AI-assisted analysis is available within the terminal UI.

Gated by the `ssh_terminal` module. The agent-side `ssh-proxy-enabled` switch in `config.yml` must also be on. This is **not** a UI toggle, due to its security implications. See the SSH section of Agent Configuration Reference.

RDP (Windows only)

Browser-based RDP session using a Guacamole (`guacd`) gateway on the PatchMon server. Like the Terminal tab, it uses the agent to reach the host without requiring inbound firewall rules.

Visible only on Windows hosts, gated by the `rdp` module, and requires the agent-side `rdp-proxy-enabled` switch to be on in `config.yml`.

Credentials Modal (Deploy Agent)

Click **Deploy Agent** (key icon) in the page header to open the modal. Two tabs:

Quick Install

A copy-ready install command for this host, pre-populated with its API ID and API key. Options:

Force install (bypass broken packages): Linux / FreeBSD only.

Self-signed certificate (SSL bypass): Windows only.

Use curl instead of Invoke-WebRequest: Windows only, workaround for hosts where PowerShell's downloader is blocked.

Click **Copy**. You're automatically moved to the **Waiting for Connection** screen, which polls until the agent connects and sends its first report. This is identical to step 4 of the Add Host wizard.

API Credentials

API ID: copyable.

API Key: obscured by default. The plaintext key is **only** available if it was just created or regenerated; otherwise the field shows *hashed – not usable*.

Regenerate: creates a new API ID and key and invalidates the old ones. Use this if credentials have been lost or compromised. The agent on the host will need to be reconfigured or the install command re-run.

Common Actions: Where to Click

Quick reference for the most-asked "how do I..." questions:

Goal	Where
Trigger an immediate report	Page header → **Fetch Report**
Force the agent to self-update	**Host Info** tab → **Update Now**
Open a shell in the browser	**Terminal** tab
See what the agent is doing right now	**Agent Activity** tab
Change which host groups the host is in	**Host Info** tab → **Host Groups** field (or the Hosts table inline edit)
Turn Docker monitoring on / off	**Integrations** tab → **Docker** toggle, then **Apply** in header
Run CIS scans	**Integrations** tab → **Compliance** selector → **Enabled** or **On-Demand**
Re-copy the install command	**Deploy Agent** (key icon) → **Quick Install**
Rotate the API key	**Deploy Agent** → **API Credentials** → **Regenerate**
Patch this single host	Page header → **Patch all** (opens wizard scoped to host)
Permanently remove the host	Page header → trash icon

Mobile Layout

On smaller screens, the tab strip is replaced with stacked cards (**Host Information**, **Network**, **System**, **Agent Activity**, and so on). The action buttons collapse into an icon row. Some dense sections (for example the Integrations mode selector) show a **Manage in Integrations tab** shortcut.

All data shown on mobile is the same as on desktop; only the layout changes.

Related Pages

Adding a Host: how a host gets here in the first place.

Managing Host Groups: editing group membership.

Managing the PatchMon Agent: CLI equivalents of the actions on this page.

Enabling Docker Integration: what turning on the Docker toggle actually does.

Docker Inventory Tour: the fleet-wide view of the data surfaced on the Docker tab.

Chapter 5: Managing Host Groups

Overview

Host groups are the primary way to organise your fleet in PatchMon. A group is a named bucket with a colour and an optional description. Each host can belong to any number of groups. Groups appear as a column and filter on the Hosts page, as a selector on the Patching page, as a scope for patch policies, and as a filter in the Integration API.

This page covers creating, editing, and deleting groups, assigning hosts to them, and how groups feed into other parts of PatchMon.

Permission required: `can_manage_settings` to create / edit / delete groups (the page sits under **Settings**); `can_manage_hosts` to change which groups a host belongs to.

Where to Find Host Groups

There are two places to manage groups in the UI, and both edit the same data:

Settings → Host Groups: full management page with a table view, create / edit / delete actions, and host counts.

Options page (`/options`): the same host-groups component, rendered inline for operators who have `can_manage_hosts` but not `can_manage_settings` .

You can get to host counts and host assignments from either place, but the **Settings → Host Groups** route is the canonical one and is the one this page describes.

Creating a Group

In the left navigation, open **Settings**.

Click **Host Groups** in the settings sub-menu. The page opens with a table listing existing groups and a **Create Group** button in the top right.

Click **Create Group**. A modal titled **Create Host Group** opens.

The form has three fields:

Field	Required?	Notes
Name	Yes	Short identifier such as `Production` , `Web servers` , `DB tier` . Shown in the UI and in API output.
Description	No	Free-text note shown in tooltips and on the group card. Use it for scope or ownership information.
Color	Yes	A hex value used for the coloured dot next to the group name. Picker + text input; defaults to `#3B82F6` (blue).

Click **Create Group** to save. The new group appears immediately in the table with a **0 hosts** count.

Assigning Hosts to a Group

Groups are assigned on the host, not on the group. You have three paths:

During enrolment

On step 2 of the **Add Host** wizard (**Host details**) tick each group the host should belong to. See Adding a Host.

Inline from the Hosts page

1. Open **Hosts** from the left navigation.
2. Find the row for the host you want to change.
3. Click the value in the **Group** column. It becomes an editable multi-select with coloured group chips.
4. Tick / untick groups, then click away (or press Enter) to save.

Bulk assign

1. Open **Hosts**.
2. Select multiple rows using the checkboxes in the **Select** column.
3. A toolbar appears above the table with **Fetch Reports**, **Assign to Group**, and **Delete**.
4. Click **Assign to Group**. A modal lists every group with a checkbox.
5. Tick one or more groups and click **Assign to Groups**. All selected hosts are updated in a single call.

> ***Note on bulk behaviour:*** *the bulk assign modal* ***sets*** *the selected groups on each host. It does not add to existing memberships. If you want to add a group without removing others, use the inline edit on the Hosts table or the Host Groups selector on the Host Detail page.*

From the Host Detail page

Open any host (**Hosts** → click friendly name). On the **Host Info** tab, the **Host Groups** field is a multi-select identical to the one on the Hosts list. Tick / untick and click away to save.

Editing a Group

From **Settings → Host Groups**:

1. Click the pencil icon in the **Actions** column of the group you want to change.
2. A modal titled **Edit Host Group** opens with the current **Name**, **Description**, and **Color** pre-filled.

Change whatever you need and click **Update Group**.

Editing a group's name or colour updates it **everywhere**: the Hosts table, the host's detail page, the Patching targets, and the API all pick up the new value immediately.

Deleting a Group

Deletion is restricted to prevent you from orphaning hosts by accident.

From **Settings → Host Groups**, click the trash icon next to a group.

The **Delete Host Group** modal opens.

If the group **has no hosts**, the **Delete Group** button is enabled. Confirm to delete.

If the group **has one or more hosts**, the modal shows a yellow warning with the list of hosts in the group, and the **Delete Group** button stays disabled.

What happens to hosts when the group is deleted?

You cannot delete a group that still contains hosts. The UI prevents it and the server returns an error. To delete a populated group, first move or remove the hosts:

Move each host out of the group: open the host, untick the group in the **Host Groups** field, and save. The host stays in PatchMon and keeps any other group memberships.

Or use bulk-assign from the Hosts page to reassign many hosts at once.

Once the group is empty, return to **Settings → Host Groups** and delete it.

> ***Note:*** *Deleting a group does **not** delete hosts. Hosts that were only in that group become **ungrouped** and still appear in the Hosts list. They can be reassigned to another group later.*

Filtering by Group

On the Hosts page

Click **Filters** in the Hosts toolbar to reveal the filter panel.

Open the **Host Group** dropdown. It lists every group plus an **Ungrouped** option.

Pick a group. The table reloads showing only hosts in that group.

Use **Clear Filters** to reset.

You can also deep-link by visiting `/hosts?group=<groupId>`; group-clicks from other parts of the UI (for example the Host Groups page's host count badge) do exactly this.

Grouping the table

Above the filter panel, the **No Grouping** dropdown lets you group rows by **Group**, **Status**, or **OS**. Picking **By Group** splits the Hosts table into sections, one per group, each with a count header. Hosts in multiple groups appear under each of their groups. This is a visual grouping, not a filter.

Hiding stale hosts

The **Hide Stale** toggle on the Hosts toolbar can be combined with a group filter to narrow a view down to "active hosts in the production group", for example.

How Groups Feed Other Features

Groups are a **selector** across the product. Wherever a workflow asks "which hosts?" you can usually answer "this group".

Patching

- When you build a patch run, the target picker offers **Host** or **Host group** as the target type.
- Choosing a host group expands the target to every current member at the time the run is queued.
- Patch policies (recurring runs) can use a host group as the primary selector, so adding a new host to the group automatically includes it in the next scheduled run.

For the full patching flow, see the Patching chapter.

Integration API

The Integration API exposes host-group membership as a filter on host-related endpoints. Common patterns:

- Listing hosts in a specific group.
- Fetching package status rolled up by group for an external dashboard.
- Triggering bulk reports only for hosts in `Production`.

See Integration API Documentation for the exact endpoints and parameters.

Alerts and Reporting

Some alert channels and scheduled reports support scoping by host group so that, for example, the platform team only receives notifications for `Production` hosts while the dev team owns `Staging`. See the Alerts & Notifications chapter for specifics.

Dashboards

Dashboard cards and the Hosts summary counts are fleet-wide by default. Individual host-centric views (Host Detail) show the groups a host belongs to as coloured chips, and clicking a chip deep-links back to a group-filtered Hosts list.

Good Practice

A few patterns worth knowing:

Keep group names short. They show up in chips, tables, and filter dropdowns with limited width.

One group per purpose. Environment (`Production` , `Staging`), role (`Web` , `DB` , `Cache`), location (`EU-West` , `US-East`), or owner (`Platform` , `Billing`) are all reasonable axes. Combine them by giving a host multiple memberships instead of creating compound groups like `Prod-Web-EU` .

Use colour consistently. For example: red for production, yellow for staging, green for development. Consistent colours make the chips in the Hosts table and on dashboards readable at a glance.

Empty is fine. Groups with zero hosts are valid; they are often created ahead of time for patch-policy planning. The UI just won't let you delete a group that still has hosts.

Related Pages

Adding a Host: assign groups during enrolment.

Host Detail Page: edit a single host's group memberships.

Integration API Documentation: query hosts by group programmatically.

Chapter 6: Package Inventory

Overview

The **Packages** page is PatchMon's fleet-wide package inventory. It aggregates every package reported by every agent across your fleet into a single searchable list, with one row per unique package name, showing how many hosts have it installed, how many need updates, whether any of those updates are security-flagged, and which repositories supply the latest version.

This page walks through the Packages list, the filters, the per-package detail page, and how the inventory ties into the **Outdated Packages** card on the Dashboard.

Permission required: `can_view_packages` to view packages. `can_manage_hosts` to trigger patch runs from this page.

Getting There

Click **Packages** in the left navigation. The page shows a summary row, a filter toolbar, and a paginated table.

You can also deep-link:

`/packages?host=<hostId>` : pre-filter to a specific host.

`/packages?filter=outdated` : only packages with at least one pending update.

`/packages?filter=security` or `/packages?filter=security-updates` : only packages with security updates available.

`/packages?filter=regular` : only packages with non-security updates.

The Dashboard's **Outdated Packages** card and the Host Detail cards use these query-string shortcuts.

Summary Cards

The heading tells you what the cards are counting, and the cards always match it.

Arriving without a host filter, the heading reads **Packages on all Hosts** and five cards summarise your whole fleet:

Card	Meaning	Click behaviour
Packages	Unique package names across the fleet	–
Installations	Sum of per-host installs across all listed packages	–
Outdated Packages	Packages with at least one host needing an update	Filters to **Packages Needing Updates**
Security Packages	Packages with at least one host needing a security update	Filters to **Security Updates Only**
Outdated Hosts	Distinct hosts that appear in the "needs update" side of those packages	Jumps to **Hosts** filtered to hosts needing updates

Arriving with a host filter, for example by clicking **Outdated Packages** on a Host Detail page, the heading reads **Packages for Host** and carries a **Clear filter** button that returns you to the fleet view. Three cards are shown, and every figure counts that host alone:

Card	Meaning	Click behaviour
Packages	Packages installed on this host	–
Outdated Packages	Packages on this host needing an update	Filters to this host's packages needing updates
Security Packages	Packages on this host needing a security update	Filters to this host's security updates

Installations and **Outdated Hosts** are hidden when a host filter is active. Both are fleet measures: on a single host each package is installed exactly once, so Installations would only restate Packages, and Outdated Hosts is not a per-host figure at all.

The **Outdated Packages** and **Security Packages** cards keep whatever host filter is active, so clicking them narrows the list rather than resetting your scope. The **Outdated Hosts** card jumps you out to the Hosts page rather than filtering in-place.

> ***Note:*** *the fleet-wide figures come from a periodic statistics snapshot rather than a live query, so shortly after an agent reports they can trail the table by a few minutes. The host-scoped figures are read live and always agree with the list beneath them.*

The Filter Toolbar

Above the table:

Search: free-text search across package names (debounced, matched server-side).

Category: package category as reported by the underlying package manager (only populated for managers that expose categories, primarily apt/dpkg).

Update Status: the main filter. Four options:

All Packages

Packages Needing Updates

Security Updates Only

Regular Updates Only

Host: limits the list to packages present on a single host. When set, the page shows only that host's packages and unlocks the **Patch all** button to run a patch on just that host.

Columns: customise which columns are visible and in which order.

> ***Note:*** *The toolbar filters by a single **host**, not by **host group**. To review packages across a group, open the **Hosts** page, filter by group, bulk-select the hosts, and review their packages via the individual host links or patch wizards. Group-wide patching is driven from the Patching page, not from here.*

Reading the Table

Default columns:

Column	Content
(select)	Checkbox. Tick to include the package in a multi-host patch run.
Package	Package name with a `Package` icon. Click the name to open the Package Detail page. An `Info` bubble appears when the package has a description; click it to see the description in a modal.
Installed On	Number of hosts with the package. When some (but not all) of those hosts need an update, the column shows `N/M hosts` (for example `3/12 hosts` means 3 of 12 are outdated). Clicking the cell opens the relevant set of hosts on the Hosts page.
Status	One of three badges: **Up to Date** (green), **Update Available** (amber), **Security Update Available** (red, with a shield icon).
Latest Version	The newest version PatchMon has seen reported across all hosts for this package.
Source Repos	Repo chips. Each chip links to that repository. If more than three sources are reported, the overflow is shown as `+N`.

The **Columns** button lets you hide any column except the select checkbox, drag to reorder, and reset to default. The column layout is persisted locally in your browser.

How "Installed On" is calculated

The count includes every host PatchMon has seen reporting the package, regardless of version. The "needs updates" part of `N/M hosts` is hosts whose currently installed version is older than the latest version available from their configured repositories.

Security vs Regular updates

A package is "security" when at least one host sees a security-flagged update for it (typically because the update is pulled from a distribution's security channel such as `*-security` on Debian/Ubuntu or a vendor advisory on RHEL). Regular updates are non-security package upgrades. The **Status** column surfaces whichever priority is higher.

Sorting and Pagination

Click any sortable column header to toggle sort direction.

The status sort uses a priority: **Security Update Available** first, then **Update Available**, then **Up to Date**, so ascending order puts the highest-risk packages at the top.

The page size selector (bottom of the table) supports 25, 50, 100, or 200 rows. Your choice is remembered per browser.

Clicking into a Package

Clicking a package name (or using the filter chip from a host's detail page) opens `/packages/<id>`, the **Package Detail** page. Two tabs:

Hosts tab (default)

Lists every host where the package is installed. Key elements:

Top summary cards: **Updates Needed**, **Latest Version**, **Updated** (when PatchMon last saw a report for the package), **Hosts with Package**, **Up to Date**.

A **Source Repositories** strip showing every repo across the fleet that supplies this package (click through to a repo detail).

Description panel: the package description as reported by the host's package manager.

Only pending filter: ticked by default; untick to see every host, including ones already up to date.

Search: filters the host list.

Per-row actions: for Linux/FreeBSD hosts, you can trigger a targeted patch of this one package. Windows hosts are marked as managed via Windows Update / WinGet.

Select multiple rows and use **Patch selected** to run the same upgrade across many hosts in one patch run.

Activity tab

Recent patch runs in which this specific package was upgraded, with timestamps, target hosts, and outcomes. Useful for "when was this CVE closed across the fleet?" audits.

Bulk Patch from the Packages Page

Two flows produce a patch run from the Packages page:

Patch selected packages across chosen hosts

Tick the checkbox on each package to include. The header shows `N selected`.

Click **Patch selected (N)** (top right).

The **Patch wizard** opens in multi-host mode, discovers which hosts have the selected packages installed and need updates, and lets you pick which to include.

If the **Host** filter is already set to a single host, the wizard locks to that host to avoid offering unrelated hosts.

Patch all on a single host

Set the **Host** filter to one host.

Click **Patch all** (top right, only appears when a single non-Windows host is filtered).

Confirm in the wizard to upgrade every outdated package on that host.

Both flows route you into the Patching chapter. See the patch-run pages there for what happens next.

The Outdated Packages Dashboard Card

The Dashboard shows a **Outdated Packages** card near the top of the Cards layout (the actual position depends on your personal dashboard customisation). The number shown is the fleet-wide count of packages with at least one host needing an update, which is the same figure as the **Outdated Packages** card on the Packages page when that page is not filtered to a single host.

Clicking the Dashboard card navigates to `/packages?filter=outdated`, which:

- Sets the **Update Status** filter to **Packages Needing Updates**.
- Clears the **Category** filter.
- Leaves other filters at their defaults.

From there, you can drill into individual packages, set up a patch run, or narrow to a specific host.

Tips

- **Live list, periodic backing data.** Rows are fetched from the PatchMon server and reflect the most recent reports submitted by each host. To refresh a host's data immediately, open the host's detail page and click **Fetch Report**. The Packages list picks up the new state on its next refetch (or click **Refresh** on the Packages page).
- **Drilling down into patch state.** The **Hosts** page is the better starting point when you care about which *hosts* are behind on updates. Use the Packages page when you care about which *packages* expose the fleet.
- **Windows hosts.** Package reporting works for Windows (via `winget`, `chocolatey`, and MSI inventory), but the **Patch all** / **Patch selected** actions do not target Windows. Patching on Windows is managed via Windows Update or WinGet directly on the host.

Related Pages

- Host Detail Page: per-host package summary and fetch-report actions.
- Repository Tracking: see which repositories each package comes from.
- Managing the PatchMon Agent: how the agent collects package data.
- Integration API Documentation: fetch the same inventory programmatically.

Chapter 7: Repository Tracking

Overview

Every Linux host configures one or more **package repositories**: `sources.list` entries on Debian/Ubuntu, `.repo` files under `/etc/yum.repos.d/` on RHEL-family, `repositories` on Alpine, `pacman.conf` entries on Arch, and `pkg` sources on FreeBSD. PatchMon's agent inventories those repositories on every report and sends them up alongside the package list. The server aggregates the results into a single fleet-wide view on the **Repositories** page.

This page walks through the Repositories list and detail view, the filters, how security is determined, and how repository data is kept current.

Permission required: `can_view_hosts` to read repositories; `can_manage_hosts` to edit or delete repository records.

What a Repository Entry Represents

A repository record in PatchMon corresponds to a package source as reported by a host's package manager:

Package manager	Repository source
apt (Debian, Ubuntu)	Each entry in `/etc/apt/sources.list` and `/etc/apt/sources.list.d/*.list` or `*.sources`
yum / dnf (CentOS, RHEL, Rocky, Alma, Fedora)	Each enabled entry in `/etc/yum.repos.d/*.repo`
apk (Alpine)	Each line in `/etc/apk/repositories`
pacman (Arch)	Each `[repo]` section in `/etc/pacman.conf`
pkg (FreeBSD)	Each configured pkg repository

Multiple hosts configured with the same URL are collapsed into a **single repository record** in the fleet view, so you can see at a glance which hosts pull from a given source. The per-host relationship is tracked separately so you can drill in and see exactly where a repo is in use.

Key fields on a repository entry:

Name: a human-friendly identifier (often the repo's `Label` or `id`).
URL: the base URL the package manager fetches from.
Distribution / release codename: e.g. `jammy`, `el9`, `3.19`.
Is Secure: `true` when the URL starts with `https://`, `false` for plaintext `http://`.
Is Active: whether the repo is currently enabled on at least one host.
Host count: number of hosts currently configured with this repo.

Getting to the Repositories Page

Click **Repositories** in the left navigation. You can also deep-link:

- `/repositories?host=<hostId>`: pre-filter to a single host's repositories.

  The **Repos** card on the Host Detail page and the source-repo chips on the Packages page both use these query-string shortcuts.

Summary Cards

Four cards at the top:

Card	Meaning
Total Repositories	Unique repositories across the fleet
Active Repositories	Repositories currently enabled on at least one host
Secure (HTTPS)	Count of repositories whose URL uses HTTPS
Security Score	`secure ÷ total` as a percentage

A low Security Score is a quick signal that you still have HTTP-only repositories in the fleet, which is a good target for remediation.

Filter Toolbar

- **Search**: free-text search across the repository name and URL (debounced, matched server-side).
- **Host filter indicator**: when `?host=<id>` is set, a pill shows *Filtered by:* with an `X` to clear.
- **Security**: **All Security Types** / **HTTPS Only** / **HTTP Only**.
- **Status**: **All Statuses** / **Active Only** / **Inactive Only**.
- **Columns**: customise visible columns and their order.

Reading the Table

Default columns:

Column	Content
Repository	Name, with a Database icon. Click to open the detail page.
URL	Full URL. For Debian-family repos, the `deb-` / `deb-src-` prefix is stripped from display names for readability.
Distribution	Distribution / codename / release.
Security	**Secure** (HTTPS) with a lock icon, or **Insecure** (HTTP) with an open-lock icon.
Status	Active or Inactive.
Hosts	Count of hosts currently configured with this repo. Click to filter the Hosts page by hosts using this repo.
Actions	Delete icon (requires `can_manage_hosts`).

Column visibility and order are persisted per browser.

Clicking into a Repository

Clicking a repository opens `/repositories/<id>`, the **Repository Detail** page. It has three main sections stacked vertically:

Repository Details

Name, **Description**, **URL**, **Distribution**, **Is Active**, **Priority**.

Inline edit toggle (pencil icon, requires `can_manage_hosts`) lets you change the friendly name, description, active flag, and priority.

Delete repository button opens a confirmation dialog.

Top-right summary chips: secure / insecure, active / inactive, last updated.

Editing or deleting a repository from this screen affects the **PatchMon record**, not the underlying host configuration. The next time an agent reports, PatchMon will reconcile with what's actually on the host. If the repo is still configured on any host, it will reappear. Delete is most useful for stale records where no host actively uses the repo.

Hosts Using This Repository

A searchable, paginated list of every host that has this repository configured. Each row shows:

Friendly name (link to the Host Detail page).

Hostname and IP.

OS and version icon.

When the host last reported the repository.

Any host-specific settings (priority override, enabled flag).

Use this view to answer "who's still pulling from this old mirror?" questions.

Packages from this Repository

A searchable, paginated list of every package PatchMon has seen delivered through this repository. Each row shows:

- Package name (link to the Package Detail page).
- Latest version.
- Status badge: **Up to Date**, **Update Available**, or **Security Update**.
- Source repo chip.

This is the easy way to audit "which packages on my fleet come from this third-party repository?"

How Repositories Are Kept Up-to-Date

Agents collect their repository configuration on every report cycle:

- The agent runs package-manager introspection (`apt-cache policy` , `dnf repolist` , etc.).
- The result is serialised and sent alongside the package inventory and system info.
- The PatchMon server upserts repository records and updates the per-host link table:
  - New repositories appear.
  - Removed repositories are marked inactive (and retained as records, so historical package activity can still reference them).
  - URL or distribution changes are reconciled. If you change a URL in `/etc/apt/sources.list` , the next report updates it.

Because updates are **report-driven**, the Repositories page reflects the last known state. To force an immediate refresh for one host, open the host and click **Fetch Report**.

Security Filter in Practice

The **HTTPS Only** / **HTTP Only** filter is the quickest audit tool for enforcing secure package sources:

- Set **Security** to **HTTP Only**.
- The list now shows every plaintext repository in the fleet.
- For each, click into the repo and use the **Hosts Using This Repository** list to see who needs reconfiguring.
- Fix on the host (swap URL to HTTPS in the relevant `.list` / `.repo` / `apk repositories` file, update the distribution's certificate stores if needed), then run **Fetch Report** on the host.
- The next report will move the host off the insecure record and onto the HTTPS one.

Some legitimate setups (for example, local intranet mirrors, or signed-but-insecure-transport repos such as classic Debian archives protected purely by GPG) are unavoidably HTTP. Use repository descriptions (via the edit dialog) to flag "approved HTTP" entries so future reviewers know they were considered.

Deleting a Repository Record

From the Repositories table or detail page, operators with `can_manage_hosts` can delete a record. The confirmation dialog lists the impact:

- The repository record is removed from PatchMon.
- Per-host links to that record are removed.
- The underlying host configuration is **not** changed; no file on the host is modified.

Because the agent re-reports every cycle, a delete is only permanent if no host actually has the repo configured any more. This is why the **Inactive Only** status filter is helpful: records with zero hosts are safe to tidy up, while records still tied to hosts will reappear after the next report.

Related Pages

- Package Inventory: browse packages, and use the repository chips there to jump to the repo detail.
- Host Detail Page: per-host repositories surface through the **Repos** summary card.
- Managing the PatchMon Agent: how the agent collects repository data on each report.
- Integration API Documentation: repositories are exposed as API objects for external tooling and compliance reporting.

Chapter 8: Patching Overview

What Patching Is

Patching is the PatchMon feature that lets you deploy package and security updates to your Linux and FreeBSD hosts on demand or on a schedule, with validation, approval, stop, retry, and live log streaming over WebSocket. You drive it from the **Patching** page in the web UI, or from the **Patching** tab on any Host Detail page.

Patching in 2.0 is a first-class module rather than a side-feature. Runs are persisted, queued through Redis/asynq, executed by the agent on the host, and streamed back to the browser live.

Module Gate

Both halves of the feature are gated by a capability module. If the module is not enabled on your plan, the relevant UI appears locked with an "Upgrade required" placeholder and the corresponding API routes reject the request.

UI area	Required module
Patching dashboard, Runs & History, trigger a patch run, approve, stop, retry	`patching`
Policies tab, policy assignments, exclusions, scheduled runs	`patching_policies`

Fleet-wide the Patching page is hidden from the sidebar when `patching` is not enabled. The Policies tab inside the page is shown with a tier badge when `patching` is enabled but `patching_policies` is not.

Who Can Use It

Patching uses two RBAC permissions on top of the module gate:

Action	Permission	Route pattern
View dashboard, list runs, open a run, watch the live stream	`can_view_hosts`	`GET /patching/*`
Trigger a run, approve, retry validation, stop, delete a run	`can_manage_patching`	`POST /patching/trigger`, `POST /patching/runs/{id}/approve`, etc.
View policies	`can_view_hosts`	`GET /patching/policies`
Create, edit, delete policies and policy assignments / exclusions	`can_manage_patching`	`POST/PUT/DELETE /patching/policies/*`

If your role has `can_view_hosts` but not `can_manage_patching`, you will see the Patching page read-only. The action buttons (Patch all, Approve, Stop, Delete) either do not appear or return a 403.

The Three Core Concepts

Everything in the Patching module revolves around three concepts:

1. Patch Run

A **patch run** is one unit of patching work against a single host. Every time you click "Patch all" on a host, approve a submission, or retry a validation, a run row is created in the database.

Each run has:

A **type**: `patch_all` (install all available package updates) or `patch_package` (install a specific package or set of packages).

A **dry-run flag**: if `dry_run=true`, the agent reports what *would* change without actually installing anything. This only applies to `patch_package`; `patch_all` cannot be dry-run because the agent's bulk upgrade path does not support `--dry-run` faithfully.

A **status** that moves through the run lifecycle (see below).

A persisted **shell output** that captures every line of stdout/stderr produced by the package manager. Live subscribers get it streamed over WebSocket; everyone else gets the full blob on completion.

Optional **policy metadata**: the effective patch policy is snapshotted onto the run at trigger time, so you can see in run detail "which policy was in effect when this was queued".

Run statuses

The server moves a run through these statuses, visible as badges in the Runs & History table and in the run detail header:

Status	Meaning
`queued`	The execution task has been enqueued on the asynq queue, waiting for the worker to pick it up.
`pending_validation`	A dry-run was queued for validation but has not completed yet (the host may be offline).
`validated`	The dry-run finished successfully; the run is waiting for an operator to approve it.
`pending_approval`	A patch run was submitted for approval without a dry-run (e.g. a `patch_all` that cannot be dry-run). An approver needs to sign off before the run is queued.
`approved`	The original validation run after someone approved it. A new execution run (linked by `validation_run_id`) is created alongside this row and queued.
`scheduled`	The run has been accepted but is waiting for its `run_at` timestamp (delayed or fixed-time policy).
`running`	The agent is currently executing the package manager command on the host. This is the status that opens the live WebSocket stream.
`completed`	The run finished successfully. The persisted `shell_output` is now authoritative.
`dry_run_completed`	A dry-run finished successfully (terminal state for dry-runs that aren't turned into a real run).
`failed`	The run finished with a non-zero exit status or the host reported an error.
`cancelled`	The run was stopped by an operator clicking **Stop Run** (or deleted before execution). The cancel is applied authoritatively in the database first; if the agent is connected the server also sends a courtesy `patch_run_stop` so the running subprocess is interrupted. With this ordering, an offline or unresponsive agent can't leave the row stuck in `running`.
`timed_out`	The periodic patch-run cleanup found this run still in `running` state past the configured stall timeout (`PATCH_RUN_STALL_TIMEOUT_MIN`, default 30 minutes) and marked it as timed out. The cleanup sweep runs every 10 minutes.
`agent_disconnected`	The agent's WebSocket dropped while this run was `running`. The server marks every in-flight run for that host as `agent_disconnected` so the row doesn't sit at `running` indefinitely. If the agent reconnects and posts a late `completed` / `failed` / `cancelled` for the same run, the server will update the row to that final state.

2. Patch Policy

A **patch policy** controls *when* an approved patch run actually fires. Policies are optional; a host with no policy attached gets the implicit "Default" policy, which runs patches immediately on trigger.

Each policy has a **delay type**:

- **Immediate**: run as soon as the task is dequeued.
- **Delayed**: wait N minutes after the trigger before running (useful for "give me 30 minutes to change my mind").
- **Fixed time**: run at a specific wall-clock time (`HH:MM`) interpreted as local time in the **organization timezone** (Settings → General → Timezone). Used for maintenance windows.

Policies are assigned to **hosts** or **host groups**. You can also add per-host **exclusions** to carve specific hosts out of a group-assigned policy. See Patch Policies and Scheduling for full details.

3. Dry-Run / Validation

A **dry-run** (also called a validation run) asks the agent to simulate the package installation without applying it. It exists to catch problems *before* you touch the host:

- For `apt-get`, the agent runs `apt-get -s install <packages>` and parses the simulation output.
- For `dnf` / `yum`, the agent runs the planning step that reports "what would be installed" without committing.
- For `pkg` on FreeBSD, the agent runs `pkg upgrade -n` or the equivalent install-no-run.
- For `pacman`, the agent uses `pacman -S -p` / `pacman -Syu -p` (print-only) as the validation step.

When the dry-run completes the run transitions to `validated` and shows you the list of packages that *would* be installed, including dependencies that were pulled in. If more packages would be installed than you originally asked for, the UI badges the run with **Extra deps** and surfaces the full list in the run detail "Packages affected" panel so you can review before approving.

Approval is the step that turns a validated dry-run into a real patching run. On approval:

- The validation run is marked `approved` (terminal) and preserved with its output for audit.
- A **new** patch run is created with `dry_run=false`, linked to the validation via `validation_run_id`.
- The new run is enqueued against the effective policy (so "Approve & Patch" at 14:00 on a host with a `03:00` fixed-time policy produces a `scheduled` run, not an immediate one).
- You can override the policy at approval time by picking **Immediate** in the approve wizard, which bypasses the delay.

Multi-OS Coverage

The agent chooses the patching back-end by detecting the host's package manager. Linux and FreeBSD patching are fully supported; Windows patching has its own path.

Package manager	OSes	Supported for patching
`apt-get`	Debian, Ubuntu, Raspbian	Yes
`dnf`	RHEL 8+, Rocky, AlmaLinux, Fedora	Yes
`yum`	RHEL 7, CentOS 7	Yes
`pacman`	Arch Linux, Manjaro	Yes
`pkg`	FreeBSD 13+ (plus `freebsd-update` for the base system on `patch_all`)	Yes
`apk`	Alpine Linux	**No.** The agent reports `apk` inventory but rejects patch runs with `package manager "apk" not supported for patching (apt, dnf, yum, pkg, pacman required)`. Alpine hosts are visible in PatchMon and get compliance scans, but patch runs on them fail on the agent side.
`zypper`	openSUSE Leap, openSUSE Tumbleweed, SLES	**No, and no inventory either. Coming soon.** The installer detects zypper and completes, and the agent detects the OS correctly, but the agent has no zypper backend. Package collection therefore fails with `unsupported package manager: unknown` and the host never completes its first report, so it sits at "Waiting for initial system report". Follow and vote for zypper support at feedback.patchmon.net (https://feedback.patchmon.net/b/feature-requests/posts/post_01kyza53c0fzst214afbr1qn9a).
Windows Update Agent (WUA) + WinGet	Windows 10/11, Server 2019/2022/2025	Yes (separate path)

> ***Note:*** *The 2.0 release notes describe Linux patching generally. If you need to patch Alpine hosts, track the package manager roadmap or use your existing Alpine tooling until* `apk` *support lands.*

Windows patching

When the agent detects it is running on Windows, patch runs are handled by the WUA + WinGet path rather than the Linux package-manager path:

- **Patch all** installs every Windows Update currently marked `approved` for that host by the server, plus runs `winget upgrade --all` for WinGet-managed applications.
- **Patch package** routes by name: strings that look like a `KB...` / GUID update are sent via WUA, anything else is treated as a WinGet package ID.
- Reboot state, superseded-update cleanup, and approved-GUID sync all go through dedicated `/patching/windows-updates/*` endpoints used by the beta Windows agent.

Windows patching is flagged **beta** in 2.0 and the Run Detail page renders the same way regardless of OS. The terminal pane simply shows PowerShell / `winget` output instead of `apt-get` output.

Where Patching Lives in the UI

There are two ways into patching from the left-hand navigation:

- **Patching** (top-level sidebar item): the fleet-wide view. This is the page described in Running a Patch, Patch Policies and Scheduling, and Patch History and Live Logs.
- **Hosts → *select a host* → Patching tab**: the per-host view. Start a **Patch all** run for that host, watch its packages list, open any previous patch run for this host. The tab is hidden when the `patching` module is disabled.

You can also enter the Patching UI from:

- A package link in the Packages page. "Patch this package" starts a `patch_package` wizard pre-loaded with the selected package and the hosts it is installed on.
- The Dashboard patching cards (queued/running counts), which deep-link into the Runs & History tab with the appropriate status filter applied.

What Happens When You Click "Patch All"

The end-to-end flow for a single `patch_all` run is:

- You click **Patch all** on a Host Detail page. The **Patch Wizard** opens, pre-loaded with the host.
- You optionally override the policy (e.g. "Run immediately" on a host that has a delayed policy) and click the fire button.
- The browser calls `POST /patching/trigger` with `patch_type=patch_all`. Because `patch_all` cannot be dry-run, the run starts in `pending_approval` if you ticked "Submit for approval", or goes straight to `queued` otherwise.

The server inserts a `patch_runs` row, snapshots the effective policy onto it, and enqueues a `run_patch` task on the `patching` asynq queue. If the policy introduces a delay, asynq schedules the task for the future and the run status shows `scheduled`.

When the task dequeues, the server sends a `run_patch` WebSocket message to the agent connected for that host.

The agent flips the run to `running`, calls `apt-get --with-new-pkgs upgrade -y` (or the equivalent for the OS), and streams stdout/stderr back over `POST /patching/runs/{id}/output` in short chunks. On Debian and Ubuntu, `--with-new-pkgs` lets the upgrade install packages it does not already have, which is what a kernel ABI bump needs. Without it apt holds those upgrades back and they would keep reappearing as outdated after every run.

The server fans each chunk out to any browsers subscribed to `GET /patching/runs/{id}/stream`, and persists the combined output to the database.

On success the agent sends a final `completed` stage with the authoritative shell output. The server marks the run `completed`, emits a `patch_run_completed` notification, and flags the host as "awaiting post-patch report" so the next inventory sync can update the package status.

The Run Detail page swaps the green **Live** pill for a subtle **Awaiting inventory report** pill, then for **New report received** once the agent sends its next scheduled inventory report and the system knows the on-host packages reflect reality.

See Running a Patch for the step-by-step operator walkthrough, and Patch History and Live Logs for everything to do with the terminal pane and log stream.

Related Documentation

Running a Patch: step-by-step, from trigger through live log to "patched".

Patch Policies and Scheduling: configure when patches actually run.

Patch History and Live Logs: work with the Runs & History table and live terminal output.

Release Notes: 2.0.0: the release that introduced the patching module.

Chapter 9: Running a Patch

This page walks you through starting a patch run from the PatchMon web UI, from the initial click through dry-run validation, approval and live log streaming, to the final "patched" state. Everything here happens in the browser against a logged-in session.

Assumes you have the `patching` module enabled and the `can_manage_patching` permission. If the action buttons are missing, see Patching Overview for the permission matrix.

Starting Points

There are three entry points into a patch run:

Entry point	What gets pre-filled	Typical use
Host Detail → Patching tab → Patch all	Target host is locked; patch type is `patch_all`	"Update everything on this host, now."
Host Detail → Patching tab → Patch selected packages	Target host is locked; patch type is `patch_package` with the packages you ticked	"Patch just these two CVEs on this host."
Packages → *select a package* → Patch this package	Package name is locked; host list is discovered from fleet inventory (only hosts that actually need the update)	"Roll out this specific package across the fleet."

All three funnel into the same **Patch Wizard**, a single modal used everywhere patching is initiated, so the mental model is identical regardless of where you started from.

The Patch Wizard

The wizard has a fixed six-step sequence, but it auto-skips steps that have no decision to make for your starting point. Unused steps are shown muted in the step indicator so you always see the full mental model.

#	Step	Shown when
1	**Hosts**	You entered from Packages (fleet rollout) and need to pick which hosts to patch. Hidden when the host is pre-locked.
2	**Packages**	You entered with a multi-package list and want to trim it. Hidden for `patch_all` or single-package runs.
3	**Validate**	`patch_package` only. Hidden for `patch_all` (cannot dry-run) and for Approve flows (validation already exists).
4	**Timing**	Always shown. Review effective policy and optionally override to "run immediately".
5	**Approval**	`patch_package` only. Choose "Approve & Patch now" or "Submit for approval". Hidden in Approve mode.
6	**Submit**	Always shown. Final per-host summary and the fire button.

Navigation is forward-only through the wizard; the **Back** button steps through enabled steps and skips the hidden ones.

Flow A: Patch All on a Single Host

This is the simplest case: update every out-of-date package on one host.

1. Open **Hosts** → *select your host* → **Patching** tab.
2. Click **Patch all**. The wizard opens at the **Timing** step (Hosts, Packages, Validate and Approval are all skipped for `patch_all`).
3. Review the effective patch policy shown on the Timing step. If the host has a delayed or fixed-time policy attached, the wizard tells you when the run will actually start (for example "Runs at 03:00 Europe/London").
4. If you need to bypass the delay, tick **Run immediately**. This sets `schedule_override=immediate` on the trigger call and fires the run as soon as the worker dequeues it.
5. Click **Next** to advance to **Submit**.
6. On **Submit**, read the per-host summary (host name, patch type, effective run time) and click **Queue & patch**.

What happens next on the server:

1. A `patch_runs` row is inserted with `patch_type=patch_all`, `dry_run=false`, and the policy snapshot.
2. A `run_patch` task is enqueued on the `patching` asynq queue.
3. If the policy introduces a delay, the task is scheduled for the future and the run is shown as `scheduled` in Runs & History. Otherwise it goes straight to `queued`.
4. The browser deep-links into the **Run Detail** page so you can watch the live terminal.

> ***Note:*** *`patch_all` cannot be dry-run. The agent's bulk-upgrade path (`apt-get --with-new-pkgs upgrade`, `dnf upgrade`, `pkg upgrade`, `pacman -Syu`) does not support a reliable simulation mode. If you want a dry-run, patch specific packages instead.*

Flow B: Patch a Specific Package with Dry-Run

This is the richer flow and the one to use for anything security-sensitive. The dry-run runs first, you review the transaction, then you approve.

1. Open **Patching** → click into the package from a host, or start from **Packages** → *package name* → **Patch this package**.
2. The wizard opens at the **Validate** step (Hosts and Packages steps may be visible for fleet rollouts).
3. Click **Run dry-run**. The browser calls `POST /patching/trigger` with `dry_run=true`. For each target host, the server:
    - Creates a `patch_runs` row in `pending_validation` status.

Sends the `run_patch` command to the agent with `DryRun=true`.

The agent runs the package-manager simulation step (for example `apt-get -s install <packages>`).

The wizard polls each run until it terminates. While you wait you see:

The per-host status badge (pending validation → running → validated).

A live terminal excerpt for the currently running host.

When the dry-run completes, the run transitions to `validated` and the wizard shows:

The final **Packages affected** list (what would be installed, always a superset of the original request because of dependency resolution).

An **Extra deps** badge if dependency resolution pulled in packages you didn't originally ask for.

The captured stdout/stderr for the simulation.

Review the output. If the transaction looks correct, click **Next** to the **Timing** step, choose **Run immediately** or leave the policy delay, and then **Approve & Patch** on the Submit step.

The browser calls `POST /patching/runs/{validationId}/approve`. The server:

Marks the validation run `approved` (terminal state; the row and output are preserved for audit).

Creates a **new** `patch_runs` row with `dry_run=false`, linked to the validation via `validation_run_id`.

Enqueues the real `run_patch` task with the effective policy delay.

Returns the new run ID; the UI deep-links you into its Run Detail page if it's going to start immediately.

Retrying a stuck validation

If the agent was offline when you triggered the dry-run, the run will sit in `pending_validation` until it comes back. You have two options:

Retry Validation: re-queues the same dry-run task. Call this once the agent is back online. Works via `POST /patching/runs/{id}/retry-validation` and is only available for `patch_package` runs.

Skip & Patch: bypasses the dry-run entirely and queues the real patch run directly. Use this when you're confident about the change and the host won't come back soon. The UI labels the button amber to make it clear you're skipping a safety step.

Both options are available from the Runs & History table, from the Run Detail page, and inline on a per-row basis.

Submitting a patch_all run for approval

`patch_all` can't be dry-run, but you can still route it through an approval gate. In the wizard's **Approval** step tick **Submit for approval**. The run is created in `pending_approval` status with no execution task enqueued. A second reviewer with `can_manage_patching` can then open Runs & History, click **Approve** on that row, and the server builds the real execution run the same way as a validated approval. Until that happens the run sits in the DB and can also be deleted.

Flow C: Patch a Package Across the Fleet

Same as Flow B, but starting from the **Packages** page. The wizard discovers which hosts have the package out of date (only those show up in the Hosts step) and validates each in parallel.

Navigate to **Packages** → click the package name → **Patch this package**.

The wizard opens at **Hosts**. Tick the hosts you want to patch, or **Select all**.

Click **Next** to **Validate**. The wizard fans out dry-runs across the selected hosts with a bounded concurrency pool (5 at a time by default) to avoid hammering the queue.

When every host has reached a terminal validation state, you see a per-host results table. Hosts with failed dry-runs are flagged so you can exclude them from the approval step or retry them.

Timing lets you pick a per-host policy override, useful when your fleet mixes policies.

Submit fires `POST /patching/runs/{id}/approve` once per validated run. The UI tracks failures in a bulk-approve result banner when you come back to Runs & History.

Watching a Run: The Run Detail Page

Once a run is fired and not delayed, the UI redirects you into `/patching/runs/{id}`. The page has:

A header with the host name, status badge, **Awaiting inventory report** pill (post-patch), and the primary action buttons for the current state (Approve & Patch, Retry Validation, Skip & Patch, Stop Run).

A left-side **Run summary** card: host, type, initiated by, approved by, started / completed timestamps, link to the validation run if any, patch policy in effect, and packages affected.

A right-side **Shell output** terminal for the live log stream.

Live log streaming

While the run is `running`, the Run Detail page opens a WebSocket connection to `/api/v1/patching/runs/{id}/stream`. The server's in-process `patchstream` hub fans out agent-published events to every connected browser:

snapshot: sent once when you connect. Contains the current stage and whatever `shell_output` is already persisted, so the terminal pane is primed even if you arrived mid-

run.

chunk: a short piece of stdout/stderr from the agent, appended to the terminal as it arrives.

done: a terminal stage (`completed`, `failed`, `cancelled`, `validated`, or `dry_run_completed`). The socket closes and the UI refetches run metadata to show the final state.

The header shows a pulsing green **Live** pill while the WebSocket is open. If you scroll up in the terminal to read earlier output, the UI stops auto-scrolling; scroll back to the bottom and it resumes.

If you arrive on a run that's already in a terminal state, the server sends a single `snapshot` message containing the persisted `shell_output` plus a synthetic `done` message, then closes the connection. The database is the source of truth, so you see exactly the same terminal contents as everyone else.

Copying output

When the run is not `queued` or `running`, a **Copy output** button appears above the terminal. It copies the full shell output to your clipboard. Use this for incident reports or to paste into a ticket.

The terminal normalises carriage returns. `apt-get` and `dpkg` use `\r` to overwrite progress bars on a single line, which would be invisible in a scrollback view. The UI converts `\r` to `\n` so every progress update becomes its own readable line.

Stopping a Running Patch

You can stop a run while it is in the `running` state. This is a hard stop with no graceful "let it finish the current package" behaviour.

On the Run Detail page, click **Stop Run** in the header.

Confirm in the dialog. The warning "Partially-installed packages may leave the host in an intermediate state" is there for a reason. Interrupting `apt` or `dnf` mid-transaction can leave dpkg or rpmdb needing manual repair.

The browser calls `POST /patching/runs/{id}/stop`. The server looks up the agent in the `agentregistry`, sends a `patch_run_stop` WebSocket message, and returns `202 Accepted`.

The agent cancels the subprocess via `SIGINT`, collects whatever output it has, and reports a terminal `cancelled` stage back to the server.

The live stream closes and the run's final status is `cancelled`.

When Stop Run is not available

The run is not in the `running` state (no subprocess to interrupt). For `queued`, `scheduled`, `pending_validation`, `pending_approval`, or `validated`, use **Delete** in Runs & History

instead. That also removes any scheduled task from the asynq queue.

The agent is not currently connected. The UI returns a 409 with "Agent is not currently connected". Wait for the agent to reconnect, or stop the agent's systemd service if you need the patch process killed at the OS level.

Post-Patch: The Awaiting Inventory Report Pill

When a `patch_all` or non-dry-run `patch_package` completes, the server sets `awaiting_post_patch_report_run_id` on the host. The Run Detail page shows an **Awaiting inventory report** pill next to the status badge, and the polling interval keeps ticking every 3 seconds.

The agent's next scheduled inventory report (usually within 60 minutes, sooner if triggered manually) updates the host's package list. The server clears the awaiting flag, and:

If the host's `last_update` timestamp is newer than the run's `completed_at`, the pill changes to **New report received**.

Otherwise the pill disappears. Absence is the soft success signal.

This is how you know the packages the patch run installed are now reflected in the Package Inventory view, not just that the apt command returned exit 0.

Troubleshooting Common Cases

The run sits in `queued` forever

The asynq worker picked up the task but cannot reach the agent, or the agent never received the WebSocket command.

Check the agent's connection status on the Host Detail page (green Connected pill).

Check the server logs for `patching:` entries around the task ID (`patch-run-<run-id>`).

If the agent has been offline and just reconnected, wait up to 30 seconds for the WebSocket re-handshake.

"package manager ... not supported for patching"

The agent rejected the run because the host's detected package manager is not in the supported list (`apt` , `dnf` , `yum` , `pkg` , `pacman`). This is the error you see on Alpine (`apk`) hosts today. The run immediately transitions to `failed` with the message in `error_message` .

The run completes but the package inventory hasn't updated

The agent hasn't sent its post-patch inventory report yet. The **Awaiting inventory report** pill will flip automatically once it arrives. If it doesn't arrive within an hour, force a report from the agent CLI:

```
sudo patchmon-agent report
```

Approve returns 400 "Only validated... runs can be approved"

Another operator already approved or deleted the run while you were looking at it. Reload Runs & History; the row will now be in `approved` or gone.

Related Documentation

Patching Overview: the three core concepts, module gates, and supported OS coverage.

Patch Policies and Scheduling: control when approved runs actually fire.

Patch History and Live Logs: the Runs & History table, filtering, and deeper dive into the live log stream.

Managing the PatchMon Agent: agent CLI, service management, and troubleshooting connection issues.

Chapter 10: Patch Policies and Scheduling

A **patch policy** controls *when* an approved patch run actually fires on a host. Policies let you carve out maintenance windows, build in a delay for "I might change my mind" runs, or force an immediate execution on anything that matters. Assignments and exclusions let you apply a policy broadly (to a host group) while still carving specific hosts out of it.

This page walks through the policy model, how effective policies are resolved, the Settings UI, and how policies interact with run triggers.

Module Gate and Permissions

Patch policies are gated by the `patching_policies` capability module, which is separate from the base `patching` module. A deployment with `patching` enabled but not `patching_policies` can still trigger patch runs; they just run immediately and cannot be scheduled through a policy.

Action	Required module	Required permission
View policies and their assignments	`patching_policies`	`can_view_hosts`
Create, edit, delete policies	`patching_policies`	`can_manage_patching`
Add or remove policy assignments (host / host group)	`patching_policies`	`can_manage_patching`
Add or remove host exclusions	`patching_policies`	`can_manage_patching`

If `patching_policies` is not enabled, the **Policies** tab on the Patching page is shown with an "Upgrade required" placeholder and a tier badge.

Where Policies Live in the UI

There are two equivalent entry points, both showing the same policy list with the same editor:

Patching page → Policies tab: the main view. Lists every policy, shows schedule type and assignment count, and lets you expand a policy to manage its assignments and exclusions inline.

Settings → Patch Management: the admin-centric view, identical in function, kept so patch policies are discoverable alongside other operational settings.

Both pages are backed by the same `/api/v1/patching/policies` endpoints, so any change you make in one is visible immediately in the other.

The Policy Model

Each policy has the following fields:

Field	Type	Description
`name`	string, required	Display name of the policy.
`description`	string, optional	Free-text description.
`patch_delay_type`	enum, required	`immediate`, `delayed`, or `fixed_time`.
`delay_minutes`	integer, required when `delayed`	Minutes to wait after the trigger before running.
`fixed_time_utc`	string, required when `fixed_time`	Time of day in `HH:MM` (or `HH:MM:SS`) format. Interpreted as local wall-clock time in the resolved organization timezone. The column name is retained for backward compatibility — see the timezone note below.
`timezone`	string, deprecated	Legacy IANA timezone field. **No longer read by the scheduler** and ignored on create/update. Persisted as `NULL` going forward. Existing values on old rows are kept for audit purposes only.

The three delay types

Immediate. The run fires as soon as the asynq worker dequeues the task. This is the default policy behaviour when no policy is attached to a host. Use for development hosts or anything where you actively approve each run.

Delayed. The run is scheduled for `now + delay_minutes` at the moment it is triggered. Typical values are 30-60 minutes, enough time for an operator to cancel if the trigger was a mistake, but short enough that the patch still lands in the current shift. The delay is counted from the trigger time (or approval time, for `patch_package`), not from policy creation.

Fixed time. The run is scheduled for the next occurrence of `HH:MM` interpreted as local time in the organization timezone. If that time has already passed today (in the local zone), the run is scheduled for the same time on the next local calendar day. Use for maintenance windows (`03:00` daily reboots, for example). Delays can be long: a run triggered at 14:00 for a 03:00 fixed-time policy will sit in `scheduled` status until the next 03:00 local.

Timezone handling

Fixed-time policies fire at the configured `HH:MM` interpreted as **local wall-clock time** in the **organization timezone**. The org timezone is resolved in this order:

1. `TZ` environment variable on the server process.
2. `TIMEZONE` environment variable on the server process.
3. **Settings → General → Timezone** (stored in the DB).
4. Final fallback: `UTC`.

The policy form shows the resolved zone next to the time input so operators can confirm which zone applies before saving. There is no per-policy timezone dropdown — scheduling is governed by a single org-wide timezone for consistency.

DST. During spring-forward, a policy time inside the missing hour is normalized one hour forward (e.g. `02:30` on the EU spring transition day fires at `03:30` local on that day). During fall-back, when a wall-clock time occurs twice, the standard-time (post-shift) occurrence is used. Review fixed-time policies in DST zones if a one-day shift on transition days would affect a maintenance window.

Column name. The DB column is still called `fixed_time_utc` for backward compatibility, but its contents are now local wall-clock time in the resolved timezone — not UTC. Renaming would require a multi-step migration; the name is preserved to avoid churn.

> ***Breaking change in this release.*** *Earlier versions parsed* `fixed_time_utc` *as a literal UTC time and ignored the per-policy timezone dropdown. Existing fixed-time policies created under the old behavior will now fire at a different absolute instant — review and adjust them after upgrade. The per-policy* `timezone` *field on the API is silently ignored on create/update going forward.*

Creating a Policy

Open **Patching → Policies** (or **Settings → Patch Management**).

Click **Create policy**. A modal opens with the policy form.

Fill in:

Name: required. Pick something that describes the window, not the host set (e.g. "Nightly 03:00 UTC", not "Production web tier"). Host assignment is done separately.

Description: optional but helpful.

Patch delay: `Immediate`, `Delayed (run after N minutes)`, or `Fixed time (e.g. 3:00 AM)`.

If you picked **Delayed**, enter the number of minutes (minimum 1).

If you picked **Fixed time**:

Enter the time in `HH:MM` format. The form displays the organization timezone next to the field; the time is interpreted as local wall-clock time in that zone (see the timezone handling note above).

There is no per-policy timezone dropdown. Change the org-wide zone under **Settings → General → Timezone** if you need a different default for scheduling.

Click **Create**. The policy appears in the list with `0 assignment(s)`.

Policies are empty until you assign them. A newly-created policy is inert and does not automatically apply to any host.

Assigning Policies

A policy can be assigned to a **host** (direct) or a **host group** (indirect). Direct assignments take precedence over group assignments; see Effective Policy Resolution below.

To assign a policy:

1. In the Policies list, click the **N assignment(s)** link on the policy row. The row expands to show the **Applied to** panel.
2. Choose **Host** or **Host group** from the dropdown.
3. Pick the target host or group from the second dropdown.
4. Click **Add**.

The assignment takes effect immediately for any future patch runs on that target. Runs already queued against the old effective policy are not recomputed; they keep the policy snapshot from the moment they were triggered (visible in the Run Detail sidebar).

To remove an assignment, click the `×` next to its chip in the **Applied to** list.

Exclusions

Exclusions let you carve a specific host out of a policy that it would otherwise inherit through a host group. **Direct host assignments cannot be excluded** because the precedence rules make the direct assignment always win.

Typical use:

1. You have a host group `production-web` with 50 hosts.
2. You assign a `Nightly 03:00 UTC` policy to that group.
3. One particular host in the group (`prod-web-api-01`) serves a customer in Singapore who cannot tolerate a 03:00 UTC outage (that's mid-day for them).
4. You add `prod-web-api-01` as an **Exclusion** on the policy. That host is then treated as having no policy (falls back to Default / immediate) even though it is still in the `production-web` group.
5. Optionally, assign `prod-web-api-01` directly to a different policy with a 19:00 UTC fixed time.

To add an exclusion, expand the policy and use the **Exclusions** row: pick a host from the dropdown, click **Exclude host**. The host is shown as an amber chip in the exclusions list.

Exclusions apply only to that specific policy. If the host is a member of another group assigned to a different policy, that other policy can still apply.

Effective Policy Resolution

When a patch run is triggered, the server resolves the effective policy for the target host using this precedence:

1. **Direct host assignment**: if the host has any policy directly assigned, that policy wins. Exclusions do not apply here (you can't direct-assign and then exclude).
2. **Group assignment**: if the host is a member of one or more host groups, the server walks the groups' policy assignments and picks the **first** policy (by assignment `created_at` ascending) where the host is **not** excluded.
3. **Default**: if none of the above applies, the effective policy is the implicit "Default" policy, which is equivalent to `patch_delay_type=immediate`. The Run Detail sidebar shows this as "Default policy. Runs immediately on trigger."

If a host is in multiple groups with conflicting policies, the oldest policy assignment wins. Order matters. If you need deterministic behaviour in a complex fleet, prefer direct host assignments over layered group policies, or design your groups so that each host is only in one "scheduling" group.

Checking the effective policy

Before triggering a run, the Patch Wizard's Timing step calls `GET /patching/preview-run?host_id=<id>` for each selected host. The response contains the `run_at_iso` time (what `ComputeRunAt` returns *right now*, computed in the org timezone for fixed-time policies) and the resolved policy's name, ID, and delay type. That's how the wizard tells you "Runs at 03:00 (Europe/London) via Nightly-Window" before you click fire.

The policy snapshot

When a run is created, the server also takes a **snapshot** of the effective policy onto the run row (`policy_snapshot` JSON). The snapshot is what the Run Detail page displays, and it is immutable; changing or deleting the policy later does not rewrite the snapshot. This is important for audit: "which policy was in effect when this run fired on 12 March?" always has an answer, even if the policy has since been deleted.

For fixed-time policies the snapshot also records `schedule_timezone`: the IANA name actually resolved when `run_at` was computed. Run Detail prefers this field when displaying the schedule, so changing the org timezone after the fact does not retroactively rewrite the audit trail.

Scheduling Semantics

Once the effective policy is resolved, the server converts it into an asynq job delay:

Policy type	`delayMs` computation	Visible run status
`immediate`	`0`	`queued` immediately
`delayed`	`delay_minutes × 60 × 1000`	`scheduled` for `run_at = now + delay_minutes`
`fixed_time`	ms until next `HH:MM` in the org timezone	`scheduled` for `run_at = next HH:MM` (local in the org zone, stored as UTC)

The `patch_runs` row stores both `created_at` (when the run was inserted) and `scheduled_at` (when asynq should release it to the worker). The Runs & History table shows `scheduled_at` as "Started" time if the run has not yet started.

Schedule overrides at trigger / approve time

Both `POST /patching/trigger` and `POST /patching/runs/{id}/approve` accept a `schedule_override` field. The only currently-supported value is `"immediate"`, which forces `delayMs=0` regardless of the effective policy. This is what the **Run immediately** checkbox in the Patch Wizard Timing step sets.

The snapshot on the run is still taken from the effective policy. The override only changes the actual firing time, not the policy metadata. Run Detail will show the real policy (e.g. "Nightly 03:00 UTC") but the run's `scheduled_at` will be absent and its status will jump straight to `queued`.

Deleting a scheduled run

A `scheduled` run can be deleted from Runs & History. When you click **Delete** on a scheduled row:

> The server removes the run's row from the `patch_runs` table.
>
> It also calls `inspector.DeleteTask("patching", "patch-run-<id>")` to remove the queued asynq task, so the run doesn't fire after being deleted.

Deletion is only allowed for runs in `queued`, `pending_validation`, `pending_approval`, `validated`, `approved`, or `scheduled` status. Anything `running` or terminal is not deletable (use **Stop Run** for a running run; terminal runs are historical records and cannot be removed from the UI).

Editing and Deleting Policies

Editing a policy in place (change name, description, delay type, or delay value) is supported from the Policies list. Click the pencil icon on a policy row to open the edit modal, then **Update**.

Existing runs are **not** re-scheduled when you edit a policy; their snapshot was taken at trigger time. Only future runs will use the new values.

Deleting a policy removes it immediately. All assignments and exclusions attached to it are removed with it (cascade delete). Any run in `scheduled` status that was created from this policy keeps its `scheduled_at` and still fires when the time comes. The policy ID on the run becomes a dangling reference, but the policy name is preserved in the `policy_snapshot` column for the UI.

If you're replacing a policy with a new one, prefer reassigning hosts and groups to the new policy before deleting the old one.

Common Patterns

Single maintenance window across the fleet

Create one policy (`Nightly 03:00 UTC`) with `fixed_time` at `03:00`. Assign it to a top-level host group that contains everything, or to each host directly. Use exclusions for the handful of hosts that need a different window.

Canary-then-production

Create two policies:

> `Canary 01:00 UTC` with `fixed_time` at `01:00`, assigned to your `canary` host group.
> `Production 04:00 UTC` with `fixed_time` at `04:00`, assigned to your `production-all` host group.

Trigger the same `patch_package` run on both groups at the same time. The canary hosts patch first; production follows three hours later. If canary reports failures, delete the still-`scheduled` production runs before they fire.

Slow-rollout "hold for 30 minutes"

Create a `Delayed 30min` policy with `patch_delay_type=delayed`, `delay_minutes=30`. Assign it to everything. Every approved run goes into `scheduled` status for 30 minutes before firing. If you realise you approved the wrong thing, delete the scheduled run; otherwise it fires automatically.

Mixed: immediate by default, fixed-window for production

> Leave most hosts with no assignment. They fall through to the Default (immediate) policy.

Create a `Prod 03:00 UTC` policy and assign it directly to the `production` group.

A patch triggered from the Packages page across the fleet then runs immediately on dev/staging and waits for the next 03:00 UTC on production.

Related Documentation

Patching Overview: the three core concepts and how patching fits together.

Running a Patch: the Patch Wizard flow, including the Timing step that reads the effective policy.

Patch History and Live Logs: reading run history, including the policy snapshot shown on each run.

Hosts and Groups: managing host groups, which are the usual unit of policy assignment.

Chapter 11: Patch History and Live Logs

The **Runs & History** tab on the Patching page is where every past and pending patch run lives. This page covers how to read the history table, filter and search it, select runs for bulk actions, and work with the live log stream on the Run Detail page.

Getting to Runs & History

From the left sidebar, click **Patching** and switch to the **Runs & History** tab. The URL becomes `/patching?tab=runs` and is shareable.

Deep-link filters are also supported via URL parameters:

- `/patching?tab=runs&status=active` : queued + running
- `/patching?tab=runs&status=failed` : only failed runs
- `/patching?tab=runs&status=completed` : only completed runs
- `/patching?tab=runs&status=pending_approval&type=patch_all` : combined filter

These are the same URLs the Patching dashboard cards link to when you click the **Total runs / Queued / Completed / Failed** tiles at the top of the page.

The Runs & History Table

The table has eight columns on desktop:

Column	What it shows
Delete checkbox	Selects the row for bulk delete. Only shown for deletable statuses (`queued` , `pending_validation` , `pending_approval` , `validated` , `approved` , `scheduled`).
Approve checkbox	Selects the row for bulk approval. Only shown for approvable statuses (`validated` , `pending_validation` , `pending_approval`).
Host	Friendly name if set, otherwise hostname, otherwise host UUID. Clickable from the Run Detail page sidebar.
Type	Summary of the run type: "Patch all", or a compact list of package names for `patch_package` (e.g. `curl, openssl`). Dry-runs render the same way but the status badge tells you it was a validation.
Status	The run status badge plus an **Extra deps** pill when a validated run would install more packages than you requested.
Initiated by	The username of the operator who triggered the run. Empty for runs triggered by automation.
Started	`created_at` timestamp for not-yet-started runs, `started_at` for running / completed runs.
Completed	`completed_at` timestamp if the run has finished, otherwise blank.
Actions	Inline action buttons: **Retry**, **Skip & Patch**, **Approve**, **View**. See Inline row actions below.

On mobile (<768px) the table collapses into per-run cards with the same information stacked vertically. Actions sit at the bottom of each card as full-width buttons.

Pagination and page size

The table is paginated server-side via `GET /patching/runs?limit=<N>&offset=<M>` . The default is 25 rows per page. You can change the page size to 50, 100, or 200 from the dropdown at the bottom; your choice is remembered in `localStorage` under `patching-runs-limit` .

The runs list is sorted by `created_at` descending by default, with newest runs at the top. The server also accepts `sort_by` (`created_at` , `started_at` , `completed_at` , `status`) and `sort_dir` (`asc` , `desc`) but the UI does not expose sort controls today; filter and paginate to narrow the window instead.

Filtering

Two filters are available above the table:

> **Status**: `All` , `Active (queued + running)` , `Queued` , `Pending validation` , `Pending approval` , `Validated (awaiting approval)` , `Approved` , `Scheduled` , `Running` , `Completed` ,

`Failed`, `Cancelled`.

Type: `All`, `Patch all`, `Patch package`.

Filters reset the pagination to page 1. Click **Clear filters** to remove both. The selected filters are also encoded in the URL, so you can bookmark or share a filtered view.

Empty states

If the filters match nothing, the table shows "No runs match your filters" with a prompt to adjust the filter.

If there are no runs at all (fresh install), the table shows "No patch runs yet. Patch runs triggered from the Overview tab or from host detail pages will appear here."

Inline Row Actions

The rightmost **Actions** column shows action buttons specific to the row's current status:

Status	Buttons shown
`pending_validation`	**Retry** (re-queue the dry-run), **Skip & Patch** (bypass validation and go straight to executing), **View**
`pending_approval`	**Approve**, **View**
`validated`	**Approve**, **View**
All others	**View** only

View always opens the Run Detail page at `/patching/runs/{id}`.

Approve and **Skip & Patch** both route through the **Patch Wizard** in approve mode, even for a single row. This is a deliberate consistency choice: every path that turns a validation into a real run uses the same UI, so you get the per-host policy override UI for free (e.g. you can pick "Run immediately" at approval time even if the host normally has a delayed policy).

Retry re-queues the dry-run task without opening the wizard. Use this after bringing an offline host back online. Only available for `patch_package` runs; `patch_all` cannot be re-validated because it cannot be dry-run.

Bulk selection and bulk actions

At the top of each row are two optional checkboxes, one for delete and one for approve. Clicking them adds the row to a selection set; the header checkbox selects every eligible row on the current page.

Once at least one row is selected, a **bulk action bar** appears above the table:

Delete N selected: deletes every selected run. This also removes any scheduled asynq task for each run.

Approve N selected: opens the Patch Wizard pre-loaded with every selected validation. You get one Timing / Submit sequence for the batch, with per-host policy overrides.

After a bulk approval, the UI shows a summary banner (e.g. "Approved 5, 1 failed"). Failures for individual hosts are surfaced without blocking the other approvals.

You cannot mix a delete selection and an approve selection for the same row. The two checkboxes toggle independently and both sets are tracked. You can clear either selection at any time with the **Clear delete** or **Clear approve** buttons.

The Run Detail Page

Click **View** on any row, or navigate directly to `/patching/runs/{id}`, to open the Run Detail page. The layout is:

Header: back arrow, host name as H1, subtitle with run type / status badge / post-patch pill, and primary action buttons (Approve & Patch, Retry Validation, Skip & Patch, Stop Run) right-aligned.

Run summary sidebar (left, on desktop): host, type, initiated by, approved by, started / completed / scheduled-for, link to the linked validation run (when applicable), patch policy in effect, and the full list of packages affected.

Primary content (right): state banners for non-terminal statuses (`pending_validation`, `pending_approval`, `validated`), an error panel when `error_message` is set, and the **Shell output** terminal.

State banners

The Run Detail page shows a context-specific banner for every non-terminal state so you immediately know what the run is waiting for:

Pending validation: "Validation pending. Host may be offline." Explains that you can retry when the host is back, or skip validation.

Pending approval: "Awaiting approval." Explains that the run was submitted for approval and needs a second reviewer.

Validated: "Validation complete. Approval required." If the run would install more packages than you asked for, the banner includes the dependency count so you know to review the **Packages affected** panel.

Polling cadence

While the page is open, the Run Detail query refetches on a status-dependent interval:

`queued`, `pending_validation`: every 3 seconds.

`running`: every 5 seconds if the live WebSocket is open, every 3 seconds otherwise (the faster poll is a safety net if the WebSocket drops).

`completed` and the host is still flagged as "awaiting post-patch inventory report": every 3 seconds so the **Awaiting inventory report** pill flips to **New report received** as soon as the agent's next report arrives.

Everything else: no automatic refetch.

Live Log Streaming

When a run is `running`, the Run Detail page opens a WebSocket to `/api/v1/patching/runs/{id}/stream`. Authentication is the same JWT cookie you use for the rest of the UI; the outer Auth middleware handles the upgrade.

Message types

The stream speaks JSON. Three message types arrive from the server:

```
// Sent exactly once when the browser connects
{ "type": "snapshot",
  "patch_run_id": "...",
  "stage": "running",
  "shell_output": "Reading package lists...\n...",
  "error_message": "" }

// Sent for each line-buffered stdout/stderr chunk the agent pushes
{ "type": "chunk",
  "patch_run_id": "...",
  "stage": "progress",
  "chunk": "Setting up libssl3 (3.0.2-0ubuntu1.15)...\n" }

// Sent once when the run reaches a terminal stage on the agent
{ "type": "done",
  "patch_run_id": "...",
  "stage": "completed",        // or failed / cancelled / validated /
dry_run_completed
  "error_message": "" }
```

The browser appends every `chunk.chunk` to its local terminal buffer. When `done` arrives, the browser closes the socket and invalidates the run query so the page refetches the final persisted state.

Keepalive

The server sends a WebSocket ping frame every 30 seconds to keep the connection alive through proxies and load balancers. Write operations use a 10-second deadline; a stuck client is dropped rather than pinning a goroutine. There is no agent or server-side retry logic for the

browser socket. If the page reconnects (for example, after a brief network blip), the `snapshot` replays the full buffered output, and any missed chunks that were persisted to the database are included in it.

Why you may see "(No output yet)"

There's a small window after clicking **Queue & patch** where the run is still `queued`:

- The asynq worker has not yet dequeued the task.
- The agent has not yet received the `run_patch` command.
- No output has been published.

The terminal shows "(No output yet)" during this window. The status badge tells you `Queued`; once the agent flips to `running`, the first `chunk` arrives within a second or two.

Terminal rendering

The Run Detail page renders output in a GitHub-dark-styled `<pre>` at ~420px tall (55vh max). It is scrollable and word-wrapping is preserved. Progress-bar `\r` characters from `apt-get` and `dpkg` are converted to `\n` so each progress update becomes its own line in the scrollback. You lose the animated overwrite but gain readability.

If you scroll up manually (more than ~32px from the bottom) the UI stops auto-scrolling so it doesn't fight you. Scroll back to within 32px of the bottom and auto-scroll resumes.

Copying output

When the run is in any non-`running`, non-`queued` state, a **Copy output** button appears above the terminal. It copies the full `shell_output` to your clipboard via `navigator.clipboard.writeText`. Use this to paste into a ticket, email, or post-incident report.

No Built-in Export or Download

There is no server-side export endpoint for runs. You cannot download a run's log as a file, and there is no CSV/JSON export of the Runs & History table. Options if you need one:

- For a single run: use **Copy output** on the Run Detail page and paste into a file yourself.
- For bulk analysis: hit the API directly (`GET /api/v1/patching/runs?limit=200&offset=0`) and write the JSON to disk. The API is paginated to 200 rows max per request, and authenticates with the same JWT bearer token as the web UI.

> ***Note:*** *The listed `GET /patching/runs` response includes the run metadata (status, timestamps, host info, package list) but not the full shell output. To get shell output in bulk, iterate over run IDs and fetch each with `GET /patching/runs/{id}`.*

Notifications for Runs

Run lifecycle events emit notifications via the normal notifications pipeline. Destinations like SMTP, webhooks, or ntfy configured in **Settings → Notifications** see them. The events are:

Event type	Emitted when	Default severity
`patch_run_started`	Agent reports `running` stage	informational
`patch_run_approved`	An operator approves a validation	informational
`patch_run_completed`	Agent reports `completed` stage (non-dry-run)	informational
`patch_run_failed`	Agent reports `failed` stage	error
`patch_run_cancelled`	An operator deletes a not-yet-running run	informational

The notification message includes the host name, patch type, package list (truncated to 5 with "... and N more"), effective policy name, and (for failures) the captured error message truncated to 300 characters. Severity is resolved via the per-event alert settings, so you can raise or lower the default by event type in the alerts configuration.

Deleting Runs

Runs in `queued`, `pending_validation`, `pending_approval`, `validated`, `approved`, or `scheduled` state can be deleted. Deletion:

- Removes the row from `patch_runs`.
- Removes the asynq task (`patch-run-<id>` and `patch-run-<id>-retry` if it exists) from the queue.
- Emits a `patch_run_cancelled` notification event.

Terminal runs (`completed`, `failed`, `cancelled`, `dry_run_completed`) cannot be deleted from the UI; they are historical audit records. If you need to purge old runs for storage reasons, contact support or write a direct database query targeting `patch_runs.created_at`.

Running runs cannot be deleted. Use **Stop Run** (see Running a Patch) instead; that issues a graceful cancel through the agent.

Related Documentation

- Patching Overview: the three core concepts: run, policy, dry-run.
- Running a Patch: detailed walkthrough of triggering, approving, and stopping a run.
- Patch Policies and Scheduling: policy model, assignments, and how the schedule is computed.

Alerts and Notifications: configure where patch run events get delivered.

Chapter 12: Enabling Docker Integration

Overview

The PatchMon agent includes an optional **Docker integration** that discovers containers, images, volumes, and networks on the host and reports them to the PatchMon server. When enabled, the agent also subscribes to the Docker event stream and relays container lifecycle events (start, stop, die, pause, unpause, kill, destroy) as status updates, keeping the fleet-wide Docker inventory broadly in sync with what is running.

This page covers what the integration does, how to enable it per host from the PatchMon UI, what the agent needs on the host, how `config.yml` reflects the toggle, and what to check when the integration doesn't report.

Module gate: The Docker views (`/docker/*` routes, Docker tabs on Host Detail) require the `docker` module to be enabled on your plan. Plans without the module show a tier badge on the Docker tab and an upgrade prompt when you navigate to `/docker` .

Permission required: `can_manage_hosts` to toggle the integration; `can_view_hosts` to see the resulting inventory.

What the Integration Does

When enabled on a host, the agent:

Discovers inventory on each report: lists all containers (running and stopped), images (including intermediate layers PatchMon elects to ignore), volumes (local / NFS / custom driver), and networks (bridge / host / overlay / macvlan / user-defined).

Streams container events in real time: subscribes to the Docker daemon's event bus. Each relevant event (start, stop, die, pause, unpause, kill, destroy) is translated into a `container_start` / `container_stop` / `container_die` / `container_pause` / `container_unpause` / `container_kill` / `container_destroy` status event and pushed to the server over the agent's WebSocket.

Resolves image provenance: for each image, PatchMon attempts to attribute it to a registry (Docker Hub, GHCR, GitLab, Quay, ECR, ACR, GCR, local, private) and makes the registry entry clickable when possible.

Tracks available updates: compares the running image tag against available tags in the registry (where the registry allows it) and flags images with newer versions.

Feeds the Compliance module: if compliance scanning is also enabled for the host, Docker Bench for Security can run as an additional scanner (see the Compliance chapter).

The resulting data powers two places in the UI:

- The **Docker Inventory** page at `/docker`: fleet-wide view across all hosts. See Docker Inventory Tour.
- The **Docker tab** on the Host Detail page: the same data filtered to one host.

Agent Prerequisites on the Host

The Docker integration talks directly to the Docker daemon through the Unix socket. What the agent needs on the host:

Requirement	Detail
Docker Engine installed	Any reasonably recent version; the agent uses the Go Docker client SDK.
Docker socket present	The agent looks for `/var/run/docker.sock`. If the socket is missing (Docker not installed, or not yet started), the integration reports as unavailable.
Agent has read access to the socket	The agent runs as `root`, which has access on standard installs. On hosts where `docker.sock` is mode `0660` and owned by `root:docker`, root access is fine. Custom Docker configurations that tighten socket permissions further may need adjustment.
Docker daemon responsive	The agent pings the daemon when it first checks availability; a responsive ping confirms Docker is up. If Docker is installed but the service isn't running, the agent waits for it and retries rather than crashing.

The Docker binary (`docker`) is **not** required on the PATH. The agent uses the Docker Engine API directly via the socket, so the CLI is optional. You can verify the socket path and daemon version from the host with a quick command:

```
ls -l /var/run/docker.sock
docker version   # if the CLI is installed
```

Windows hosts: The agent's Docker integration is Linux / FreeBSD only. Windows hosts do not surface the Docker tab regardless of whether Docker Desktop is installed.

Enabling the Integration from the UI

There are two places to switch it on:

On a new host during enrolment

On **step 2** of the **Add Host** wizard (**Host details**), the **Integrations** section has a **Docker** toggle. Tick it before clicking **Next**. When the agent first connects, it will already have `docker: true` in its `config.yml` and start collecting data on the first report.

See Adding a Host.

On an existing host

1. Open **Hosts** → click the host's friendly name to open its Host Detail page.
2. Click the **Integrations** tab.
3. Find the **Docker** panel.
4. Click the toggle at the right of the panel to set it to **Enabled**.
5. A yellow banner appears at the top of the tab and the page header: *Pending configuration changes*.
6. Click **Apply** in the page header to push the change to the agent over the WebSocket.

The agent then:

1. Updates `config.yml` to set `integrations.docker: true`.
2. Re-initialises its integration manager.
3. Starts collecting Docker inventory on the next reporting cycle (typically within one report interval; the default is 60 minutes, but the initial report after enabling is sent immediately).
4. Begins streaming container events.

What "Pending configuration changes" means

The toggle on the UI writes the **desired** state to the PatchMon server. The change is only actually sent to the agent when you click **Apply**, which broadcasts the new config over the WebSocket. If the agent is offline, **Apply** is disabled and the banner tells you so. The change waits in pending state until the agent reconnects.

You'll see `integrations.docker` change in the agent's `config.yml` shortly after **Apply** is clicked, without needing to restart the service (the update interval and integration toggles are synced at runtime).

Disabling the Integration

From the same **Integrations** tab on the host:

1. Click the **Docker** toggle to **Disabled**.
2. Click **Apply** in the page header.

After the change is applied:

1. The agent stops discovering Docker inventory.
2. Existing inventory records remain in PatchMon (so historical queries and event history are preserved), but the agent will no longer refresh them.
3. Container event streaming is stopped.

Disabling does not remove Docker from the host or stop containers; it only instructs the agent to stop monitoring.

How It Looks in `config.yml`

The agent config file is located at:

- **Linux / FreeBSD:** `/etc/patchmon/config.yml`
- **Windows:** `C:\ProgramData\PatchMon\config.yml`

The integrations block contains the Docker toggle:

```
integrations:
  docker: true          # Enabled by the UI toggle
  compliance:
    enabled: false
    on_demand_only: true
    openscap_enabled: true
    docker_bench_enabled: false
```

See Agent Config YML Reference for the full schema and how each field behaves.

You can enable Docker integration by editing the config file directly, but using the UI toggle is strongly preferred: it keeps the server's view of the host in sync with the config and prevents a subsequent **Apply** from silently overwriting your edit.

When Docker Integration Doesn't Report

Symptoms you might see:

- The **Docker** tab never appears on the Host Detail page, even after enabling the integration and clicking **Apply**.
- The host enabled Docker, but no containers or images appear at `/docker`.
- Real-time status (container start / stop) doesn't update.

Work through the checks below in order.

1. Confirm the server saw the toggle

On the Host Detail page, open the **Integrations** tab. The Docker panel should show **Enabled** with a green badge. If it shows **Disabled**: the change was not saved. Toggle again and click **Apply**.

2. Confirm the agent received the config

On the host:

```
sudo grep -A 4 '^integrations:' /etc/patchmon/config.yml
```

Expect:

```
integrations:
  docker: true
  ...
```

If the file still shows `docker: false`, the **Apply** button wasn't clicked or the agent's WebSocket wasn't connected at the time. Back in the UI, look at the page header. If the **Apply** button is still visible, click it again (the agent must be connected).

3. Confirm the Docker socket is accessible

```
ls -l /var/run/docker.sock
sudo docker ps        # agent runs as root, so sudo mimics its view
```

If the socket is missing, Docker isn't installed or isn't running. Install / start Docker and watch the next report.

4. Look at the agent's log for Docker errors

```
sudo tail -n 50 /etc/patchmon/logs/patchmon-agent.log | grep -i docker
```

Typical messages:

- `Docker socket not found`: Docker isn't installed, or the socket has a non-standard path.
- `Failed to create Docker client`: the socket exists but the agent can't open a client; check permissions.
- `Docker container event`: confirms the event stream is active and receiving events.
- `Docker daemon ping failed, retrying`: Docker is installed but unresponsive. The agent will keep retrying.

For the full logging reference, see Managing the PatchMon Agent.

5. Force a report and recheck

Back in the UI, on the Host Detail page, click **Fetch Report**. The agent collects a fresh inventory (including Docker) and reports immediately. Watch the **Docker** tab count badges update.

6. Refresh integration status

On the **Integrations** tab, the **Refresh Status** button asks the agent to report its current integration readiness state. Useful after installing Docker, fixing socket permissions, or starting the Docker service.

7. Module gate

If the Docker tab shows a tier badge instead of content, the `docker` module is not enabled on your plan. Contact your PatchMon administrator to enable it on the subscription plan.

Related Pages

Docker Inventory Tour: what the inventory looks like once the integration is working.

Host Detail Page: where the Integrations tab lives and how **Apply** works.

Adding a Host: enable Docker at enrolment time.

Agent Config YML Reference: every field in `config.yml`, including the integrations block.

Managing the PatchMon Agent: agent logs, diagnostics, and service management.

Chapter 13: Docker Inventory Tour

Overview

Once the **Docker integration** is enabled on one or more hosts, PatchMon aggregates the discovered containers, images, volumes, and networks into a fleet-wide **Docker Inventory**. The inventory answers "what's running where" questions: which hosts have Docker, which containers are running, which images are out of date, and which volumes and networks exist across the estate.

This page is a guided tour of `/docker` (the fleet view), `/docker/hosts/:id` (per-host view), and the detail pages for containers, images, volumes, and networks.

Module required: `docker`. Plans without the module show a tier-badge prompt on the Docker tab and the `/docker` routes. Contact your PatchMon administrator to enable it.

Permission required: `can_view_hosts` to view the inventory; `can_manage_hosts` to delete Docker resources from the UI.

Getting to the Docker Page

Click **Docker** in the left navigation. You land on `/docker` with **Stacks** selected by default. The URL accepts a `?tab=` parameter (`stacks`, `containers`, `images`, `volumes`, `networks`, `hosts`) so you can deep-link to a specific tab.

If no hosts have the Docker integration enabled, the list sections are empty. See Enabling Docker Integration to turn it on for a host.

Top-of-Page Statistics

Four summary cards sit above the tabs:

Card	Meaning	Click behaviour
Hosts with Docker	Hosts actively reporting Docker inventory	–
Running Containers	`running / total` counts across the fleet	–
Total Images	Distinct images reported across all hosts	–
Updates Available	Images PatchMon knows have newer tags in their registry	Opens the **Images** tab filtered to *Updates available*

These figures come from the `/docker/dashboard` endpoint and are refreshed every **30 seconds** automatically while the page is open.

Tab Strip

Six tabs. Each has a counter badge to show at a glance how big the fleet is across that dimension:

Stacks: containers grouped by their Compose project / stack label.

Containers: every container on every host.

Images: every image on every host.

Volumes: every volume on every host.

Networks: every network on every host.

Hosts: a directory of hosts with Docker enabled.

Clicking a tab resets the page's search field and sets a sensible default sort for that view (status on containers, repository on images, name elsewhere).

Stacks tab

Groups running containers by their Compose project / stack label. Each group card shows:

Stack name.

Number of containers in the stack, broken down by status.

Host the stack runs on (stacks are scoped per host; a stack of the same name on two hosts appears twice).

Links to the individual containers and their images.

Use this tab when you think in terms of "my `wordpress` stack" rather than "individual containers".

Containers tab

One row per container. Columns include:

Name: click to open the container detail page (`/docker/containers/:id`).

Image: the image tag, with registry-aware linking (click a Docker Hub image to jump to Docker Hub, GHCR to GitHub, etc.).

Status: colour-coded badge: **running** (green), **exited** (red), **paused** (yellow), **restarting** (blue), or plain for other states.

Host: friendly name of the host the container lives on. Click to jump to the per-host Docker view.

Optional columns: created timestamp, ports, state transitions.

Filters:

Search across name, image, and host.

Status filter: All, Running, Exited, Paused, Restarting.

Sort by name, image, status (with secondary sort by name within status), or host.

Actions:

Delete (trash icon): deletes the container via the agent. Requires `can_manage_hosts` . Errors are surfaced in an alert.

Images tab

One row per image. Rows show:

Repository + tag (and registry link when recognised).

Size.

Source (Docker Hub, GHCR, GitLab, Quay, ECR, ACR, GCR, local, private, unknown) as a coloured badge.

Container count: how many containers on which hosts reference this image.

Update indicator: a chip when a newer tag is available in the registry.

Filters include source type and an **Updates available** filter (the same filter the top **Updates Available** card opens).

Clicking an image opens `/docker/images/:id` : the image detail page with a list of the hosts that have the image and the containers using it.

Volumes tab

One row per volume with:

Name, driver (`local` , `nfs` , custom).

Mountpoint on the host.

Host the volume lives on.

Container count: how many containers mount it.

Filter by **Driver** and search. Click a volume name to open `/docker/volumes/:id`, which shows which containers currently mount it, along with the host.

Networks tab

One row per network with:

Name, driver (`bridge`, `host`, `overlay`, `macvlan`, `none`, custom), scope.

IPAM subnet / gateway.

Host the network exists on.

Container count.

Filter by **Driver** and search. Click a network name to open `/docker/networks/:id` with container membership.

Hosts tab

A compact directory of hosts that have the Docker integration enabled, sorted alphabetically by friendly name. Each row summarises container / image counts for that host and links to the per-host Docker view at `/docker/hosts/:id`.

Use this tab as the starting point when you want to focus on a single host rather than pivot by resource type.

Per-Host Docker View

The URL `/docker/hosts/:id` (and the row link on the **Hosts** tab) opens a view scoped to one host. It shows:

The host's friendly name, hostname, and a link back to the main Host Detail page.

Container and image counts, running / exited / paused breakdowns.

The host's container list, grouped by stack where available.

The host's image list.

This view is equivalent to the **Docker** tab on the host's main Host Detail page (see Host Detail Page). Either works. Pick whichever route you land on.

Resource Detail Pages

Each Docker resource has its own detail page. They follow the same pattern: top section with identifying metadata, cards with stats, related resources, and any actions.

Container detail: `/docker/containers/:id`

Shows the container's name, image, status, ports, host, created and started timestamps, restart policy, command and entrypoint, labels, mounts, and networks.

A **Similar containers** strip at the bottom lists other containers using the same image across the fleet, useful for "is this `redis:7` running anywhere else?" questions.

Image detail: `/docker/images/:id`

Shows repository, tag, digest, size, architecture, OS, labels, history (layers), and the registry link.

Below, two lists:

- **Hosts with this image**: every host pulling it, with the tag they currently have.
- **Containers using this image**: every container across the fleet referencing this image.

An **Updates** panel appears when a newer tag is available in the source registry.

Volume detail: `/docker/volumes/:id`

Shows driver, mountpoint, size (when Docker reports it), labels, and options. The **Containers using this volume** list shows where it is mounted.

Network detail: `/docker/networks/:id`

Shows driver, scope, IPAM configuration, and options. The **Containers attached** list shows what's connected to the network.

How the Data Stays Current

Docker data flows into PatchMon on two channels:

Periodic inventory reports

Every time the agent runs its regular report cycle (default: 60 minutes, configurable server-side), it enumerates containers, images, volumes, and networks and sends the snapshot to the server. The full inventory in `/docker` reflects the **last snapshot** from each host.

To force an immediate refresh for a single host, open its **Host Detail** page and click **Fetch Report**.

Real-time container events

When the Docker integration is enabled, the agent also subscribes to the Docker event stream and pushes container lifecycle events over its existing WebSocket connection. The relevant event types are:

- `container_start` (maps to `running`)
- `container_stop` / `container_die` / `container_kill` (all map to `exited`)
- `container_pause` (`paused`)

`container_unpause` (`running`)

`container_destroy` (`removed`)

The server records these events against the container record so that, for example, a container crash is visible in the UI within seconds rather than waiting for the next full report.

UI refresh cadence

On top of those agent-driven pushes, the `/docker` page itself refreshes the dashboard summary every 30 seconds via polling, and per-tab queries refetch when you switch tabs. The manual **Refresh** button (top right, circular arrow) forces an immediate refetch of whichever tab is active.

> ***Tip:*** *If you change something on a host (start / stop a container, pull an image) and want to see it in the UI, the event should appear within a few seconds via the WebSocket push. A full refresh of image / volume / network inventory waits for the next report. Use* ***Fetch Report*** *on the Host Detail page if you can't wait.*

Deleting Docker Resources

Containers, images, volumes, and networks can be deleted from their table rows (trash icon) or from their detail pages. Deletion:

Requires `can_manage_hosts`.

Opens a confirmation modal listing the resource and its host.

Sends a delete command to the agent over the WebSocket.

The agent executes the equivalent `docker rm` / `docker rmi` / `docker volume rm` / `docker network rm` and reports the outcome back.

If Docker refuses (for example because a container is still running, or an image is still referenced by a container), the UI surfaces the error inline.

Search and Sort Persistence

Search and filter state are per-tab and reset when you switch tabs (so switching from Containers to Images doesn't carry a container-specific filter into the Images view). Sort field and direction reset to the tab's default when you switch, too.

The main **Refresh** button also clears any per-tab "updates available" filter that was set via the dashboard card click.

Related Pages

Enabling Docker Integration: how to switch the integration on for a host.

Host Detail Page: the per-host Docker tab, equivalent to `/docker/hosts/:id`.

Managing the PatchMon Agent: the agent that collects the Docker data.

Agent Config YML Reference: the `integrations.docker` setting in `config.yml`.

Chapter 14: Compliance Overview

What Compliance Scanning Is

Compliance scanning evaluates your hosts against published security benchmarks: **CIS Benchmarks** for the operating system and **Docker Bench for Security** for container hosts. Results are reported per rule (pass, fail, warning) back to the web UI, giving you a fleet-wide compliance score, per-host rule detail, and optional auto-remediation for failed rules.

Scanning is performed by the PatchMon agent on each host, not by the server. The agent runs the scanner locally, parses the output, and submits structured results back to the server via `POST /api/v1/compliance/scans`. The server aggregates the data into dashboards and rule views.

This page covers the overall model: what the scanners are, how SSG content is delivered in 2.0, the module gate, and the permission matrix. For walkthroughs of actually running scans and reading results, see Running Compliance Scans and Results and Remediation.

Module Gate

All compliance UI and API routes are gated by the `compliance` capability module. Some plans (smaller tiers) do not include compliance at all; on those plans the **Security Compliance** sidebar item is hidden and the corresponding API endpoints return 403.

UI area	Required module
Security Compliance page (all tabs)	`compliance`
Host Detail → Compliance tab	`compliance`
Compliance-related per-host settings (mode, scanner toggles, default profile)	`compliance`

If the module is disabled, the Host Detail page shows an "Upgrade required" placeholder in the Compliance tab and the dashboard hides compliance cards.

Permission Matrix

Compliance uses three RBAC permissions on top of the module gate. Each API route applies a specific combination:

Action	Required permission	Example route
View the dashboard, scan history, host compliance detail, rule detail, trends, and active scan list	`can_view_reports`	`GET /compliance/dashboard`, `GET /compliance/scans/{hostId}`
Trigger scans (single or bulk), cancel a running scan, install the scanner, upgrade SSG content, trigger per-rule remediation	`can_manage_compliance`	`POST /compliance/trigger/{hostId}`, `POST /compliance/cancel/{hostId}`, `POST /compliance/remediate/{hostId}`
Change per-host compliance mode, per-host scanner toggles (OpenSCAP / Docker Bench), default profile for a host	`can_manage_hosts`	`POST /hosts/{hostId}/integrations/compliance/mode`, `POST /hosts/{hostId}/integrations/compliance/scanners`

In practice a "compliance operator" role typically has `can_view_reports` + `can_manage_compliance`; a "host owner" role typically has `can_manage_hosts` so they can enable or disable compliance on their own hosts. A pure auditor role with `can_view_reports` alone can see everything but cannot change anything.

> ***Note:*** *The release-notes shorthand of "can_view_reports and can_manage_hosts" doesn't quite line up with the handler: triggering a scan requires* `can_manage_compliance`*, not* `can_manage_hosts`*. Use the table above as the source of truth.*

The Two Scanners

The compliance integration on the agent (`patchmon-agent/internal/integrations/compliance/compliance.go`) runs two independent scanners. A "scan" as submitted to the server is actually an array of sub-scans, one per scanner that ran successfully.

1. OpenSCAP: CIS Benchmarks

What it is. OpenSCAP is the OS-level security compliance scanner. On supported Linux distributions it evaluates the host against the CIS Benchmark datastreams published by SCAP Security Guide (SSG). The agent picks the relevant `ssg-*-ds.xml` datastream for the host's OS and runs `oscap xccdf eval` against it.

Profile levels. Each datastream ships with two CIS-derived profiles:

- **CIS Level 1 Server** (`level1_server`): the baseline profile, intended for general-purpose systems with minimal operational impact. This is the default profile used for ad-hoc scans.
- **CIS Level 2 Server** (`level2_server`): the extended profile, for environments that require defence-in-depth (classified/regulated workloads). Some L2 rules impose real operational restrictions (e.g. disabling `wireless` where it's present).

The per-host **default profile** setting (Host Detail → Integrations → Compliance) controls which profile is used for scheduled scans. A manual scan can override the default by passing `profile_id` in the trigger request; the Host Detail Compliance tab exposes this as "Pick a profile".

Supported operating systems (as surfaced by the Compliance Settings panel):

OS	Profiles shipped in SSG
Ubuntu	CIS Level 1 Server, CIS Level 2 Server
Debian	CIS Level 1 Server, CIS Level 2 Server
RHEL	CIS Level 1 Server, CIS Level 2 Server
CentOS	CIS Level 1 Server, CIS Level 2 Server
Rocky Linux	CIS Level 1 Server, CIS Level 2 Server
AlmaLinux	CIS Level 1 Server, CIS Level 2 Server
Fedora	CIS Level 1 Server, CIS Level 2 Server
SLES	CIS Level 1 Server, CIS Level 2 Server
OpenSUSE	CIS Level 1 Server, CIS Level 2 Server

Any host OS not listed has no SSG datastream available and OpenSCAP scans will be skipped on it. The scanner is still "available" in the integrations metadata if `oscap` is installed; it just has nothing to evaluate.

> ***SLES and OpenSUSE:*** *the datastreams above ship with the server and the agent already maps SUSE-family hosts to the right profiles, but these rows are not usable yet in practice. SUSE hosts cannot complete enrolment because the agent has no zypper backend (see the package manager table earlier in this guide), so they never reach the point of running a scan. These profiles become available once zypper support lands.*

Default per-host state: OpenSCAP is **enabled by default** on every host that has compliance turned on. This is controlled by the `compliance_openscap_enabled` host flag, defaulted to `true` in 1.4.2 and preserved on upgrade.

2. Docker Bench for Security

What it is. Docker Bench is the container-host security scanner from the Center for Internet Security. It evaluates the Docker daemon, its configuration files, running containers, images, and Swarm configuration against the CIS Docker Benchmark (https://www.cisecurity.org/benchmark/docker). Rules are categorised into sections:

- Host Configuration
- Docker Daemon Configuration
- Docker Daemon Configuration Files
- Container Images and Build File
- Container Runtime
- Docker Security Operations

Docker Swarm Configuration

Results use a different status model from OpenSCAP: instead of pass/fail, most Docker Bench rules either **pass** or emit a **warning**. There are very few hard fails. The Compliance dashboard surfaces Docker Bench statistics separately in the "Docker Bench Analysis" section, with "Warnings by Section" charts instead of the severity-based ones used for OpenSCAP.

When it runs. Docker Bench runs only when **both** of the following are true:

The Docker integration is enabled on the host (the scanner reads the same Docker socket).

Docker Bench is enabled on the per-host scanner toggle.

If either is off, Docker Bench is skipped even if the binary is installed.

Default per-host state: Docker Bench is disabled by default on every host (per 1.4.2). You must explicitly toggle it on per host (Host Detail → Integrations → Compliance → Docker Bench). Most hosts do not run Docker, and running Docker Bench on a host without Docker produces a long list of misleading "Docker daemon not running" failures.

Per-Host Scanner Configuration

Every host that has compliance enabled exposes four compliance-related fields, manageable from the **Host Detail → Integrations → Compliance** panel or from the Compliance page's **Hosts** tab:

Field	Values	Meaning
`compliance_mode`	`disabled`, `on-demand`, `enabled`	Overall compliance switch for this host. `disabled` means the agent does not run any scanner. `on-demand` means scans run only when manually triggered. `enabled` means scans run on the fleet-wide `compliance_scan_interval`.
`compliance_openscap_enabled`	`true` / `false`	Whether OpenSCAP runs on this host. Default `true`.
`compliance_docker_bench_enabled`	`true` / `false`	Whether Docker Bench runs on this host. Default `false`.
`compliance_default_profile_id`	profile ID or null	The OpenSCAP profile used for scheduled / "all profiles" scans on this host. Null means the agent defaults to `level1_server`.

Changes to these fields are queued as **pending config** and pushed to the agent on the next heartbeat via the Apply Pending Config flow (see Managing the PatchMon Agent). They do not take effect until the agent confirms receipt.

Mode: disabled vs on-demand vs enabled

Disabled: nothing runs on this host. The scanner integration is marked off and the Compliance UI shows "Disabled" in the Mode column.

On-demand: the scheduled-scan path is disabled but manual **Run Scan** buttons still work. Use this when you only want to scan on investigation.

Enabled: the agent runs scheduled scans at the fleet-wide interval set in **Security Compliance → Settings → Scan Interval** (default 24 hours, configurable from 6 hours to 7 days).

The fleet-wide **default compliance mode** (Security Compliance → Settings → Default Compliance Mode) applies only to newly registered hosts. Existing hosts keep their current mode across server upgrades.

SSG Content Is Bundled in the Server Binary

This is one of the most important architectural changes in 2.0 for compliance.

In 1.x and earlier, each agent fetched SCAP Security Guide (SSG) content from GitHub at scan time. That required every agent to have outbound access to `github.com`, created occasional transient failures when GitHub was unavailable, and allowed agents to drift to different SSG versions depending on when they last pulled.

In 2.0, SSG and CIS benchmarking content is **bundled into the server image at build time** and served from a single `SSG_CONTENT_DIR` on the server. Agents now fetch content from the server itself, via two new endpoints:

`GET /api/v1/compliance/ssg-version`, returns the SSG version string and the list of `ssg-*-ds.xml` files available.

`GET /api/v1/compliance/ssg-content/{filename}`, streams a specific datastream file to the agent.

Both endpoints accept agent API-key authentication.

What this means operationally

No external network calls at scan time. The server is the only place an agent gets SSG content from. Agents never contact `github.com` for it under any circumstance, so air-gapped fleets need nothing beyond the server access they already had for heartbeats.

One SSG version across the fleet. Every agent gets the same content bundle. The Compliance Settings page shows the active version and the list of content files under **OpenSCAP Content**.

Version-pinned scanning. Because the image ships with the content, upgrading the server is the way to get new SSG rules. Hosts pick the new version up automatically on the daily

content check, and you can push it to a single host sooner with the API call described under "Upgrading SSG content on a host".

A server with no content fails loudly. If the content directory is empty, for example because a volume was mounted over it, the server logs a warning at startup, Compliance Settings reports the content as unavailable, and agents are told so explicitly rather than quietly falling back to some other source.

Where to see the active version

Security Compliance → Settings → OpenSCAP Content shows:

- The SSG version string (e.g. `0.1.77`).
- The number of content files bundled.
- A collapsible list of every `ssg-*-ds.xml` filename.
- The table of supported OSes and their profiles.

Where Compliance Lives in the UI

There are three ways in:

- **Security Compliance** (top-level sidebar): the fleet-wide view. Five tabs: Overview (dashboard), Hosts (per-host table with scan controls), Scan Results (drill into rules), History (chronological scan list), Settings (default mode, interval, SSG content).
- **Hosts → select a host → Compliance tab**: per-host drill-down with the same scan controls, latest scan summary, rule breakdown by status / severity / section, and a per-rule remediation action.
- **Dashboard → Compliance cards**: the main PatchMon dashboard includes a compliance summary card that deep-links into the Compliance page with the relevant filter applied. Hidden when the `compliance` module is disabled.

The top of every compliance view has five status cards: Total hosts, Compliant, Warning, Critical, Never scanned. Clicking "Never scanned" filters the Hosts tab to just the never-scanned subset so you can fix coverage gaps.

What a Scan Looks Like End-to-End

A typical ad-hoc scan flows like this:

- An operator clicks **Run Scan** on a host (either from the Compliance Hosts tab or from the Host Detail page).
- The browser calls `POST /compliance/trigger/{hostId}`. The server clears any stale "cancel" flag for this host in Redis, enqueues a `run_scan` task on the `compliance` asynq queue, and returns the job ID.

The task dequeues and sends a `run_scan` WebSocket command to the agent. The server updates the `compliance_scans` record to `running` status as soon as the agent confirms receipt.

The agent runs one or both scanners in sequence. OpenSCAP calls `oscap xccdf eval` against the SSG datastream; Docker Bench calls `docker-bench-security` (bundled with the agent).

Each sub-scan produces structured rule results. The agent batches them into a `CompliancePayload` and submits via `POST /api/v1/compliance/scans`.

The server's `ReceiveScans` handler validates API credentials, applies a 10-requests-per-minute rate limit, and writes the scan + results into the database in a single transaction. It honours the per-host `openscap_enabled` and `docker_bench_enabled` flags on the server side too, so accidental submissions from a scanner the host has disabled are rejected.

On success, the server emits a `compliance_scan_completed` notification event (with per-profile summaries) and, if any scan errored, a separate `compliance_scan_failed` event.

The UI's active-scan poll sees the row disappear from the `active_scans` endpoint, shows a toast "Compliance scan completed", and the dashboard refetches.

Scheduled scans go through the same `run_scan` → agent → `ReceiveScans` pipeline; only the initiator differs.

Stuck Scans and Auto-Cleanup

Compliance scans can take a long time: a full OpenSCAP L2 scan on a mid-size host can run 15–45 minutes, so PatchMon has an explicit stall detection threshold rather than a short timeout.

A scan is considered **stalled** if it has been in `running` status for more than **3 hours** without completing. A recurring asynq job (`ComplianceScanCleanup` , at `POST /api/v1/compliance/scans/cleanup`) runs periodically and moves every stalled scan to a terminal state with the error message `Scan terminated automatically after running for more than 3 hours` . This guarantees the **Scans in Progress** widget doesn't accumulate ghost scans and frees up the per-host "currently scanning" flag.

The `GET /compliance/scans/stalled` endpoint lets you see which scans are about to be cleaned up. The Compliance page exposes this via the stalled-scans widget (when any rows exist).

Related Documentation

Running Compliance Scans: triggering scans, bulk scan modal, cancelling, handling stuck scans.

Results and Remediation: reading the dashboard, drilling into host detail and rule detail, auto-remediation.

Docker Monitoring: Docker integration prerequisite for Docker Bench.

Managing the PatchMon Agent: Apply Pending Config flow used to push compliance toggles to agents.

Release Notes - 1.4.0: the release that introduced compliance scanning.

Release Notes - 1.4.2: per-host scanner toggles, scan cancel, 3h auto-cleanup.

Release Notes - 2.0.0: bundled SSG content and the rewrite.

Chapter 15: Running Compliance Scans

This page covers how to trigger a compliance scan, watch it progress, cancel it if needed, and set up scheduled scans across the fleet. All actions are from the web UI; everything runs against a logged-in session with the `compliance` module enabled.

You need `can_manage_compliance` to trigger, cancel, or install scanners; `can_view_reports` to watch progress without changing anything; and `can_manage_hosts` to change per-host compliance mode or scanner toggles.

Three Ways to Start a Scan

Entry point	Best for	Scope
Host Detail → Run Scan button	Investigating a single host	1 host, "all profiles" (whatever scanners are enabled for that host)
Security Compliance → Hosts tab → green Play button	Re-scanning a specific host from the fleet view	1 host, "all profiles"
Security Compliance → Scheduled, via fleet interval	Ongoing coverage across hosts where compliance mode is `enabled`	Every host with mode=`enabled`, runs periodically

Bulk ad-hoc scans across a selected host set are also supported. See Bulk Scans below.

Triggering a Scan on One Host

From the Host Detail page

Open **Hosts → *select the host* → Compliance** tab (also reachable via Security Compliance → host row → host name link).

Look at the top-right of the Compliance tab. You'll see:

A **Connected** or **Disconnected** pill: this is the agent's WebSocket connection status. Scans require a connected agent.

A **Run Scan** button (green with a Play icon).

Click **Run Scan**. The UI calls `POST /api/v1/compliance/trigger/{hostId}` with `profile_type=all` (run every scanner enabled for this host).

The button flips to a spinner with "Scanning..." and a toast confirms "Compliance scan triggered". The response includes a `jobId` you can correlate with server logs if needed.

If the agent is disconnected, the button is disabled and the tooltip reads "Host is disconnected". Re-enable connectivity (see Managing the PatchMon Agent) before retrying.

You can also pick a specific profile instead of running all scanners:

On the Host Detail Compliance tab, expand the profile selector (if visible for your role) and choose a profile, for example, `level2_server` instead of the default `level1_server`.

Tick **Enable Remediation** if you want the agent to apply OpenSCAP's remediation scripts for any failed rule during this scan. Remediation in-scan is destructive, only tick this when you've reviewed what the rules would do.

Click **Run Scan**. The request body includes `profile_type`, `profile_id`, and `enable_remediation`.

From the Security Compliance Hosts tab

Open **Security Compliance** → **Hosts** tab. You see a table of every compliance-enabled host.

Click the green Play button in the **Run** column for the host you want to scan. This has exactly the same effect as **Run Scan** on Host Detail, with `profile_type=all`.

Watch the row turn blue: the **Last activity** column shows an animated "OpenSCAP" / "Docker Bench" / "Scanning..." label while the scan is in progress.

The Play button turns into a red **StopCircle** button when the scan is active. Click it to cancel. See Cancelling a Scan below.

Watching Scan Progress

There is no live log stream for compliance scans (unlike patch runs). Instead, the UI relies on **active-scan polling**.

The active-scans widget

On the Compliance page Overview tab, when any scan is running, a blue **Scans in Progress** card appears with a spinner. Each running scan is shown as a pill with:

Host name (link to Compliance Host Detail).

Profile type badge (OpenSCAP or Docker Bench) and started-at timestamp.

Connection indicator (green Wi-Fi icon if the agent is connected, red if not).

The list also appears inline above the hosts table and refetches via `GET /api/v1/compliance/scans/active` every **30 seconds** while scans are active, and every **2 minutes** when idle. The dashboard uses the same cadence.

The pending-scans window

Between the moment you click Run Scan and the moment the agent updates the DB row to `running`, there's a few-second gap where the scan exists as an asynq task but not yet as a database row. The UI bridges this with **pendingScans** state: the just-triggered host shows up in the active-scans widget with a "Triggering..." status immediately, and is replaced by the real DB row once it lands (or removed after 60 seconds if no corresponding scan shows up).

Completion notifications

When an active scan disappears from the `/compliance/scans/active` response, the UI compares against the previous poll's active-scan set and shows a success toast:

"Compliance scan completed" for a generic completion.

"Scan completed for *host name*" for a tracked pending scan.

The dashboard and the history tab refetch automatically at this point.

Per-rule progress during scanner install

If the host doesn't have OpenSCAP or the CIS benchmark content installed yet, the first action is usually to install the scanner, which has its own progress model. See Installing the Scanner below.

Cancelling a Scan

A scan can be cancelled while it is running. Unlike patch runs, there is no "Stop Run" confirmation modal. Cancel is a one-click action because scanners are read-only and safe to interrupt.

From the Hosts tab

On the **Hosts** tab, find the row with the blue "Scanning..." indicator.

Click the red **StopCircle** button in the **Run** column. The UI calls `POST /api/v1/compliance/cancel/{hostId}`.

A toast confirms "Cancel request sent for *host*".

What cancel actually does

The `CancelScan` handler on the server does three things:

- **Removes any queued `run_scan` task from asynq**, so a scan that hasn't yet reached the agent won't start.
- **Sets a `compliance_scan_cancel` flag in Redis** for this host, so if the worker picks up the task between DeleteTask and the agent message, the worker sees the cancel flag and skips execution.
- **Sends a `compliance_scan_cancel` WebSocket message to the agent**, so an already-running scan is interrupted at the process level.

If the agent is connected and busy scanning, it receives the cancel, terminates the OpenSCAP / Docker Bench subprocess, and submits whatever partial results it has. The scan record is marked cancelled.

If the agent is offline, only the queue-level cancel applies: the scan won't run when the agent reconnects because the task has already been removed from the queue.

Cancel is idempotent. Calling it on a host with no active scan returns success with "Scan cancel sent".

Scheduled Scans

Scheduled scans are the "set and forget" path. Every host with `compliance_mode=enabled` gets a scan on the fleet-wide interval, no operator intervention required.

Fleet-wide defaults

Set from **Security Compliance → Settings**:

- **Default Compliance Mode**: applied to newly registered hosts only. Existing hosts keep whatever mode they already have.
  - `Disabled`, new hosts register with compliance off. You must explicitly enable per host.
  - `On-Demand`, new hosts register with scanning available but scheduled off. Manual `Run Scan` works.
  - `Enabled`, new hosts register ready for scheduled scanning.
- **Scan Interval**: how often `enabled` hosts scan. Presets: 6h, 12h, 24h (default), 48h, 3d, 7d. Also accepts a raw minutes value between 60 and 10080 (7 days).

Saving Settings pushes the new interval to every connected agent on the next heartbeat. Offline agents pick it up when they reconnect.

Per-host mode overrides

The default is advisory; every host has its own `compliance_mode`. To change it:

1. Open **Hosts → *host* → Integrations** tab.
2. Scroll to **Compliance**.
3. Pick **Disabled**, **On-Demand**, or **Enabled**.
4. The change is queued as **pending config** and applied via the Apply Pending Config flow on the host's next agent heartbeat.

The Compliance page's Hosts tab shows the current mode in the **Mode** column (`Disabled`, `On-demand`, `Scheduled`).

Per-host scanner toggles

On the same Integrations → Compliance panel you'll find two checkboxes:

- **OpenSCAP**: default `on`. Tick to have the agent run OpenSCAP CIS scans during each scheduled and on-demand scan.
- **Docker Bench**: default `off`. Tick only for hosts that actually run Docker and where the Docker integration is enabled.

These toggles are also reflected in the Hosts-tab **Scanners** column (`OpenSCAP`, `Docker`, `OpenSCAP, Docker`, or `-` if nothing is enabled).

Default profile

Set from **Host Detail → Integrations → Compliance → Default profile**. Pick between the available profiles exposed by the agent (typically `level1_server`, `level2_server`, and possibly `docker-bench` on hosts where Docker is present). This is the profile used for scheduled scans and for ad-hoc scans where no explicit profile is passed (`profile_type=all`).

Bulk Scans Across the Fleet

For ad-hoc "scan everything right now" operations, use the Bulk Scan modal (opened from the Compliance page; the exact entry point depends on your module / edition, usually a bulk action button on the Hosts tab).

The modal lets you:

1. Choose a **Profile Type**. `All Profiles`, `OpenSCAP Only`, or `Docker Bench Only`.
2. Tick **Enable Remediation** if you want OpenSCAP to apply remediation scripts during the scan.
3. Tick the hosts you want to include (or **Select All**).
4. Click **Scan N Hosts**.

The UI sends one `POST /api/v1/compliance/trigger/bulk` request with the full host list. The server enqueues one `run_scan` task per host. Hosts that are offline are still queued. They scan as soon as they reconnect and the worker dequeues their task (or the task is cleaned up if the queue drops it before reconnection).

The modal shows a results banner:

- Green if every trigger succeeded.
- Yellow if some failed, with the list of host names and the specific error per host.

After a successful bulk scan, the modal auto-closes after three seconds and each triggered host appears in the active-scans widget as pending / running.

Installing the Scanner

Compliance scanning on a host needs OpenSCAP (`oscap` binary) installed and the SSG content available locally. The first time you enable compliance on a host, the scanner usually isn't there yet. PatchMon handles this with an install job.

- Enable **compliance mode** on the host (set to `On-Demand` or `Enabled`).
- On next Apply Pending Config, the agent receives the new integration state and reports that the scanner is not installed.
- From the Host Detail Compliance tab, click **Install Scanner**. The UI calls `POST /api/v1/compliance/install-scanner/{hostId}` .
- The server enqueues an install task. The worker sends an install message to the agent.
- The agent installs the OpenSCAP scanner with the host's own package manager, then downloads SSG content from the server via `GET /api/v1/compliance/ssg-content/{filename}` . Progress events are reported back to Redis and surfaced in the UI via `GET /compliance/install-job/{hostId}` , which returns the current state plus a per-step message and progress percent. The states are `none` (no install has been requested), `waiting` , `active` , `completed` , `failed` , and `unknown` when the job can no longer be found. A `failed` state also carries an `error` field with the reason.
- When the install completes, the Run Scan button becomes active.

The scanner package name is not the same on every release, so the agent asks the archive which one it has rather than assuming. Debian 12 and newer, and Ubuntu 24.04 and newer, provide `openscap-scanner` plus `openscap-common` ; Debian 10 and Ubuntu 22.04 provide `libopenscap8` instead. RHEL-family hosts use `openscap-scanner` and SUSE hosts use `openscap-utils` .

Content is handled separately from the scanner, and a host that gets one without the other is not a failure. Ubuntu packages no SSG content at all, so on Ubuntu the datastream always comes from the PatchMon server. The install only fails outright when the host ends up with no working `oscap` binary.

Install can be cancelled mid-flight from the same UI via `POST /api/v1/compliance/install-scanner/{hostId}/cancel`.

Platforms where the scanner cannot be installed

Some supported PatchMon hosts cannot run compliance scanning at all, because the platform publishes no OpenSCAP package, no SCAP content, or neither. On most of these the install reports the reason in plain terms. The exception is Amazon Linux 2, where the failure surfaces as raw `yum` output:

Platform	Reason
Debian 11 (bullseye)	OpenSCAP was removed from the archive before bullseye released, so no package provides `oscap`
Amazon Linux 2	The amzn2 repositories carry neither `openscap-scanner` nor `scap-security-guide`
Alpine Linux	No SCAP datastream is published for Alpine, so a scanner would have nothing to scan against
Arch Linux	OpenSCAP is available only from the AUR, and no datastream is published
FreeBSD	No OpenSCAP port and no datastream
Windows	The agent has no compliance integration on Windows

Everything else in the supported matrix can install the scanner: Debian 10, 12, 13 and 14, Ubuntu 22.04 and 24.04, the RHEL family (Rocky, AlmaLinux, CentOS Stream, Oracle Linux, Fedora, Amazon Linux 2023), and openSUSE.

On Debian 10 the install succeeds but the results are not useful. PatchMon ships no Debian 10 datastream, so the host falls back to the very old content in buster's own `ssg-debian` package, and every rule comes back as "not applicable" because the content targets an earlier Debian release. Treat compliance scanning on Debian 10 as unavailable in practice.

If the status panel shows **Partial Installation**, some scanners are present and others are not. A common case is a host running Docker, where Docker Bench is ready but OpenSCAP is missing, so scans return Docker Bench results only and the scanner panel reports "No SCAP content found". The same button appears in that state, labelled **Retry install**, and installs whatever is missing.

Upgrading SSG content on a host

When the server is upgraded to a newer PatchMon version with newer bundled SSG content, existing hosts may still have older content cached locally.

You do not normally need to do anything. A daily check compares each host's reported SSG version against the version bundled with the server and pushes an upgrade to any host that is behind, so the fleet converges on its own within a day of a server upgrade.

To push it to one host sooner:

Call `POST /api/v1/compliance/upgrade-ssg/{hostId}` (requires the `can_manage_compliance` permission). The server enqueues an `ssg_upgrade` task and the agent downloads the current `ssg-*-ds.xml` for its OS from the server.

Poll `GET /api/v1/compliance/ssg-upgrade-job/{hostId}` for `waiting`, `active`, or `completed` with a message.

Re-running **Install Scanner** from Host Detail also re-syncs content from the server as part of the install.

The Compliance Settings page always shows the currently-active server-side SSG version under **OpenSCAP Content → SSG *x.y.z***.

Handling Stuck Scans

Any scan that has been in `running` status for more than **3 hours** is considered stalled. A recurring cleanup job (`ComplianceScanCleanup`, triggered at `POST /api/v1/compliance/scans/cleanup`) marks every such scan as cancelled with the error message:

> *Scan terminated automatically after running for more than 3 hours*

This prevents orphaned "forever running" scans from clogging the active-scans widget. The cleanup runs on a schedule driven by the recurring automation queue; administrators with `can_manage_compliance` can also trigger it on demand from the Automation UI.

Seeing stalled scans

The `GET /api/v1/compliance/scans/stalled` endpoint returns every scan older than 3 hours that is still marked `running`. If the Compliance page shows a **Stalled Scans** widget (rendered when any stalled rows exist), clicking a row links into the host's compliance detail so you can inspect it.

Why a scan might legitimately stall

The agent crashed mid-scan. There was no opportunity to submit a terminal result.

The agent lost network connectivity mid-scan. Results were generated but not submitted; by the time connectivity returned, the 3-hour window had passed.

A profile on an unusually large host simply exceeded 3 hours (uncommon for L1, possible for L2 with heavy file-integrity rules on deep filesystems). The cleanup will fire; re-run the

scan manually afterwards.

What the operator should do

If a scan was cleaned up automatically and you need results:

- Check agent health (`sudo patchmon-agent diagnostics` on the host, or review the host's recent logs from the Host Detail page).
- If the agent is healthy, **Run Scan** again from the Host Detail or Compliance Hosts tab.
- If scans reliably take more than 3 hours on a specific host (typically an extra-large file server), consider switching that host to an on-demand schedule so it only scans when you're actively watching.

Rate Limits

Agent-side submission of scan results is rate-limited to **10 submissions per minute per host** at `POST /api/v1/compliance/scans`. Legitimate use never hits this: a host only submits once per scan. The limit exists to contain a misbehaving agent that tries to re-submit results in a loop.

Server-side scan triggers are not individually rate-limited beyond the general-auth rate limits you configure for the API, but asynq's worker pool naturally paces execution: a fleet-wide "scan everything now" will queue 100+ tasks and work through them at a sensible rate.

Related Documentation

- Compliance Overview: module gate, permissions, scanner architecture, bundled SSG content.
- Results and Remediation: what to do with scan results once they land.
- Docker Monitoring: the Docker integration you need for Docker Bench scans.
- Managing the PatchMon Agent: diagnostics and the Apply Pending Config flow used to push compliance settings.

Chapter 16: Compliance Results and Remediation

Once a compliance scan completes, results appear in three layers of the web UI: the fleet-wide **dashboard**, the per-host **Compliance tab**, and the per-rule **Rule Detail** page. This page walks through each layer in operator terms, then covers the optional auto-remediation paths and the compliance trends view.

You need `can_view_reports` to see any of this; `can_manage_compliance` to trigger remediation.

Fleet-Wide Dashboard

The **Security Compliance** landing page opens on the **Overview** tab, which is the fleet-wide dashboard. It's designed to answer "how is my overall compliance posture, and where do I look first?" in a single glance.

The five summary cards

Across the top of every Compliance page view sit five identical cards:

Card	What it counts	Derived from
Total hosts	`total_hosts + unscanned` : every host with compliance visibility, scanned or not	`summary.total_hosts` + `summary.unscanned`
Compliant	Hosts whose latest scan score is **≥ 80%**	`summary.hosts_compliant`
Warning	Hosts whose latest scan score is between **60% and 79%**	`summary.hosts_warning`
Critical	Hosts whose latest scan score is **< 60%**	`summary.hosts_critical`
Never scanned	Hosts that have never successfully submitted a scan	`summary.unscanned`

The Never scanned card is clickable, it toggles a filter on the **Hosts** tab to show only never-scanned hosts, which is the fastest way to find coverage gaps.

Six charts

The Overview tab grid has five charts:

Failures by Severity: stacked doughnut of critical / high / medium / low failed rules across the fleet. Clicking a slice deep-links into the Scan Results tab filtered to that severity.

OpenSCAP Distribution: split of pass / fail rules across OpenSCAP scans.

Compliance Profiles: pie of scans by profile type (OpenSCAP vs Docker Bench). Clicking drills into the matching filter.

Last Scan Age: distribution of when hosts were last scanned (today / this week / this month / older).

Host Compliance Status: bar chart of hosts by Compliant / Warning / Critical / Never Scanned.

All charts refetch every 2 minutes (or every 30 seconds when there's at least one active scan) so the dashboard stays useful for an operator who leaves the page open during a rollout.

Profile-type filter

Above the charts is a **profile type** filter with three values: `All Scans` (default), `OpenSCAP`, `Docker Bench`. Switching to OpenSCAP or Docker Bench reveals additional profile-specific panels:

> **OpenSCAP Analysis**: Rule Results doughnut, Failures by Severity bar, Score Distribution, Scan Freshness.
>
> **Docker Bench Analysis**: Rule Results doughnut (pass vs warning), **Warnings by Section** bar (broken down into the CIS Docker Benchmark sections), Score Distribution, Scan Freshness.

The Docker Bench view intentionally uses "Warnings" instead of "Failures" because Docker Bench's status model is pass-or-warning for most checks, not pass-or-fail.

Hosts Tab: Per-Host Fleet View

The **Hosts** tab shows a row per compliance-enabled host. This is your work list, sort and scan from here.

Column	What it shows
Run	Green Play button (trigger scan) or red Stop button (cancel scan).
Host name	Friendly name (clickable, opens Compliance Host Detail).
Status	Shield icon colour-coded: green ≥80%, yellow 60-79%, red <60%, grey never-scanned.
Last activity	Friendly date of last scan and activity label ("Scan", or "Scanning..." while active).
Passed / Failed / Skipped	Click the count to deep-link into the Scan Results tab, filtered by this host and that status.
Scanner status	`Scanned`, `Enabled`, or `-`; whether the agent has actually produced a scan, has the scanner enabled without results, or has no scanner integration active.
Mode	`Scheduled`, `On-demand`, or `Disabled`: this host's `compliance_mode`.
Scanners	Which scanners are enabled per host: `OpenSCAP`, `Docker`, `OpenSCAP, Docker`, or `-`.

Clicking any Passed / Failed / Skipped number pivots you straight into the Scan Results tab with the host and status filters applied, so "click the 12 Failed for hostA" gets you a filtered rule list for that host's latest scan.

The page also respects the **tableFilter** override driven by the `Never scanned` summary card, click that card and the table filters to hosts with no scan records.

Scan Results Tab: Rule Drill-Down

The **Scan Results** tab (also reachable via the per-host clicks described above) is where you investigate specific rules across the fleet. It shows every rule that has been evaluated by any scanned host, with:

- **Rule title**, **rule reference** (e.g. `xccdf_org.ssgproject.content_rule_...`), **section** (CIS section number for OpenSCAP, bench section for Docker Bench).
- **Severity** badge (critical / high / medium / low / unknown).
- **Profile type** badge (OpenSCAP / Docker Bench).
- **Hosts passed / failed / warned / total** across the fleet.

Filters above the table: status (pass / fail / warn / error / skipped), severity, profile type, specific host, and full-text search. These all push down to `GET /api/v1/compliance/rules` so the filtering is server-side and consistent with the dashboard drill-down links.

Click any rule to open **Rule Detail**.

Rule Detail: Single Rule Across the Fleet

The Rule Detail page (`/compliance/rules/{ruleId}`) is what you open when you want to understand "what is this rule, why is it failing, how do I fix it, and which hosts does it affect?".

It has four sections:

1. Summary cards

Four cards at the top: **Affected Hosts**, **Passing**, **Failing**, **Warnings**, counts across the fleet for the rule's latest scan per host.

2. Description

The human-readable explanation from the benchmark, expanded in full. For OpenSCAP rules this is the SCAP `<Description>` element; for Docker Bench it's the rule prose from the benchmark.

3. Why this failed (Rationale)

The benchmark's rationale for why this rule exists, why it matters, what risk it mitigates. Shown as plain text.

4. What the fix does + Remediation

The right column contains two panels that help operators act on a failure:

- **What the fix does**: a short plain-English explanation of the remediation, derived heuristically from the remediation text (for example, "This fix will update file permissions or

ownership..." when the script uses `chmod` / `chown`; "This fix will update SSH daemon configuration..." when the script touches `/etc/ssh`). This is UI scaffolding, not audit-grade. Always read the actual remediation script below.

Remediation: the exact script the scanner would run. For OpenSCAP rules this is the shell fix pulled from SSG. For Docker Bench it's the benchmark-prescribed command. A **Copy** button copies the script to your clipboard so you can paste into a change ticket or runbook.

If the benchmark doesn't ship a remediation script (common for high-level "document this" rules), the panel reads "No remediation steps available."

5. Affected Hosts table

Every host that has evaluated this rule shows up here with:

Host name (click to open Compliance Host Detail).

Status for this rule on this host (Pass / Fail / Warning / N/A / Skipped / Error).

Why (this host): either the scanner's `finding` text, or a `Current: X → Required: Y` string built from `actual` + `expected`. This is the concrete reason why *this* host failed, which is usually enough to diagnose without opening a shell.

Use this view to scope impact: "this rule fails on 14 hosts; is it the same root cause on all of them?" Sort by the status column, look for clusters of identical finding text, and fix them as a batch.

Compliance Host Detail

The per-host compliance view (`/compliance/hosts/{hostId}`) is reached by clicking any host name across Compliance. Its layout:

Header: back link, Shield icon, host name as H1, **Run Scan** button with connection-status pill, link to Full Host Details.

Five summary cards: Passed, Failed, Warning, N/A, and Score (or recent scan metadata).

Scan Results table: paginated at 25 rows per page, drilled into the most recent scan for this host by default, filterable by status and severity.

Inline rule actions: each failed rule row has an expand button that shows the Why / Rationale / Remediation inline, plus a "Remediate this rule" button (see below).

Filters on the five summary cards: click the **Passed rules** card to filter results to pass-only, click **Failed** for fail-only, etc. The card selected gets a coloured ring so you know the active filter.

The scan shown is the latest per-profile. A profile-type filter above the table lets you flip between the latest OpenSCAP scan and the latest Docker Bench scan for the host. If the latest scan is older than a week, a soft warning reminds you that the results may be stale.

Auto-Remediation

There are two remediation paths, each driven from a different part of the UI.

1. Per-rule on-demand remediation

Fix a single failed rule without running a full scan.

1. Open **Compliance Host Detail** for the host.
2. In the Scan Results table, expand a failed rule.
3. Click **Remediate this rule**.
4. The UI calls `POST /api/v1/compliance/remediate/{hostId}` with `{ "rule_id": "<rule-ref>" }`. The server validates that the agent is connected and sends a `remediate_rule` WebSocket message to the agent.
5. The agent runs `oscap xccdf eval --remediate --rule <rule>` against the SSG datastream, that runs OpenSCAP's targeted remediation script for just that one rule.
6. The UI shows a toast "Remediation triggered". The next scan (manual or scheduled) should show that rule flipping from Fail to Pass if the fix was successful.

Per-rule remediation is limited to OpenSCAP rules today. Docker Bench does not ship executable remediation scripts in the benchmark. The UI greys out the Remediate button on Docker Bench rules for this reason.

2. In-scan remediation

Apply remediation scripts for *every* failing rule as part of a scan.

1. From the **Run Scan** dialog on Host Detail, tick **Enable Remediation** before starting the scan.
2. From the **Bulk Compliance Scan** modal (Compliance page), tick **Enable Remediation** before triggering the batch.

When `enable_remediation=true` is set on the trigger, the agent runs OpenSCAP in `--remediate` mode, which attempts the remediation fix for every rule that fails. The scan results submitted back to the server include a `remediation_applied` / `remediation_count` summary so you can tell the difference between a regular scan and a remediating scan.

When to use which

Per-rule: tightly-scoped changes, especially on production where you want to see exactly one thing change at a time. Safer, slower.

In-scan: bulk cleanup on a newly-built host, or a lab environment you just rebuilt and want to harden in one pass. Faster, but applies every fix, review the affected rule set first.

> ***Warning:*** *Some OpenSCAP remediation scripts are destructive. They can change SSH configuration, disable protocols, modify PAM settings, or set kernel parameters that break unrelated tooling. Always test in-scan remediation on a non-production host before rolling it across the fleet. Per-rule remediation is safer because you've read the script first.*

Release-note lineage

Auto-remediation was introduced in 1.4.0 ("Optional auto-remediation of failed rules during scans"). In 2.0 it remains under the Compliance module, it did **not** move into the Patching module. If you're reading older release notes, per-rule remediation still runs through `POST /api/v1/compliance/remediate/{hostId}`, not through a patch run.

Trends Over Time

`GET /api/v1/compliance/trends/{hostId}?days=30` returns the host's scan history as a time series: `completed_at`, `score`, `profile_name`, `profile_type` for each scan in the window. The UI uses this to show trend lines on Compliance Host Detail.

The API supports `days` between 1 and 365. Typical use is the default 30 days for day-to-day monitoring, or 365 when writing an annual compliance report.

History Tab: Chronological Scans

The **History** tab is a flat list of every scan the system has ever recorded, newest first. Paginated at 25 per page, filterable by status, profile type, and host.

Each row shows:

- Host name.
- Profile (e.g. `level1_server` OpenSCAP, or `Docker Bench for Security`).
- Started at and duration.
- Totals: total rules, passed, failed, warnings, skipped, not applicable.
- Score and any error message.

Scans that were auto-cancelled after the 3-hour stall threshold appear here with the error message "Scan terminated automatically after running for more than 3 hours" and a status of cancelled, useful for spotting hosts that consistently time out.

There is no export endpoint for scan history. To archive scans for regulators, call `GET /api/v1/compliance/scans/history` directly and write the JSON to disk.

Notifications for Scans

Every completed scan emits a `compliance_scan_completed` notification event. The notification body includes:

- Fleet-friendly title: `Compliance Scan - <hostname>` (`- N Failed Rules` suffix when failures are present).
- Per-profile summary lines: profile name, score as a percentage, passed count, failed count.
- Structured metadata for downstream processing: host ID, host name, failed count, passed count, total rules, profile summaries.

Default severity is `informational`, escalated to `warning` when there's at least one failed rule. The per-event alert settings let you override severity or suppress these if you already have a dashboard.

A separate `compliance_scan_failed` event is emitted for each sub-scan that errored during a multi-scanner run (e.g. OpenSCAP succeeded but Docker Bench failed). Default severity is `error`. Metadata includes the profile name, profile type, and the captured error.

Practical Workflow

A typical compliance cycle in PatchMon looks like:

- **Baseline**, turn compliance on across the fleet (`Default Compliance Mode = On-Demand`, then enable per host) and bulk-scan everything once to build the baseline. Expect a lot of failures; that's the starting point.
- **Triage**, open the Overview tab. Use **Failures by Severity** to find critical failures; click through to the Scan Results tab filtered to critical.
- **Investigate**, on each rule, open Rule Detail. Read the rationale, read the remediation, pick a handful of identical-finding hosts and fix them manually or via per-rule remediation.
- **Rescan**, on each fixed host, click Run Scan. Confirm the rule flipped to Pass.
- **Enable scheduling**, once the baseline is clean, switch `compliance_mode=enabled` on the hosts that care, set the scan interval to 24h (or whatever suits your SLO), and leave it running. The dashboard becomes your ongoing signal for drift.
- **Revisit**: the **Scan Freshness** chart on the OpenSCAP / Docker Bench tabs tells you which hosts haven't scanned recently; bring those back into the loop.

Related Documentation

- Compliance Overview: scanner architecture, permissions, module gate.
- Running Compliance Scans: triggering, cancelling, scheduling, stuck-scan handling.
- Docker Monitoring: the Docker integration prerequisite for Docker Bench scans.

Alerts and Notifications: routing `compliance_scan_completed` and `compliance_scan_failed` events to your destinations.

Release Notes - 1.4.0: introduction of auto-remediation.

Chapter 17: Alerts Overview

What alerts are in PatchMon

An **alert** is a record of a noteworthy condition detected by the server: a host that has stopped reporting, a security-updates threshold being crossed, a new PatchMon server version becoming available, and so on. Alerts appear in **Reporting** and can be routed to chat, email, or ntfy through the notifications pipeline. The same event that creates an alert can also be sent to external destinations. Alerts are the in-app representation of operational signals.

Alerts are grouped into categories in Settings and the UI: **host**, **patching**, **compliance**, **docker**, **security**, **remote_access**, and **system**.

The global master switch lives under **Reporting → Alert Lifecycle → Alerts system**. When this is off, no alerts are created at all, regardless of per-type configuration.

Alert types implemented today

The following alert types fire from the server code in 2.0. Each can be individually enabled, tuned, and routed.

Type	Category	Fired when
`host_down`	host	The host's agent WebSocket has been disconnected for longer than the `host_down` threshold (default 30 seconds, configurable in **Reporting → Alert Lifecycle**), or the host has not reported within that threshold during a periodic check. Displayed as **Host Agent Down**.
`host_recovered`	host	A previously-down host's agent reconnects or its agent starts reporting again. Displayed as **Host Agent Recovered**.
`host_enrolled`	host	A new host is successfully enrolled
`host_deleted`	host	A host is removed from the inventory
`host_security_updates_exceeded`	security	A host has more security updates than the configured threshold
`host_pending_updates_exceeded`	security	A host has more pending updates than the configured threshold
`host_security_updates_resolved`	security	Security updates count drops below threshold again
`host_pending_updates_resolved`	security	Pending updates count drops below threshold again
`server_update`	system	A newer PatchMon server version is detected via the DNS version check
`agent_update`	system	A newer agent version is released
`patch_run_started`	patching	A patch run begins
`patch_run_completed`	patching	A patch run finishes
`patch_run_failed`	patching	A patch run exits with errors
`patch_run_approved`	patching	A patch run is approved for execution
`patch_run_cancelled`	patching	A patch run is cancelled by an operator
`patch_reboot_required`	patching	Packages requiring a reboot were installed
`compliance_scan_completed`	compliance	An OpenSCAP compliance scan finishes
`compliance_scan_failed`	compliance	A compliance scan errors out

Type	Category	Fired when
`container_stopped`	docker	A tracked Docker container stops unexpectedly
`container_started`	docker	A previously-stopped container starts again
`container_image_update_available`	docker	A newer image digest is available for a tracked container
`ssh_session_started`	remote_access	A user opens a web SSH session to a host
`rdp_session_started`	remote_access	A user opens a web RDP session to a host
`user_login`	system	A user signs in
`user_login_failed`	system	A failed sign-in attempt
`account_locked`	system	An account is locked after repeated failures
`user_created`	system	A new user is created
`user_role_changed`	system	A user's role is changed
`user_tfa_disabled`	system	A user's two-factor authentication is removed

If a type listed in the release notes is not in this table, it is not implemented in the server. Anything absent from the `alert_config` table is treated as enabled by default for backwards compatibility, but only types with emitters in the server code actually fire.

The Reporting page

Alerts are managed from **Reporting** in the main navigation. The page has a fixed header with four severity cards (Informational, Warning, Error, Critical) plus a **Total Active** card, and a tab bar underneath.

Tabs

Tab	Purpose
Overview	Dashboards: alerts by severity, volume trend, alerts by type, recent alerts, responder workload, deliveries by destination
Alerts	The filterable table of open and historical alerts
Alert Lifecycle	Per-type configuration (gated by the `alerts_advanced` module)
Destinations	Notification destinations (SMTP, webhook, ntfy, internal)
Event Rules	Routing rules that fan events out to destinations
Scheduled Reports	Cron-scheduled fleet reports delivered to destinations
Delivery Log	Every outbound notification attempt, with status and errors

Clicking any severity card jumps straight to the **Alerts** tab filtered to that severity and status = `open`.

Filters

The **Alerts** tab supports four filters in addition to a free-text search box:

Filter	Values
Severity	`All Severities`, `Informational`, `Warning`, `Error`, `Critical`
Type	`All Types` or any alert type that exists in your alert history
Status	`All Status`, `Open`, `Acknowledged`, `Investigating`, `Escalated`, `Silenced`, `Done`, `Resolved`
Assignment	`All Assignments`, `Assigned to me`, `Assigned`, `Unassigned`, or a specific responder

Filters persist in the URL (`?tab=alerts&severity=critical&status=open`), so you can deep-link directly to a filtered view. The severity cards in the header also highlight when a filter is active.

Filtering and searching are applied by the server across every alert, not just the alerts on the page you are looking at, so a search always returns matches from the whole history. The search box waits briefly after you stop typing before it runs, so results settle rather than flickering on every keystroke.

Sorting

Three columns in the alerts table are sortable by clicking the header: **Severity**, **Type**, and **Created**. The arrow icon next to the column header indicates the current sort direction. Sorting is applied across the whole filtered set, so the first page always shows the true top of the sort.

Pagination

The alerts table is paginated. Below the table you will find:

Rows per page: 25, 50, 100, or 200. Your choice is remembered in the browser.

A count of the rows shown and the total number of alerts matching the current filters.

Previous and next page controls, with the current page number.

Changing a filter, the search text, the sort, or the page size returns you to page one. Any row selection is cleared when you change page or filter, so a bulk action can only ever apply to alerts you can see.

Alert lifecycle

PatchMon tracks alerts with two concepts:

`is_active`: a boolean on the alert row. An alert is **active** when it is open or still being worked on, and **inactive** once it has been resolved.

Current state: a label derived from the most recent action recorded against the alert (e.g. `acknowledged`, `resolved`).

Actions

The available actions are driven by a database table rather than being hardcoded, so the list may vary slightly between deployments. Actions split into two groups:

Workflow actions keep the alert active and just record progress. Typical names:

`acknowledged`
`investigating`
`escalated`
`silenced`

Resolution actions close the alert: they set `is_active=false`, record `resolved_at` and `resolved_by`, and move the alert out of the active stats. Typical names:

`resolved`
`done`

Running a resolution action on an already-resolved alert is safe. Running a workflow action on a resolved alert re-activates it (this is how "reopen" works in practice: pick a workflow action like `acknowledged`).

Workflow actions appear under **Workflow** and resolution actions under **Resolve** in both the row menu and the alert details modal.

Assignment

Alerts can be assigned to a user from three places:

- The **Assigned To** dropdown on the alerts table: changes the assignment inline.
- The **Assigned To** dropdown in the alert details modal.
- The **Auto-assign** column in **Alert Lifecycle**: sets a default assignee for all new alerts of a given type.

Choose **Unassigned** to clear. Every assignment change is written to the alert history.

History

Every action (created, assigned, unassigned, acknowledged, resolved, and any custom action) is recorded in `alert_history`. Open the alert details modal and scroll to **History** to see who did what and when. System-driven actions (e.g. `host_recovered` auto-resolving a `host_down` alert) are recorded with user "System".

Bulk actions

Select one or more alerts using the checkboxes in the **Alerts** tab to reveal a bulk-actions bar above the table. You can:

- Apply any workflow or resolution action to every selected alert in one call.
- **Delete** the selected alerts permanently.

Bulk updates stream through the same history recording as individual actions. Deleting an alert does not leave a history trail; use a resolution action if you want to keep the audit record.

The number of selected alerts is shown on the left. Use the checkbox in the table header to select or deselect every row on the current page. Selection does not carry across pages: to act on more alerts at once, raise the rows-per-page setting first.

Per-alert-type configuration

Each alert type has its own row in **Reporting → Alert Lifecycle**. This tab is gated by the `alerts_advanced` module; plans without it show an upgrade prompt here.

Each row exposes:

Column	Meaning
Active	Master switch for this alert type. When off, no alerts of this type are created and no notifications are emitted.
Severity	Default severity applied to new alerts of this type.
Alert delay	Seconds to wait before delivering the outbound notification. If a cancelling counterpart event (e.g. `host_recovered` for `host_down`) fires within the delay window, the notification is suppressed. Useful for flappy hosts.
Frequency	For periodic checks only (`host_down`, `host_security_updates_exceeded`, `host_pending_updates_exceeded`). Minutes between checks.
Threshold	For threshold alerts. Numeric threshold above which an alert fires. Units depend on the alert type: `host_security_updates_exceeded` and `host_pending_updates_exceeded` use a *count* (number of pending updates); `host_down` uses *seconds* (how long the agent's WebSocket can be disconnected before the alert fires). The `host_down` row shows a `sec` suffix to make this explicit; default is 30 seconds.
Auto-assign	Toggle plus user picker: any new alert of this type is assigned to the chosen user automatically.
Retention	Days to keep alerts of this type before cleanup. Empty = never auto-clean.
Auto-resolve	Days after which active alerts auto-resolve if no one touches them.

Changes are staged locally. Use the **Apply** button on the top bar to save them, or **Discard** to revert. The browser warns you if you navigate away with unsaved changes.

Cleanup

Below the table, the **Alert cleanup** card runs the retention policy:

> **Preview cleanup** shows the list of alerts that would be deleted under the current retention and auto-resolve rules.
>
> **Delete N alerts** commits the preview. The action is irreversible.

Cleanup only deletes alerts that satisfy `retention_days`. Whether it also deletes unresolved alerts is governed by **cleanup_resolved_only** per type (default: resolved only).

Permissions

Permission	What it grants
`can_manage_alerts`	Create/modify alert configurations, run cleanup, act on alerts
`can_manage_notifications`	Create, edit, test, and delete destinations, routes, and scheduled reports
`can_view_notification_logs`	Read the **Delivery Log** tab
`can_view_hosts`	List host groups and hosts when building routes and reports

Admins and superadmins bypass these checks. Regular users without `can_manage_alerts` can still view the **Alerts** tab but cannot perform actions.

Related pages

- Notification Destinations
- Notification Routes and Delivery Log
- Scheduled Reports
- Host Agent Down and Host Agent Recovered Alerts

Chapter 18: Notification Destinations

What a destination is

A **destination** is an endpoint that outgoing notifications are sent to: an SMTP mailbox, an HTTP webhook URL, an ntfy topic, or the built-in **Internal Alerts** destination that records alerts inside PatchMon itself.

Destinations are the "where" half of the notifications pipeline. The "what goes there" half is handled by **event rules**, covered in Notification Routes and Delivery Log.

Destinations live under **Reporting → Destinations** in the web UI.

Channel types

PatchMon 2.0 ships four destination channel types. The list is fixed in the server code:

Channel	Value	What it does
Webhook	`webhook`	HTTP POST of a JSON payload to any URL. Generic by default; Discord, Slack, Mattermost and Rocket.Chat webhook URLs are auto-detected and formatted with the appropriate rich payload.
Email	`email`	SMTP delivery to one or more recipients, with HTML body and an optional attachment for scheduled reports.
ntfy	`ntfy`	Push notification via ntfy.sh (https://ntfy.sh) or a self-hosted ntfy server.
Internal Alerts	`internal`	Built-in destination that drops events into the **Alerts** tab. You cannot create or delete this one; it is created automatically and can only be enabled or disabled.

> ***Discord is a webhook, not a channel type.*** *To post alerts to a Discord channel, create a* ***Webhook*** *destination with the Discord webhook URL. The separate* ***Settings → Discord Authentication*** *area is only for Discord OAuth2 sign-in; it is unrelated to notifications.*

Permissions

Creating, editing, testing, and deleting destinations requires the `can_manage_notifications` permission. Admins and superadmins bypass the check. Users without the permission do not see the **Destinations** tab at all.

Creating a destination

1. Open **Reporting → Destinations**.
2. Click **Add destination**.
3. Pick a channel type (Webhook, Email, or ntfy) and click **Next**.
4. Give the destination a **Display name**: this is what appears in the event rules picker, the delivery log, and scheduled report selectors.
5. Fill in the channel-specific configuration (see below).
6. Leave **Enabled** on (default) or turn it off to save the configuration without sending anything yet.
7. Click **Create**.

A successfully-created destination shows up in the destinations table with its channel icon, display name, and enabled switch.

Webhook

Pick this for **generic JSON webhooks, Discord, Slack, Mattermost or Rocket.Chat**. Discord, Slack, Mattermost and Rocket.Chat URLs are auto-detected and sent rich payloads; other URLs receive a generic JSON body.

Field	Required	Notes
Webhook URL	Yes	Full HTTPS URL. Discord: `https://discord.com/api/webhooks/...`. Slack: `https://hooks.slack.com/services/...`. Mattermost: `https://your-mattermost/hooks/...`. Generic: any endpoint that accepts `POST` with `Content-Type: application/json`.
Signing secret	No	Optional HMAC secret. When set, each webhook is signed with SHA-256 over the payload; the signature is sent in a header so the receiver can verify authenticity.

> ***Mattermost and Rocket.Chat.*** *Both are self-hosted on your own domain, so PatchMon cannot detect them by hostname. It detects them by the* `/hooks/<token>` *path that both use for incoming webhooks, and sends the Slack-compatible payload they expect. A URL such as* `https://chat.example.com/hooks/xxxgeneratedkeyxxx` *is matched, and so is one behind a subpath proxy, for example* `https://example.com/mattermost/hooks/xxxgeneratedkeyxxx`*. The match requires one or two long opaque tokens after* `/hooks/`*, so an automation platform that happens to use a* `/hooks/` *path for routing (a Zapier catch hook, for example) is not mistaken for a chat server and keeps the generic body. Jira and Confluence automation URLs on* `automation.atlassian.com` *are excluded by name, because they are shaped like a chat webhook but are consumed as structured data.*

The generic JSON body also carries a top-level `text` key, alongside the structured `event_type`, `severity`, `title`, `message`, `reference` and `metadata` fields. Receivers that read the structured fields are unaffected, and any receiver that requires a `text` key works without extra configuration. Note that `text` is formatted in Slack-style markdown, so it contains `*bold*` markers and Slack link syntax such as `<https://…|View in PatchMon>`; a receiver that renders it as literal text will show that markup. On scheduled reports the `text` key is a short preview only, because the full report is already present in the `html` and `csv` fields.

Email (SMTP)

Field	Required	Notes
SMTP host	Yes	e.g. `smtp.example.com`, `smtp.sendgrid.net`.
SMTP port	No	Defaults to `587`. Use `465` for implicit TLS, `25` for unencrypted relay (avoid).
Username	No	SMTP auth user. Leave blank if your relay does not require it.
Password	No	SMTP auth password. Stored encrypted.
From	Yes	Envelope + header `From` address, e.g. `patchmon@example.com`. Must be accepted by the relay.
To	Yes	Comma-separated list of recipients.
TLS mode	Yes	Choose how the SMTP transport secures the connection. See **TLS modes** below. Defaults to **STARTTLS** for new destinations.

TLS modes

PatchMon offers four TLS modes on every email destination. Pick the one your relay actually supports rather than leaving it on **Auto**, so a misconfigured server fails closed instead of silently downgrading to plaintext.

STARTTLS (recommended). PatchMon connects in plaintext on the SMTP port (typically 587) and then requires the server to advertise `STARTTLS`. The connection is upgraded to TLS before any credentials or message body are sent. If the server does not advertise `STARTTLS`, PatchMon refuses to send and reports the failure. This is the right choice for the vast majority of modern relays (Microsoft 365, Google Workspace, SendGrid, Postmark, Mailgun, Amazon SES on port 587, and most on-prem mail servers).

Implicit TLS / SSL. PatchMon opens a TLS connection from the very first byte, with no plaintext handshake. The default port for this mode is 465. Use it when your relay only accepts TLS on a dedicated port and does not support `STARTTLS`. Some legacy or appliance-based servers only offer this mode.

None (insecure). Cleartext SMTP, no TLS at any stage. If a username or password is set on the destination, PatchMon refuses to send unless you also tick **Send credentials over an unencrypted connection**, because authenticating in cleartext puts the credentials on the wire. See **Authenticated relays without TLS** below. Use this mode only for trusted internal relays on a private network where TLS is genuinely unavailable.

Auto. Legacy opportunistic mode kept for backward compatibility. PatchMon tries `STARTTLS` first and falls back to implicit TLS on the same host and port if `STARTTLS` is not advertised. Existing destinations that were saved before the explicit modes were added load as **Auto** so they keep working unchanged. Open the destination, pick **STARTTLS** or **Implicit TLS / SSL** explicitly once you have confirmed which one your relay supports, and save. New destinations should not be configured as **Auto**.

Authenticated relays without TLS

Some internal relays require SMTP AUTH but do not offer TLS at all. By default PatchMon refuses that combination, because PLAIN authentication over cleartext puts the username and password on the wire in a form anyone on the network path can read.

If the relay is on a network you trust and you accept that trade-off, set **TLS mode** to **None (insecure)** and tick **Send credentials over an unencrypted connection** on the destination. PatchMon then authenticates in cleartext for that destination only.

Points worth knowing before you enable it:

- The setting is per destination, not global. Other email destinations are unaffected.
- It applies only in **None (insecure)** mode. In **STARTTLS**, **Implicit TLS / SSL**, and **Auto** it is ignored, so it can never weaken a mode that does use TLS.
- It is off by default, including on every destination that already exists.
- Anyone able to observe traffic between PatchMon and the relay can read the credentials. Use a dedicated relay account with no privileges beyond sending mail, and never reuse a password from elsewhere.
- The preferred fix remains enabling STARTTLS on the relay, even with a self-signed or internal CA certificate.

> *Port and mode are independent. The port field is just the TCP port to connect to; the TLS mode controls how the connection is secured. The defaults (587 for STARTTLS, 465 for implicit TLS) match the conventional ports, but you can override the port if your relay listens elsewhere.*

Send test email

Saved email destinations have a **Send test email** button next to the standard **Test** action. Unlike **Test**, which enqueues a synthetic notification through the worker, **Send test email** performs a synchronous live SMTP probe directly from the API request and reports the result inline:

- On success the toast confirms delivery and the recipients should receive a short test message.
- On failure PatchMon reports which stage of the SMTP exchange failed: `validate` (the configuration is rejected before any network activity, for example a missing host, or a username set with TLS mode **None** without the cleartext credentials opt-in), `dial` (the TCP connection or implicit TLS handshake could not be established), `starttls` (the server did not advertise `STARTTLS` in the chosen mode), `auth` (the relay rejected the credentials), or `send` (the relay accepted the session but rejected the recipients or message). The toast includes the underlying error message returned by the relay.

This is the fastest way to diagnose a TLS or auth misconfiguration without trawling through server logs. The probe respects the same `can_manage_notifications` permission as editing the destination.

> *The probe always uses the **last saved** configuration, not what is currently on screen. Save the destination before testing, otherwise you are testing the previous settings. This matters most when you have just ticked **Send credentials over an unencrypted connection**: until you save, the probe still fails at the `validate` stage.*

ntfy

Field	Required	Notes
Server URL	No	Leave empty for `https://ntfy.sh`. Fill in your own URL for self-hosted ntfy.
Topic	Yes	ntfy topic name. Subscribe to the same topic on your phone/desktop to receive push notifications.
Access token	No	ntfy bearer token for protected topics. Alternative to basic auth.
Username / **Password**	No	Basic auth. Use instead of access token when your ntfy server is configured for HTTP basic auth.

Internal Alerts

You cannot create this destination; it is seeded automatically with the ID `internal-alerts` and appears with the **Built-in** tag. Its sole job is to write events into the in-app **Alerts** tab. You can:

- **Enable / disable** it from the destinations table (disable if you do not want internal alert records at all, for example when you only use external chat or email).
- Reference it from event rules, same as any other destination.

You cannot delete it. Attempting to delete returns `400 Bad Request: The Internal Alerts destination cannot be deleted. You can disable it instead.`

Editing a destination

Click **Edit** in the destinations table. The modal re-loads the current configuration (secrets included, so you do not have to re-type passwords or tokens) and lets you change any field. Click **Save** to apply.

The **enabled** switch is inline in the table; click it to toggle without opening the modal.

Secrets are always encrypted at rest using PatchMon's `SESSION_SECRET`. When you re-enter a secret and save, the value is re-encrypted. The decrypted value is returned only to operators with `can_manage_notifications`.

Testing a destination

Use **Test** in the destinations row to verify the configuration without waiting for a real event:

Click **Test** next to any enabled non-built-in destination.

PatchMon enqueues a synthetic event with type `test`, severity `informational`, and the message *"This is a test message from PatchMon notification settings."*

A toast confirms the test is **enqueued**. Actual delivery happens through the notifications worker and takes a second or two.

The **Delivery Log** updates automatically after about three seconds; look there for the outcome.

> *Tests do **not** bypass global rate-limiting. If your destination is already at its per-minute rate cap (60 messages/minute), the test returns `429 Too many notifications; try again shortly`.*

Failure cases returned by the test endpoint:

HTTP	Message	Meaning
`400 Bad Request`	`Destination is disabled`	Enable it first.
`404 Not Found`	`Destination not found`	The destination was likely deleted in a parallel tab.
`429 Too Many Requests`	`Too many notifications; try again shortly`	Rate limit hit.
`503 Service Unavailable`	`Notifications not configured`	Background worker or Redis is not running.

Deleting a destination

Click the trash icon in the destinations table. The confirmation dialog warns that the action is permanent. Deleting a destination does **not** delete the event rules that target it: those routes will be orphaned and should be updated to point at a different destination or removed. Deliveries in the **Delivery Log** keep their historical `destination_id` and appear as a raw ID if the name can no longer be resolved.

You cannot delete the `internal-alerts` destination (see above).

What gets stored

Every destination is a database row with:

A stable UUID (`id`) used by event rules and the delivery log.
`channel_type` from the list above.
`display_name`.
`enabled` boolean.

`config_encrypted`: the JSON configuration, encrypted with `SESSION_SECRET` so a database dump does not expose SMTP passwords, ntfy tokens, or HMAC secrets.

`created_at` / `updated_at` timestamps.

The list endpoint never returns the raw config, only a `has_secret` flag. The decrypted config is fetched on demand from a separate endpoint when the edit modal opens.

Discord sign-in vs Discord webhooks

The two "Discord" areas in PatchMon are independent:

Area	Purpose	Where
Discord Authentication	OAuth2 sign-in, users log in to PatchMon with their Discord account, optionally requiring membership of a server and role.	Settings → Discord Authentication
Discord webhook destination	Post alerts into a Discord channel via a channel webhook URL.	Reporting → Destinations → Add destination → Webhook

Configure them separately. The OAuth2 settings are not required to send alerts to Discord.

Related pages

- Alerts Overview
- Notification Routes and Delivery Log
- Scheduled Reports

Chapter 19: Notification Routes and Delivery Log

Overview

In PatchMon, a **route** (labelled **Event Rule** in the UI) connects one or more event types, and optionally a severity floor, a host scope, or a match rule, to a **destination**. When an event fires, the notifications engine evaluates every enabled route; each matching route produces a delivery to its destination.

Routes handle fan-out: one `host_down` event can notify your on-call ntfy topic, post to a Discord #alerts channel, and write an internal alert record, all from a single emit.

Both routes and the **Delivery Log** live under **Reporting** in the main navigation:

- **Reporting → Event Rules**: create, edit, and disable routes.

Reporting → Delivery Log: every outbound delivery attempt, sent or failed.

Permissions

Action	Permission
Create / edit / disable / delete routes	`can_manage_notifications`
Read the delivery log	`can_view_notification_logs`

Admins and superadmins bypass these checks.

Creating a route

Go to **Reporting → Event Rules**.

Click **Add event rule**. (Disabled until at least one destination exists. Create one first under Notification Destinations.)

Fill in the modal:

Field	Notes
Destination	Required. Pick from the list of configured destinations. You can only route to enabled destinations; disabled destinations are skipped at delivery time.
Events	Tick **All events** to match every event type, or tick individual events. Selecting every individual event collapses back to "All events".
Minimum severity	Floor for the route. Events below this severity are ignored. Order is `informational < warning < error < critical`.
Host groups	Optional. If any are selected, only events whose host is a member of at least one of the groups match. Leave empty for "any host".
Individual hosts	Optional. If any are selected, only events for those specific hosts match. Leave empty for "any host".
Enabled	On by default. Turn off to keep the rule for later without it firing.

Click **Add**.

Event type reference

Pick from the same set documented in Alerts Overview. `host_down`, `host_recovered`, `patch_run_completed`, `ssh_session_started`, and so on. You can also select high-volume or low-volume events like `user_login` and `account_locked` to route sign-in telemetry.

Host group and host filters combined

If both **host groups** and **individual hosts** are set, the event must satisfy **both** filters. In practice you usually pick one or the other, not both.

Events without a host context (e.g. `server_update`, `user_created`) are filtered out by **any** host scope you add. Leave both scope fields empty to match those as well.

Severity, delay, and lifecycle

Per-type **Alert delay** in **Alert Lifecycle** applies *before* the route fan-out: if the event has a configured `alert_delay_seconds`, PatchMon enqueues the delivery with that delay. If a counterpart event fires within the window (for example, `host_recovered` while a delayed `host_down` is queued), the delayed notification is cancelled. Counterpart mapping:

Delayed event	Cancelled by
`host_down`	`host_recovered`
`container_stopped`	`container_started`
`host_security_updates_exceeded`	`host_security_updates_resolved`
`host_pending_updates_exceeded`	`host_pending_updates_resolved`

Editing and deleting routes

Each row in **Event Rules** has **Edit** and **Delete** buttons.

Edit reopens the modal with the saved values. Save to update; the new criteria take effect for the next matching event.

Delete removes the rule entirely. Deliveries already enqueued finish, but no new deliveries are produced.

Disabled routes are displayed with a muted **Disabled** badge and do not receive deliveries. Disable is the safer option if you want to pause a rule temporarily.

How matching works

For each outgoing event, the server:

Looks up all routes whose `event_types` include the event type (or the wildcard `*`).

Drops routes whose **destination is disabled**.

Drops routes whose `min_severity` is above the event's severity.

For each remaining route, applies the host-group and host-ID filters.

Deduplicates: events that repeat within a 2-minute window for the same destination are collapsed into one delivery. The fingerprint key is `event_type + reference_id + destination_id + 2-minute bucket`.

Rate-limits: each destination is capped at **60 deliveries per minute**. Deliveries over the cap are dropped with a warning in the server log.

Enqueues an asynq task to the `notifications` queue with `MaxRetry=5`.

The queue worker then dispatches the delivery according to the destination's channel type (SMTP send, HTTP POST, ntfy publish, or internal alert write).

The Delivery Log

The **Delivery Log** tab shows every outbound notification attempt with its result. Use it when a destination is not receiving messages, a webhook recipient reports errors, or you want an audit trail of what went where.

Columns

Column	Meaning
Time	When the delivery was processed, shown as a relative time ("5m ago"). Hover for the exact timestamp.
Status	`sent` for success (green), anything else for failure (red).
Event	The event type that produced the delivery (e.g. `host_down`, `patch_run_failed`).
Destination	The destination display name at the time of delivery. Shows the UUID if the destination has been deleted.
Reference	`reference_type:reference_id`, clickable for `host`, `patch_run`, and `alert` references so you can jump to the source.
Error	Error message returned by the delivery attempt. Empty for successful deliveries.

Pagination

The log is paginated at 50 rows per page. Use the left and right arrows at the bottom to move through history. The most recent deliveries are on page 1.

Use the **Refresh log** button in the page header to pull the latest entries without navigating away.

Retries

The notifications worker retries failed deliveries up to **5 times** with exponential back-off (handled by asynq). Each attempt is recorded on the same delivery log row: the `attempt_count` field increments, and the row is upserted with the latest `status` and `error_message`. The provider message ID (e.g. SMTP queue ID, webhook `Message-ID`) is captured in `provider_message_id` when the remote end returns one.

If all five retries fail, the delivery row stays at the last `failed` state. There is no automatic escalation; diagnose the failure from the **Error** column.

Common failure reasons

Error (excerpt)	Likely cause
`connect: connection refused` / `i/o timeout`	Destination host is unreachable from the PatchMon server. Check firewall / network.
`authentication failed` / `535 5.7.8`	Wrong SMTP credentials or token. Re-edit the destination and re-enter.
`400 Bad Request` from Discord/Slack webhook	Webhook URL is wrong, revoked, or the rich payload is malformed for a customised Slack app.
`webhook status 400` from Mattermost or Rocket.Chat	The webhook was deleted or the token is wrong, or the target channel no longer exists. Re-copy the URL from the integration page. Versions before 2.1.3 returned this for every delivery regardless of the URL; upgrade if you are on an older release.
`403 Forbidden` from ntfy	Topic requires auth you have not provided, or token is expired.
`destination disabled`	Someone disabled the destination between enqueue and delivery. Re-enable and re-trigger.

If an expected entry is missing, check that the route is enabled, the destination is enabled, the event passed the severity and scope filters, and the alert type itself is enabled in **Alert Lifecycle**.

Deduplication and rate-limiting in the log

Duplicates suppressed by the 2-minute dedup window do **not** appear in the delivery log; they are silently skipped before a delivery task is created. Rate-limited deliveries are also skipped silently (a warning goes to the server log, not the delivery log). If a destination suddenly stops receiving events, check:

- The destination is enabled.
- No route has been deleted.
- The per-minute rate cap is not being exceeded upstream. 60 messages/minute is per-destination.

Diagnosing a webhook with `LOG_LEVEL=debug`

The delivery log records the outcome of a delivery, not how it was built. To see which payload format a webhook destination actually received, set `LOG_LEVEL` to `debug` (**Settings > Environment**, or the `LOG_LEVEL` env var). Every webhook delivery then writes one line:

```
level=DEBUG msg="webhook dispatch" destination_id=... event_type=host_down
  format=slack_compatible host=chat.example.com body_bytes=229 signed=true
```

Field	Meaning
`format`	`discord`, `slack_compatible` or `generic`. This is the payload shape that was sent, so it tells you whether auto-detection recognised your receiver.
`host`	Host of the destination URL. The path is deliberately omitted because it carries the webhook's secret token.
`body_bytes`	Size of the JSON body, useful when a receiver enforces a size limit.
`signed`	Whether an HMAC signature header was attached.

If `format` is `generic` for a chat server you expected to be recognised, the URL is not on a `/hooks/<token>` path. The delivery still works, because the generic body carries a `text` key, but the message will be less richly formatted. The destination form shows the same detection result before you save.

Debug level is verbose across the whole server, so turn it back to `info` once you have the answer.

App links in notifications

Every notification includes an `app_link` in its metadata pointing back to the most relevant page in PatchMon:

Reference type	Link
`patch_run`	`/patching/runs/<id>`
`host`	`/hosts/<id>`
`alert`	`/hosts/<host_id>` if known, otherwise `/`
`user`	`/settings/users`
`test`	`/reporting`

Formatters for each channel render this as a clickable button (Discord/Slack rich embeds), an `<a>` tag (email), or a `Click` action (ntfy).

Related pages

- Alerts Overview
- Notification Destinations
- Scheduled Reports

Chapter 20: Scheduled Reports

Overview

A **scheduled report** is a periodic fleet summary that PatchMon renders to HTML (with a CSV attachment) and delivers through one or more notification destinations on a cron schedule. Use them to keep leadership and on-call teams informed about compliance posture, patching throughput, pending updates, and open alerts, without anyone needing to log into the UI.

Scheduled reports are managed under **Reporting → Scheduled Reports**. They share the same destinations as event-driven notifications, so any email, webhook, or ntfy destination you have already set up can receive reports too.

Permissions

Creating, editing, running, and deleting scheduled reports requires `can_manage_notifications`. Admins and superadmins bypass the check. Users without the permission do not see the tab.

To include host-group scoping, the user must also have `can_view_hosts` (so the group picker can populate).

Creating a report

1. Open **Reporting → Scheduled Reports**.
2. Click **New report**. (Disabled until at least one destination exists. Create one under Notification Destinations.)
3. Fill in the modal:

Field	Notes
Report name	Required. Shown in the table and as the email subject prefix.
Schedule	Frequency + time of day. See Schedule options.
Sections	Which blocks to include in the rendered report. See Report sections.
Deliver to	Tick every destination that should receive this report. You can send the same report to multiple destinations.
Scope to host groups	Optional. Limit the report's per-host sections to the selected host groups. Leave empty for fleet-wide.
Top rows per section	Numeric cap on per-host lists, defaults to **20**.
Enabled	On by default. Disable to keep the report saved but paused.

Click **Create**.

After creation, the report appears in the table with its next run time, status badge, and action buttons.

Schedule options

The modal composes a standard five-field cron expression for you, so you rarely see cron syntax directly. Frequencies and the resulting cron:

Frequency	Cron produced	What it means
Daily	`M H * * *`	Every day at the chosen time.
Weekdays (Mon to Fri)	`M H * * 1-5`	Mondays to Fridays at the chosen time.
Weekly	`M H * * D,D,…`	The chosen days of the week. Pick one or more via the Mon/Tue/… toggle buttons.
Monthly	`M H D * *`	The chosen day of the month (`1st` , `15th` , `Last day` , or a custom day `1–31`).

All schedules evaluate in the **server timezone** configured in PatchMon settings; the modal labels this next to the time picker. Changes to the server timezone after the report is saved do **not** automatically re-schedule existing reports. Edit the report and save again to re-evaluate.

The schedule is displayed on the table in plain English ("Daily at 08:00", "Weekdays at 09:30", "15th of month at 06:00"), computed from the underlying cron.

Report sections

Each report is a composition of **sections**, ticked independently:

Section	Content
Executive summary	Total hosts, average compliance score, critical hosts, compliant hosts, plus a patching overview (runs, completed, failed, running).
Compliance summary	Passed rules, failed rules, critical hosts, hosts with no recent scan.
Recent patch runs	Latest patch runs by status with timestamps and target counts.
Hosts / status	Host status rollup: offline, stale, active.
Open alerts	Currently active alerts grouped by severity.
Hosts by outstanding updates	Top hosts sorted by pending updates (respects the **Top rows per section** cap).
Top outdated security packages	Packages with the most hosts needing a security update.

New reports default to **Executive summary + Compliance summary + Recent patch runs** unless you customise the selection.

Delivering a report

Every tick in **Deliver to** adds a destination to the report's fan-out. At run-time, PatchMon:

1. Resolves the destinations (skips disabled ones).
2. Renders the HTML body and CSV attachment once.
3. Sends the same payload to each destination in parallel.

For each channel type the payload adapts:

Destination	What the recipient sees
Email	HTML email rendered inline; CSV attached. Subject contains the report name and timestamp.
Webhook	JSON POST with report metadata, a summary, and the HTML body in a field. Use this to fan reports into a downstream system (data warehouse, Google Sheets ingester, etc.).
ntfy	Short push notification with a link back to the latest report in the UI. The full HTML does not fit ntfy, so it is summarised.
Internal Alerts	A system record under the **Alerts** tab, useful when you want a run history inside PatchMon without email.

A report's appearance in the **Delivery Log** uses `event_type: scheduled_report`. Filter the log by the report's destinations to audit deliveries.

Running a report manually

Click the green **Play** button in the report's row to run it immediately. The report is queued for instant execution and delivered to the configured destinations.

Manual runs respect the same destination state: disabled destinations are skipped, and rate limits still apply.

Disabled reports show the play button greyed out. Enable the report (or edit and tick **Enabled**) before running. The button tooltip tells you why it is unavailable.

Editing and deleting

Edit reopens the same modal pre-filled with the current schedule, sections, and destinations. Saving re-computes the next run time.

Delete removes the report permanently. Past deliveries in the log remain.

Enabled switch: edit the report and toggle **Enabled** in the modal. Disabled reports keep their schedule but do not fire until re-enabled; their next-run time is still displayed.

How scheduling works internally

Scheduled reports are stored in the `scheduled_reports` table. On create or update, PatchMon computes the next run via the cron expression in the server's timezone and writes it to `next_run_at`. The scheduler enqueues the report task to asynq at exactly that time, with no background polling loop.

When the task executes, the worker:

Re-reads the report row.

Aborts if it has been disabled since enqueue.

Renders HTML + CSV via the server's report renderer (see `internal/notifications/report_render.go`).

Fans out to each destination with the same fingerprint + rate-limit + retry semantics as regular notifications.

Updates `last_run_at` and queues the next occurrence.

Because the schedule is stored as a cron string plus a timezone, daylight-saving transitions are handled by the cron library. Jobs that would fall in a skipped hour are pushed to the next valid slot; jobs repeated in a duplicate hour fire once.

Known limits

The scheduled-report pipeline does **not** attempt full re-delivery of a whole report's fan-out on transient failure. A delivery that fails retries per-destination (up to 5 times via asynq), but the render is not re-done. In practice this means a report either reached each destination successfully (with retries covering transient issues) or ended up in the delivery log as `failed` for that destination.

There is no "skip next run" option. To skip a single run, disable the report before its scheduled time, then re-enable it afterwards.

Report templates are not customisable from the UI in 2.0. The rendered HTML layout is fixed; customise by choosing sections and host-group scope. Custom templates are a candidate for a future release.

Related pages

Alerts Overview

Notification Destinations

Notification Routes and Delivery Log

Chapter 21: Web SSH Terminal

Overview

PatchMon ships an in-browser SSH terminal that lets operators connect to any monitored Linux/FreeBSD host without leaving the web UI. The terminal is a full xterm with line editing, colours, scrollback, resize, and keyboard shortcuts, powered by a WebSocket between the browser and the PatchMon server.

Two connection modes are supported:

Direct: the PatchMon server dials the host's SSH port (22 by default) and bridges the session. Use this when your server has network reach to the hosts.

Proxy: the PatchMon server asks the host's own agent to open a local SSH connection (to `localhost:22` on the host) and tunnels it back through the agent's existing outbound WebSocket. **No inbound SSH port exposure required on the target host.**

Authentication to the host uses an SSH password or an SSH private key (with an optional passphrase). Authentication to PatchMon itself is handled by your existing session cookies plus a one-time **ticket** described below.

Web SSH is shipped in PatchMon from 1.4.0 onwards; in 2.0 it is provided under the **remote_access** capability module.

Permissions

Role	Web SSH access
admin / **superadmin**	Always granted.
Any other role	Requires `can_use_remote_access` on the role permissions.

Users without `can_use_remote_access` attempting to open the terminal are rejected with HTTP `403 Access denied` during the WebSocket handshake.

The **Hosts** related permission (`can_manage_hosts`) is required to see the "Open Terminal" control on the host detail page in the first place.

Opening a terminal

1. Go to **Hosts** and click the host you want to connect to.
2. On the **Host Detail** page, open the **Terminal** tab (or click the **SSH Terminal** button in the header).
3. Pick the **Connection mode** (Direct or Proxy).
4. Enter the SSH **username** (defaults to `root` ; the last-used username per host is cached in your browser's local storage).
5. Choose an **Authentication method**:
   - **Password**: type the host password.
   - **Key**: paste the private key (OpenSSH or PEM format) and the passphrase if encrypted.
6. Adjust the **SSH port** if needed (default `22`).
7. If you picked **Proxy** mode, set the **Proxy host** (default `localhost`) and **Proxy port** (default `22`). These are the destination the agent will dial, typically `localhost:22` when you want the agent to SSH into its own host.
8. Click **Connect**.

Once the green *"SSH connection established"* line appears, the terminal is live and interactive.

Your SSH credentials are **never stored** by the server or browser. They are sent over the authenticated WebSocket once at connect time and held in browser state only for the life of the session. Disconnecting clears them from memory.

Direct mode

In Direct mode, the PatchMon server dials the host directly:

1. Browser → `POST /api/v1/auth/ssh-ticket` with `{ "hostId": "<id>" }` . Requires your PatchMon session cookie. Returns a 30-second, single-use ticket.
2. Browser opens `wss://<patchmon-host>/api/v1/ssh-terminal/<hostId>?ticket=<ticket>` .

Server consumes the ticket (deleted from Redis on use), validates the user is active and has permission, and upgrades to WebSocket.

Browser sends the `connect` message with auth credentials, terminal size, and connection mode.

Server dials `host.ip` (falling back to `host.hostname`) on the chosen port, authenticates with password or private key, and starts an interactive shell.

Host key verification: the server uses `~/.ssh/known_hosts` on the PatchMon container if it exists, and falls back to `InsecureIgnoreHostKey` otherwise. Direct mode does **not** prompt the user to accept host keys. Keys are accepted on first use when the fallback is active. Supply a `known_hosts` file via volume mount for production deployments that require strict verification.

Use Direct mode when:

The PatchMon server has network reach to the host on the SSH port.

You accept bridging SSH via the server host rather than via the host's agent.

Proxy mode

Proxy mode routes the SSH session through the host's existing agent WebSocket, avoiding the need to expose an SSH port inbound to PatchMon.

Flow:

Browser → ticket + WebSocket as in Direct mode.

Server receives the `connect` message with `connection_mode: "proxy"`.

Server generates a 16-byte session ID, stores a proxy session record, and sends `{ "type": "ssh_proxy", "session_id": …, "host": "localhost", "port": 22, "username": … }` over the agent's existing WebSocket.

The agent dials `<proxy_host>:<proxy_port>` (defaults `localhost:22`) **on its own host** and pipes the stream back to the server over the WebSocket as `ssh_proxy_data` frames.

The server forwards those frames to the browser as terminal `data` events.

Agent config requirement. Proxy mode requires `integrations.ssh-proxy-enabled: true` in the agent's `/etc/patchmon/config.yml`. This setting is not pushed from the server. It has to be set manually and the agent service restarted. If the agent rejects the request, the terminal shows *"Agent not connected"* or an agent-supplied error.

Use Proxy mode when:

The host has no inbound SSH exposure (behind NAT, in a restricted VPC, behind a corporate firewall).

You already trust the agent's outbound connection to PatchMon and want to reuse it.

You want to SSH to `localhost` through the agent without punching holes through the edge firewall.

One-time tickets for WebSocket auth

WebSocket upgrades cannot include the normal authentication cookies reliably across all browsers, and passing long-lived tokens via query parameters would expose them in server logs and browser history. PatchMon avoids both problems with **one-time tickets**:

- Tickets are 64 hex characters, generated from `crypto/rand` on the server.
- Tickets live in Redis with a **30-second TTL**.
- A ticket carries the user ID and the host ID it was minted for.
- The WebSocket handler **consumes** the ticket on first use (atomic `DEL`). A second attempt to open a WebSocket with the same ticket fails with `Invalid or expired ticket`.
- Ticket validation also verifies the `hostId` in the URL matches the one encoded in the ticket. Stolen tickets cannot be reused against a different host.

You get the ticket implicitly by clicking **Connect** in the UI; there is no operator-visible ticket string.

Keyboard and terminal interactions

The embedded xterm supports the usual shortcuts:

Action	Shortcut
Copy selection	Browser-standard (Ctrl+Shift+C / Cmd+C)
Paste	Ctrl+Shift+V / Cmd+V
Send Ctrl+C to the remote	Ctrl+C (when no selection)
Scrollback	Mouse wheel or trackpad
Clear screen	Remote `clear` command

The terminal automatically resizes when the PatchMon browser window resizes, the AI Assistant panel opens/closes, or the sidebar collapses. The server is notified over the WebSocket so the remote TTY keeps `cols` and `rows` in sync. Resize events are honoured in Direct mode (when supported by the remote SSH server) and in Proxy mode via `ssh_proxy_resize` messages to the agent.

Terminal output is also captured in a rolling 5 000-character buffer for the AI Terminal Assistant, if enabled.

Session lifetime and idle timeout

- **Ticket TTL**: 30 seconds. A session that takes longer than that to start must re-request a ticket.

Idle disconnect: after **15 minutes** with no terminal activity the session is closed automatically. A visible warning appears 1 minute before the disconnect. Any input or output resets the timer.

Manual disconnect: click **Disconnect** in the toolbar or close the terminal panel. Credentials are wiped from browser state on disconnect.

What happens when the WebSocket drops

If the server-side SSH process exits (e.g. you type `exit` on the remote shell), the terminal shows *"SSH connection closed"* and the WebSocket stays open for a potential new `connect`.

If the WebSocket itself drops unexpectedly and you were connected, the terminal attempts to **reconnect once** after 3 seconds, via a brand-new ticket and WebSocket. Authentication re-uses the cached username but you must re-enter password or key, as credentials are not persisted in the browser.

Close codes that are **not** retried: `1000` (normal close), `1006` (abnormal close, often auth failure), `1008` (policy violation). For those, you get *"Connection failed: Session may have expired. Please refresh the page or log in again."*.

Auditing

Every successful ticket mint (i.e. the user has requested a terminal session) fires an `ssh_session_started` event:

Severity: `informational` (configurable in **Alert Lifecycle**).

Metadata: `host_id`, `host_name`, `user_id`.

Reference: the host record.

Route this event type to a destination (for example, a `#security` Discord channel) if you want a live audit trail of all web SSH sessions. Configure routing in Notification Routes and Delivery Log.

Server logs also record each upgrade and ticket consumption under `ssh-terminal connected` and `ssh-terminal ticket invalid` log lines.

Troubleshooting

Symptom	Likely cause and fix
"Authentication required. Please log in again." when clicking **Connect**	Your PatchMon session cookie is missing or expired. Refresh the page and sign in.
"Invalid or expired ticket" on upgrade	More than 30 seconds elapsed between ticket mint and WebSocket open, or the ticket was already consumed. Retry; PatchMon mints a new ticket on the retry.
"Agent not connected." in Proxy mode	Host's agent WebSocket is down. Verify from **Host Detail → Status**; restart the agent service on the host.
Agent rejects with *"ssh-proxy-enabled must be true"*	Set `integrations.ssh-proxy-enabled: true` in the agent's `config.yml` and restart the agent service.
"Failed to parse private key"	Key is encrypted: add the passphrase. Or the key format is unsupported; use OpenSSH or PEM PKCS#8.
Connection established but first-time host key warning on server log	The PatchMon container has no `known_hosts` for this host. Add one via a volume mount, or accept that first-use keys are auto-trusted.

Related pages

- AI Terminal Assistant
- RDP via Guacamole
- Managing the PatchMon Agent
- Agent Configuration Reference

Chapter 22: RDP via Guacamole

> ***Known issue (2.0.0).*** *The RDP connection flow has a known bug in PatchMon 2.0.0. Sessions may fail to establish, disconnect early, or return opaque errors in certain environments. A fix is planned for the next release. See Release Notes 2.0.0 for details. If RDP is mission-critical for your rollout, validate the workflow in a staging instance before relying on it in production.*

Overview

PatchMon 2.0 lets you open a full RDP session to a **Windows host** from your browser, with no RDP client installed locally and no inbound RDP port exposed to the outside world. The session travels:

From the browser as a Guacamole WebSocket tunnel, into the PatchMon server.

From the PatchMon server into a `guacd` sidecar, which speaks the RDP protocol.

From `guacd` through a short-lived TCP proxy to the host's own PatchMon agent, which forwards to `localhost:3389` on the Windows host.

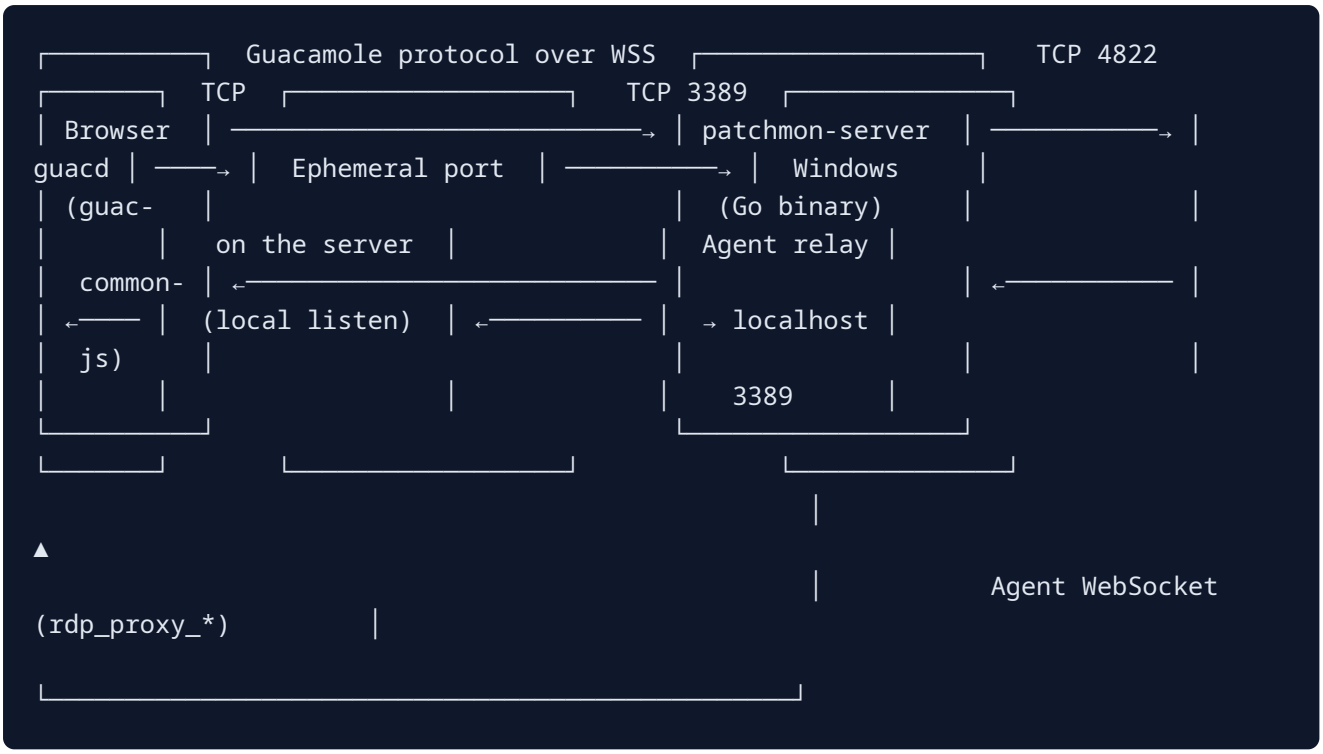
This is a one-time ticketed connection with keyboard, mouse, and clipboard support. Screen size is configurable, and NLA/TLS/legacy RDP is auto-negotiated.

RDP is provided under the **remote_access** capability module.

Known issue: 2.0.0 RDP bug

> ***Before using RDP in production, read this.***
>
> *Version 2.0.0 has a known bug in the RDP connection flow that can cause:*
>
> - *Sessions to fail handshake with generic errors.*
> - *Ticket resolution issues that surface as "invalid or expired ticket" on otherwise valid sessions.*
> - *Early disconnects after a successful handshake under some network conditions.*
>
> *A fix is planned for the next release. In the meantime:*
>
> - *If RDP is critical, stay on the last 1.4.x release that works for you, or retry the session.*
> - *Always verify the **Web SSH Terminal** works end-to-end first. It has no such known issue and is a good baseline check for connectivity and agent health.*
> - *When reporting RDP issues, include the **server log** lines tagged `rdp-ticket` and `rdp tunnel`, and the agent's `rdp_proxy_*` log lines.*
>
> *Full context: Release Notes 2.0.0, section **Known issues**.*

Architecture



Key components:

- **`guacd`**: Apache Guacamole's daemon, shipped as a sidecar container in PatchMon's Docker Compose (`guacamole/guacd:1.6.0`). Runs on `4822/tcp` inside the `patchmon-internal` network. No public ports.
- **PatchMon server**: acts as the Guacamole WebSocket tunnel endpoint and owns the RDP ticket store. It asks the host's agent to set up a local TCP proxy, then hands that proxy to `guacd`.
- **Agent proxy**: on receiving `rdp_proxy` over its WebSocket, the agent opens a local TCP bridge and forwards bytes between PatchMon and `localhost:3389` on the Windows host. Requires `integrations.rdp-proxy-enabled: true` in the agent config.
- **Windows host**: runs the standard Windows RDP service on `127.0.0.1:3389` (bound to localhost via the agent; no inbound exposure needed).

See Installing PatchMon Server on Docker for the sidecar configuration as deployed by the standard compose file. If you run PatchMon without the sidecar, install `guacd` separately (`apt install guacd` / `yum install guacd`) and set `GUACD_ADDRESS` to point at it.

Permissions and module

Access	Requirement
Open an RDP session	**admin**, **superadmin**, or `can_use_remote_access` + `can_view_hosts` on your role
Create RDP ticket for a host	`can_manage_hosts` (needed to see the control on the host detail page in the first place)
Deployment	`remote_access` capability module enabled

Users without the required permission are rejected at `POST /auth/rdp-ticket` with `403 Access denied`.

Prerequisites

Before you can open an RDP session to a host, all of these must be true:

- The host is identified as **Windows** in PatchMon (`os_type` or `expected_platform` contains "windows"). Non-Windows hosts are rejected with `400 RDP is only available for Windows hosts`.
- The host's **PatchMon agent is online** and connected via its WebSocket.
- The agent's `config.yml` has `integrations.rdp-proxy-enabled: true`. This setting is **not** pushable from the server. You edit it on the host and restart the `PatchMonAgent` service.
- RDP is enabled on the Windows host, and the agent's user context can reach `localhost:3389`. (The default NLA mode is fine; PatchMon negotiates security automatically.)
- The PatchMon server is able to reach `guacd` at its configured address (defaults to `127.0.0.1:4822` or `guacd:4822` depending on deployment). If not, RDP ticket creation fails early with `503 guacd is not reachable on the PatchMon server`.

Opening an RDP session

- Go to **Hosts** and click the Windows host.
- On **Host Detail**, open the **Remote Access** area and click **Open RDP** (or the RDP icon in the toolbar).
- Enter the Windows **username** and **password** for the account to sign in as.
- Optionally adjust the **Screen size**. Defaults to `1024 × 768`. Allowed range is `320–8192` on each axis; values outside this range are clamped.
- Click **Connect**.

The server then:

- Preflights `guacd` with a 2-second TCP dial.

Asks the rdpproxy to allocate an ephemeral listener (one per session), which is the "port" that `guacd` will dial into.

Sends `rdp_proxy` to the host's agent, waits up to **12 seconds** for the agent to acknowledge with `rdp_proxy_connected`.

Mints an RDP ticket and returns the WebSocket tunnel URL to the browser.

The browser opens `wss://<patchmon-host>/api/v1/rdp/websocket-tunnel?ticket=…&width=…&height=…`, completes the Guacamole handshake, and starts streaming frames.

Once connected you see the Windows sign-in screen (or desktop, if NLA authenticated) in your browser.

Credentials handling

Username and password are sent once over HTTPS to `POST /auth/rdp-ticket` and stored **encrypted** in the RDP ticket record alongside the session ID, host ID, and screen dimensions.

The ticket is **single-use** and expires quickly (tens of seconds). When `guacd` consumes it to set up the tunnel, the stored credentials are forwarded to `guacd` and then to RDP. PatchMon itself does not keep them after the session starts.

For environments where Windows asks for a certificate, the default guacd config uses `ignore-cert=true` to accept the self-signed certificate Windows generates out of the box, matching `mstsc.exe` behaviour. Hardened per-host overrides are a candidate for a future release.

Security mode is negotiated as `any`, which lets FreeRDP pick the strongest common mode (NLA → TLS → legacy RDP). Hardcoding NLA would break hosts with Negotiate/TLS-only security layers and refuses blank-credential sessions.

Empty username and password are allowed (some hosts accept blank-credential sessions) but `guacd` usually fails handshake in that case; the server logs `missing_username_or_password: true` so you can spot it in the audit trail.

One-time tickets

RDP tickets work like SSH tickets:

64-character hex string, `crypto/rand` entropy.

Stored in Redis with a short TTL.

Consumed atomically on first use by `doGuacConnect`.

Bound to a user ID, a host ID, a proxy session ID, a port, the encrypted credentials, and the requested screen width/height.

Validated against the user's current active state. A deactivated user cannot re-use a still-live ticket.

You never see or handle the ticket directly; the UI requests it under the hood when you click **Connect**.

Keyboard layouts and clipboard

Keyboard: the Guacamole client maps the browser's keydown events to scancodes. For most Latin keyboards (en-GB, en-US) this "just works". For non-Latin layouts, match the Windows layout to the browser's. Guacamole has no per-session keyboard-layout selector in PatchMon 2.0.

Clipboard: bidirectional text clipboard is supported via Guacamole's native clipboard channel. Copy in Windows, paste in the browser, or vice versa. Rich clipboard (images, file lists) is not supported.

Mouse: primary, secondary, and wheel. Mouse-wheel-click middle button is supported.

Full-screen: toggle via your browser's F11/fullscreen mode. Guacamole resizes the RDP session to the browser viewport where the host allows dynamic resolution.

Printer redirection, audio, drive mapping, and USB forwarding are **not** enabled in 2.0.

Session limits

Limit	Default	Source
Concurrent RDP sessions per server	**50**	`rdpproxy.DefaultMaxSessions`
Per-session idle timeout	**30 minutes**	`rdpproxy.sessionIdleTimeout`
`guacd` preflight timeout	2 seconds	`guacdPreflightTimeout`
Agent handshake timeout	12 seconds	`agentHandshakeTimeout`

Exceeding the concurrency cap returns `503 Too many concurrent RDP sessions on this server, please try again later.`

Disconnecting

Manual disconnect: close the browser tab or click the disconnect control in the RDP panel. The server tears the session down, tells the agent to close the TCP bridge, and releases the Redis ticket record.

Windows sign-out: the RDP session closes normally; the tunnel stays open for a brief grace period before cleanup.

Idle close: after 30 minutes of no data flow the session is killed server-side.

Auditing

Every successful RDP ticket creation fires an `rdp_session_started` event:

Severity: `informational`.

Metadata: `host_id`, `host_name`, `user_id`.

Reference: the host record.

Route this event type in Notification Routes and Delivery Log if you want a live audit trail of who is signing into Windows hosts from PatchMon.

Server logs include `rdp-ticket` and `rdp session opened` lines with the session ID, user ID, host ID, negotiated security posture, and a `missing_username_or_password` field. Use these to triage incidents; the session ID ties everything together.

Troubleshooting

Symptom	Response from the server	Likely cause and fix
`guacd is not reachable on the PatchMon server.`	`503`, `code: guacd_unavailable`	The sidecar is not running. Check `docker compose ps guacd`, or install `guacd` on the host and set `GUACD_ADDRESS`.
`The PatchMon agent on this host is not connected.`	`503`, `code: agent_disconnected`	Agent is offline. Start / restart the `PatchMonAgent` service on the host.
`The PatchMon agent did not respond to the RDP proxy request in time.`	`504`, `code: agent_timeout`	The agent is connected but its handler is stuck, or blocked by firewall. Check agent logs for `rdp_proxy` entries.
rdp proxy is not enabled (via `rdp-proxy-enabled`)	`502`, `code: agent_rdp_disabled`	Set `integrations.rdp-proxy-enabled: true` in the agent `config.yml` and restart the agent.
invalid host	`502`, `code: agent_invalid_host`	Proxy host format rejected (reserved for future per-target proxies).
connection refused / *no route to host* on port 3389	`502`, `code: rdp_port_unreachable`	RDP is not running on the Windows host, or a local firewall blocks `localhost:3389`. Enable RDP on the host.
`RDP is only available for Windows hosts`	`400`	Non-Windows host. Use the Web SSH Terminal instead.
`Forbidden: origin not allowed` in WebSocket upgrade	`403`	Your browser's `Origin` header isn't in PatchMon's `CORS_ORIGIN` allow-list. Update `CORS_ORIGIN` (or the dynamic origin resolver) to include your PatchMon URL and restart.
Guacamole handshake fails repeatedly with a valid user and password	Check `rdp tunnel guacd handshake failed` in the server log. This is the 2.0.0 known-issue scenario; consult the release notes and retry.	

Related pages

- Web SSH Terminal
- AI Terminal Assistant
- Installing PatchMon Server on Docker
- Release Notes 2.0.0

Chapter 23: AI Terminal Assistant

Overview

The **AI Terminal Assistant** is an optional chat panel inside PatchMon's Web SSH Terminal. Operators open it alongside the terminal to ask questions about what they are seeing ("why did `apt` fail?", "how do I restart this service?", "explain this stack trace") and get answers from an LLM of their choice. The assistant can also turn code snippets in its replies into paste-to-terminal actions, so you stay inside a single window.

The assistant uses PatchMon as a proxy to a supported third-party AI provider (OpenRouter, Anthropic, OpenAI, or Google Gemini). The provider, model, and API key are configured once at the system level; individual operators don't have to set anything up.

Web SSH shipped in 1.4.0, and the AI assistant in the same release.

Supported providers

Four providers are supported in 2.0, each with a curated list of models:

Provider	Default model	Additional models
OpenRouter	`anthropic/claude-3.5-sonnet`	Claude 3 Haiku, GPT-4o, GPT-4o Mini, Gemini Pro 1.5, Llama 3.1 70B
Anthropic	`claude-sonnet-4-20250514`	Claude 3.5 Sonnet, Claude 3.5 Haiku
OpenAI	`gpt-4o-mini`	GPT-4o, GPT-4 Turbo
Google Gemini	`gemini-1.5-flash`	Gemini 1.5 Pro, Gemini 2.0 Flash (experimental)

Pick one provider per PatchMon deployment. To change providers, edit the AI settings. The API key is cleared automatically when you switch, and you'll be asked to enter a new one for the new provider.

Module gate

The AI assistant is part of the **ai** capability module (also referred to as **ai_assist** in some settings). If your subscription does not include the AI module, the settings page is visible but cannot be enabled. Ask your account administrator if the AI features are missing entirely from your instance.

Permissions

Area	Permission
Configure AI settings (provider, model, API key)	`admin` or `superadmin` only
Use the AI assistant in a terminal	Any user who can open the SSH terminal (admin/superadmin, or `can_use_remote_access`)

There is no separate per-user toggle. If AI is enabled at the system level and you have terminal access, the assistant is available to you.

Configuring a provider

Go to **Settings → AI Terminal Assistant**.

1. Pick your provider

Use the **Provider** dropdown. The **Model** dropdown below it repopulates with that provider's models and auto-selects the provider's default. Changing the provider immediately clears the stored API key (because keys belong to one provider each).

2. Enter your API key

Each provider issues its own key:

Provider	Get your key from
OpenRouter	openrouter.ai/keys (https://openrouter.ai/keys)
Anthropic	console.anthropic.com/settings/keys (https://console.anthropic.com/settings/keys)
OpenAI	platform.openai.com/api-keys (https://platform.openai.com/api-keys)
Gemini	aistudio.google.com/apikey (https://aistudio.google.com/apikey)

Paste the key into the **API Key** field and click **Save**. PatchMon encrypts the key with your instance's `SESSION_SECRET` before writing it to the database. The key is never returned to the browser after saving; only a boolean "is set" flag is exposed via the API.

> ***API Key Needs to be Re-entered.*** *If PatchMon later fails to decrypt the stored key (for example, because `SESSION_SECRET` was rotated or was inconsistent across restarts), the settings page shows a yellow banner. Re-enter the key to clear it.*

3. Test the connection

Click **Test Connection**. The server sends a one-sentence round-trip to the configured provider and checks the response. A green check plus *"AI connection test successful"* confirms everything works; a red error means the key is wrong, the model is unavailable, or the provider is unreachable from the PatchMon server.

4. Enable the assistant

Flip the **Enable AI Assistant** toggle at the top of the page. Until this is on, the chat panel inside the SSH terminal is hidden for everyone.

Using the assistant in a terminal

1. Open a web SSH terminal to any host (see Web SSH Terminal).
2. Click the robot icon in the terminal toolbar to open the assistant panel on the right.
3. Type a question and press **Enter**. Example questions:
   - *"The `systemctl status nginx` output says `(code=exited, status=1/FAILURE)`. What's wrong?"*
   - *"How do I check disk usage on this Ubuntu host?"*
   - *"Explain this error message."*

   The assistant replies inline. Code snippets (fenced with triple backticks or tagged as commands) get a **Play** and **Copy** button so you can paste the command into the terminal without typing.

The panel is a normal chat. You can keep asking follow-ups and the assistant keeps context.

What data is sent to the provider

Each request to `/api/v1/ai/assist` includes:

- The **system prompt**: hardcoded in PatchMon, positions the model as a terminal helper for Linux/Unix administration.
- The **terminal context**: the last ~3 000 characters of terminal output captured from the browser buffer, wrapped in Markdown fences, so the model can read what you're seeing. The server caps the uploaded context at 10 000 characters as a safety net.
- The **conversation history**: up to the last 10 messages (user + assistant) from the current chat session, each trimmed to 2 000 characters.
- The **question**: your current message, 1–2 000 characters.

The question is then proxied to the provider you configured (OpenRouter, Anthropic, OpenAI, or Gemini). PatchMon does not retain the request beyond the normal server access log.

Command-completion requests (when you pause while typing into the terminal, if completion is enabled) send:

- Up to 5 000 characters of context.
- The partial command you're typing (2–500 characters).
- A low-temperature prompt instructing the model to output only the completion.

Privacy considerations

- Terminal output you capture in the buffer **is** sent to the third-party provider as context. If you've just run a command that shows sensitive data (API keys, secrets, customer data), clear the terminal or don't ask the assistant about it.
- Your provider's terms of service govern what they may do with the request. Review the provider's data-processing policy before enabling on production hosts. OpenRouter, Anthropic, OpenAI, and Gemini all publish policies.
- API key secrecy: keys are encrypted at rest and never echoed back over the API. Admins with database access could still read the encrypted value; rotate `SESSION_SECRET` carefully.
- All provider traffic leaves the PatchMon server over HTTPS directly to the provider's endpoint. PatchMon does not route it through any intermediate service.

If these trade-offs are not acceptable for a particular environment, leave the assistant disabled. The normal SSH terminal works fine without it.

Rate limiting

Each user is limited to **30 AI requests per minute** across `assist` and `complete` combined. The limit is enforced in Redis with a 60-second window. Exceeding the limit returns `429 Rate limit exceeded. Please wait a moment.` The panel shows the error inline and you can retry after the window resets.

Rate limiting is per PatchMon user, not per IP. It exists to protect your provider spend, not to throttle normal interactive use. 30/min is plenty of headroom for a single operator, while catching runaway scripts.

Input and response limits

- Questions: 1–2 000 characters. Longer input is rejected with `400`.
- Context: 10 000 characters max (server trims longer input).
- Conversation history: last 10 messages sent to the provider.
- Per-message trim: 2 000 characters.
- Completion input: 2–500 characters.
- Completion context: 5 000 characters.

`max_tokens` per assistant reply: **1024**.

Assistant `temperature`: **0.7** (creative but focused).

Completion `temperature`: **0.3** (conservative).

These values match the product defaults in `internal/ai/service.go` and are not currently configurable via the UI.

Enabling and disabling

Per-deployment: the admin toggle in **Settings → AI Terminal Assistant**. Off = panel is hidden for everyone.

Per-user (soft): any user can simply keep the panel closed. There is no per-user opt-out flag.

Emergency off: clear the API key in **Settings → AI Terminal Assistant**. The server-side AI endpoints return `400 AI API key not configured` and the panel surfaces that error.

Troubleshooting

Symptom	Likely cause
AI panel does not appear in the terminal	AI module not included in your subscription, or AI not enabled in settings, or no API key set.
"AI assistant is not enabled" in the panel	Toggle is off in settings.
"AI API key not configured"	Key field is empty or decryption failed. Re-enter the key.
"Rate limit exceeded. Please wait a moment."	30 requests/minute cap hit for your user. Back off and retry.
Test connection fails with `401` from provider	API key is wrong or revoked. Re-issue and re-enter.
Test connection fails with `404 model not found`	The model listed in PatchMon is not available on your account. Switch to a different model in the dropdown.
Replies are truncated mid-sentence	Response hit the 1024-token cap. Ask a narrower follow-up or paste a smaller context.

Related pages

Web SSH Terminal

RDP via Guacamole

Chapter 24: Users, Roles, and RBAC

PatchMon uses role-based access control (RBAC) to decide who can see and do what inside the application. Every user has exactly one role, and every role is a collection of permissions. This page covers the built-in roles, the full permission list, and how to manage users and roles from the Settings UI.

> ***Related pages:***
>
> *Setting Up OIDC / Single Sign-On: authenticate users against an external IdP*
>
> *Setting Up Microsoft Azure Entra ID (SSO) with PatchMon: Entra-specific walkthrough*
>
> *Two-Factor Authentication: per-user TOTP and trusted devices*

The Built-In Roles

PatchMon ships with five roles. You see these in **Settings → Users** (in the **Role** dropdown) and in **Settings → Roles** (as the matrix columns).

Role	Default Permissions	Typical Use
Super Admin (`superadmin`)	Everything, including managing other superadmins	The very first user, or dedicated platform owners
Admin (`admin`)	Everything except managing other superadmins	Day-to-day platform administrators
Host Manager (`host_manager`)	Monitoring + host/infrastructure management + operations (patching, compliance, alerts, automation, remote access)	NOC / Ops engineers
User (`user`)	Monitoring + data export	Engineers who need to look but not break
Readonly (`readonly`)	Monitoring only	Auditors, read-only dashboards, management

Two important rules about built-ins:

- **Cannot be deleted.** `superadmin` , `admin` , `host_manager` , `user` and `readonly` are always present. The **Delete** button does not appear for them.
- **The core three cannot have their permissions edited.** `superadmin` , `admin` and `user` are *locked*: their permission matrix is hardcoded and the **Edit** button is disabled. `host_manager` and `readonly` can still be edited if you want to tune them.

> ***First user is always Super Admin.*** *When PatchMon is first installed and has no users, the setup wizard creates the initial account as* `superadmin`*, regardless of what role you type. If OIDC is configured for auto-create before first boot, the very first OIDC login is also promoted to* `superadmin` *automatically so you cannot lock yourself out.*

The Full Permission List

Permissions are grouped into four risk tiers. The colour you see in the **Roles** matrix corresponds to this risk level.

Monitoring & Visibility (Low risk)

Read-only access to dashboards, hosts, packages, reports, and logs.

Permission key	Label	What it lets the user do
`can_view_dashboard`	View Dashboard	View the main dashboard and its stat panels
`can_view_hosts`	View Hosts	See the host list, host detail pages, and connection status
`can_view_packages`	View Packages	See the package inventory across all hosts
`can_view_reports`	View Reports	See compliance scan results and alert reports
`can_view_notification_logs`	View Notification Logs	See notification delivery history and status

Host & Infrastructure (Medium risk)

Create, modify and delete hosts, packages, and containers.

Permission key	Label	What it lets the user do
`can_manage_hosts`	Manage Hosts	Create / edit / delete hosts, host groups, repositories and integrations
`can_manage_packages`	Manage Packages	Edit package inventory and metadata
`can_manage_docker`	Manage Docker	Delete Docker containers, images, volumes and networks

Operations (Medium-High risk)

Day-to-day NOC tasks.

Permission key	Label	What it lets the user do
`can_manage_patching`	Manage Patching	Trigger patches, approve patch runs, manage policies
`can_manage_compliance`	Manage Compliance	Trigger compliance scans, remediate findings, install scanners
`can_manage_alerts`	Manage Alerts	Assign, delete and bulk-action alerts
`can_manage_automation`	Manage Automation	Trigger and manage automation jobs
`can_use_remote_access`	Remote Access	Open SSH and RDP terminals against managed hosts

Administration (High risk)

Organisation-wide control.

Permission key	Label	What it lets the user do
`can_view_users`	View Users	See the user list and account details
`can_manage_users`	Manage Users	Create, edit and delete user accounts
`can_manage_superusers`	Manage Superusers	Manage `superadmin` accounts and elevated privileges
`can_manage_settings`	Manage Settings	System configuration, OIDC / SSO, AI, alert config, enrollment tokens
`can_manage_notifications`	Manage Notifications	Configure notification destinations and routing rules
`can_export_data`	Export Data	Download and export data and reports

> ***Billing:*** *On PatchMon Cloud there is also a `can_manage_billing` permission that governs access to the Billing page. On self-hosted instances this permission exists in the schema but the Billing page is not enabled by default.*

Viewing the Role Matrix

1. Sign in as a user with `can_manage_settings`.
2. Go to **Settings → Roles**.

You'll see a matrix: rows are permissions (grouped by tier), columns are roles. A green tick means the role has that permission.

Each column header also shows an `n/N` counter showing the number of permissions that role currently holds out of the total 20.

Creating a Custom Role

Custom roles let you tailor the permission set beyond the built-in five.

> ***Availability:*** *The* ***Add Role*** *button is only shown when the* `rbac_custom` *module is enabled on your PatchMon deployment. On self-hosted installs this module is typically enabled by default; on PatchMon Cloud it depends on your plan. If you don't see* ***Add Role*** *and the URL* `https://patchmon.example.com/settings/roles` *shows a "Not Available" screen, the module isn't enabled on your plan.*

To create one:

1. Go to **Settings → Roles**.
2. Click **Add Role** in the top-right.
3. Fill in the modal:
   - **Role Name:** lowercase, underscores instead of spaces. Examples: `host_manager`, `compliance_auditor`, `noc_operator`. This is the internal key; it cannot be renamed later.
   - **Preset** (optional): four quick-start presets are available:
     - **Read Only:** just the Monitoring & Visibility group
     - **Operator:** everything except the Administration group
     - **Admin:** every permission
     - **Clear All:** start from zero
   - **Permissions:** tick / untick individual permissions, or use the **Select all / Deselect all** shortcut on each group header.
4. Watch the counter at the bottom (`n/20 permissions selected`) as a sanity check.
5. Click **Create Role**.

The new role appears as a new column in the matrix and is selectable when creating or editing users.

Editing a Custom Role

1. In the matrix, click the pencil icon in the column header of the role you want to edit.
2. An editor panel opens below the matrix with all permissions listed.

Tick / untick as needed, then click **Save**.

Changes take effect immediately. Any session held by a user with that role has its in-memory permissions refreshed on their next request.

Deleting a Custom Role

You can only delete a role that is **not assigned to any user**. If any user holds that role, the delete endpoint rejects the request with "Cannot delete role: users are assigned to it". Reassign those users to a different role first (see Editing a Role for an Existing User).

To delete:

Click the pencil in the role's column header to open the editor panel.

Click **Delete** (appears only for non-built-in roles).

Confirm.

Creating Users

Go to **Settings → Users** and click **Add User** in the top-right.

Field	Notes
Username	Minimum 3 characters. Lowercase recommended
Email	Must be a valid email. Used for OIDC account linking and email alerts
First Name / Last Name	Optional
Password	Must satisfy the active password policy (configured under **Settings → Server Config → Security**)
Role	Choose from built-in or custom roles

Click **Add User**. The account is created immediately and can sign in straight away.

> ***Role escalation protection:*** *You cannot create a user with a role that's more privileged than your own. Only* `superadmin` *users can create new* `admin` *or* `superadmin` *accounts. Non-superadmin accounts that hold the* `can_manage_superusers` *permission can also create and manage* `superadmin` *accounts.*

Self-Service Sign-Up

PatchMon can also let users register themselves rather than having an admin invite them.

Go to **Settings → Users**.

Scroll to **User Registration Settings**.

Tick **Enable User Self-Registration**.

Pick a **Default Role for New Users**: the role that self-registered accounts are assigned.

Click **Save Settings**.

A sign-up link now appears on the login page. Anyone who can reach the login page can create an account.

> ***Security warning:*** *Only enable self-registration on internal or private-network deployments. If your PatchMon is internet-facing, leave it off and invite users manually, or front it with OIDC SSO (which lets your IdP decide who can log in).*

Editing a Role for an Existing User

Go to **Settings → Users**.

Find the user in the table and click the **Edit** (pencil) icon.

Change **Role** in the dropdown and click **Save**.

Important side effects:

Sessions are revoked. When a user's role changes, all of their existing JWT sessions are invalidated on the server. They must sign in again. This ensures the old role's privileges cannot be replayed from an existing browser tab.

You cannot change your own role. The API rejects a self-role change with "Cannot change your own role". This is a deliberate safety net: two admins must cooperate to demote each other.

You cannot promote a user above yourself. An `admin` cannot promote a user to `superadmin`. Only a `superadmin` can create or promote to `superadmin`, and likewise only `superadmin` can assign the `admin` role.

Resetting a User's Password

In the users table, click the **Reset** (key) icon on that user's row.

Enter a new password.

Click **Reset Password**.

After a reset, all of that user's sessions and trusted-device records are revoked. This is the standard post-compromise response. The user must sign in with the new password on every device.

> *You cannot reset the password of an inactive user. Reactivate them first.*

Disabling (Deactivating) a User

Disabling is the safer alternative to deletion. The user record, their history, and their audit trail are preserved, but they cannot log in.

1. Go to **Settings → Users**.
2. Click the **Edit** icon on the user you want to disable.
3. Untick the **Active** checkbox.
4. Click **Save**.

Effects:

- All their sessions are revoked immediately.
- All their trusted devices are revoked (so re-activating them later cannot reuse a "remember this device" cookie that predates the deactivation window).
- The user's row is shown with a red **Inactive** badge in the users table.

To re-enable: edit and tick **Active** again.

Deleting a User

Deletion is permanent and removes the user record and their associated dashboard preferences, sessions, trusted devices and notification preferences.

1. Click the **Delete** (trash) icon on the user's row.
2. Confirm.

Restrictions:

- You cannot delete your own account.
- You cannot delete the last `superadmin` (the API refuses).
- You cannot delete the last `admin` if there are no `superadmin` users (ensures at least one admin always exists).
- You cannot delete a user who holds a role that's more privileged than yours.

How Permissions Are Evaluated

Admin and Super Admin always have every permission, even if the `role_permissions` table says otherwise. The middleware short-circuits their permission checks. This is a safety net: if someone mis-edits the `admin` row (which shouldn't be possible via the UI, but could happen via direct database access), admins don't get locked out.

Every other role (built-in or custom) has its permissions read from the database at each request. Changes made in **Settings → Roles** take effect on the user's next API call; no restart required.

Role hierarchy for user management is enforced separately from the permissions above:

`superadmin` → rank 100

`admin` → rank 90

`host_manager` → rank 50

custom roles → rank 30 (mid-tier)

`user` → rank 20

`readonly` → rank 10

You can only modify, delete, or reset the password of users whose role rank is less than or equal to your own. This is distinct from the permission checks. Even if a custom role were granted `can_manage_users`, its holder still could not touch `admin` or `superadmin` accounts unless they additionally had `can_manage_superusers`.

When OIDC Role Sync Is Enabled

If **Settings → OIDC / SSO → Sync roles from IdP** is on, PatchMon stops letting admins manage users and roles from the UI. Instead:

- The **Add User** and **Add Role** buttons disappear.
- The Users tab shows a read-only list.
- The Roles tab shows a banner reminding you that group membership in your IdP drives role assignment via environment variables: `OIDC_SUPERADMIN_GROUP`, `OIDC_ADMIN_GROUP`, `OIDC_HOST_MANAGER_GROUP`, `OIDC_USER_GROUP`, `OIDC_READONLY_GROUP`.
- Users' roles are re-evaluated on every login based on their current IdP group membership.

If you want to use OIDC for authentication but still manage roles locally in PatchMon, leave **Sync roles from IdP** off. See Setting Up OIDC / Single Sign-On for the full toggle reference.

Troubleshooting

"You do not have permission to assign the role: admin"

Only a `superadmin` can create or promote users to `admin` or `superadmin`. If you're an `admin` and try to promote someone to `admin`, the API refuses. Ask a superadmin to do it.

"Cannot modify built-in role permissions"

The `superadmin`, `admin` and `user` rows are locked against permission edits. If you need a role with tweaked permissions, create a custom role based on a preset and assign users to that instead.

"Cannot delete role: users are assigned to it"

Before a role can be deleted, reassign every user who holds it. Use **Settings → Users → Edit** to change each user's role, then try the delete again.

"Cannot delete the last superadmin user" / "Cannot delete the last admin user"

At least one `superadmin` must always exist. If there are no superadmins at all, at least one `admin` must exist. Create a replacement first (and sign in as them to confirm the login works) before deleting the final one.

User's old role is still in effect after I changed it

Changing a role revokes all existing sessions, but the user's browser may still hold an old JWT cookie that hasn't been rejected yet. Ask them to refresh the page or sign out and back in; the server will reject the stale token and redirect them to login.

"Add User" / "Add Role" button is missing

Three possible causes:

Your role doesn't have `can_manage_settings` or `can_view_users`. Check `/settings/users`: if the page is empty or you get a Forbidden, your role lacks the view permission.

OIDC role sync is on. See When OIDC Role Sync Is Enabled.

The `rbac_custom` module is not enabled. This only affects the **Add Role** button on the Roles tab. Custom role creation is a gated feature. The **Add User** button on the Users tab is always available when the other two conditions are met.

Chapter 25: Two-Factor Authentication

PatchMon supports time-based one-time password (TOTP) two-factor authentication (2FA, sometimes called MFA) on top of the normal username / password login. Once enabled on a user's account, every sign-in asks for a 6-digit code from an authenticator app, or a one-time backup code.

This page covers enabling 2FA per user, using backup codes, the "Remember Me" trusted-device feature, and how admins recover an account if the user loses their authenticator.

> ***Related pages:***
>
> *Users, Roles and RBAC: manage user accounts*
>
> *Setting Up OIDC / Single Sign-On: delegate authentication to an external IdP*
>
> *PatchMon Environment Variables Reference: the full env-var list*

Scope and limitations

2FA is **opt-in per user**. Each user decides whether to turn it on from their own profile.

2FA is **not available for OIDC accounts.** If a user signs in via OIDC / SSO, their IdP is responsible for MFA. The PatchMon TFA tab is hidden on the profile page for OIDC-only accounts, and the setup endpoint refuses with *"MFA is managed by your OIDC provider"*.

There is **no global "require 2FA for all users"** flag in the current release. Administrators cannot enforce 2FA for every account from the UI or an environment variable. If you need enforced 2FA, drive authentication through an OIDC provider that enforces MFA (e.g. Authentik, Entra ID) and set `OIDC_DISABLE_LOCAL_AUTH=true`.

The first-time setup wizard offers new admins the option to set up 2FA during initial account creation (Step 2 of the wizard). This is voluntary and can be skipped.

Enabling 2FA on Your Account

Each user enables 2FA themselves from their profile. Admins cannot enable it on behalf of another user.

Sign in to PatchMon with your username and password.

Click your avatar (top-right) → **Profile**.

Open the **Multi-Factor Authentication** tab.

Click **Enable TFA**.

A QR code appears. Scan it with your authenticator app of choice. Known-good options:

Authy

Google Authenticator

1Password

Bitwarden

Microsoft Authenticator

Duo Mobile

If you can't scan the QR code (shared device, desktop-only app), copy the **Manual Entry Key** instead and paste it into your authenticator.

Click **Continue to Verification**.

Enter the current 6-digit code from your authenticator app.

Click **Verify & Enable**.

You are now shown a one-time list of **backup codes** (see next section). Save them before clicking **Done**.

From now on, every password-based login will prompt for a 6-digit verification code after the password step.

Backup codes: save these

After enabling 2FA, PatchMon generates a batch of single-use backup codes. These let you sign in if you lose access to your authenticator app (lost phone, wiped device, etc.).

Each code can be used exactly once. Once used, it is consumed and cannot be reused.

They are shown only once, in plaintext, immediately after setup or regeneration. PatchMon stores them as bcrypt hashes in the database; neither you nor an admin can recover the plaintext later.

Treat them like a second password. Store them in a password manager, or print them and lock them away.

Click **Download Codes** to save a plain-text file for offline storage.

Regenerating backup codes

If you think your backup codes have leaked, or you've used most of them:

Go to **Profile → Multi-Factor Authentication**.

Scroll to the **Backup Codes** panel.

Click **Regenerate Codes**.

A new set of codes is generated and shown. The old set is immediately invalidated.

Using a backup code

On the 2FA prompt at login, you enter backup codes in the **same field** as TOTP codes. There is no separate "use a backup code" button. PatchMon tries the code as a TOTP first; if that fails, it checks whether it matches one of the stored backup-code hashes. If it matches, that backup code is consumed (removed from the stored list) and you are logged in.

Typical workflow if you've lost your phone:

At the login page, enter your username and password as usual.

On the "Two-Factor Authentication" screen, type one of your backup codes in the **Verification Code** field.

Click **Verify**.

The code is spent. Your next login cannot use the same backup code again.

"Remember Me": Trusted Devices

When you enter your 2FA code, there's a **Remember me on this computer (skip TFA for 30 days)** checkbox. If ticked, PatchMon plants a long-lived, HttpOnly `patchmon_device_trust` cookie on that browser and records a hashed trust token in the database.

On subsequent logins from the same browser:

You still enter your password.

PatchMon sees the trust cookie, matches it to the database record, confirms the record belongs to you and hasn't expired, and skips the 2FA prompt.

A `last_used_at` timestamp on the trust record is bumped each time it's used, so you can see when each remembered device last signed in.

How the trust is keyed

The trust cookie is keyed only on **(user ID, cookie hash)**. It is deliberately **not** bound to IP address or user agent, so:

Roaming between Wi-Fi, mobile hotspot, and office network does not invalidate the trust.

Updating your browser does not invalidate the trust.

Copying the cookie to a different browser on a different machine **would** bypass 2FA for that user (standard web cookie security model). Protect your browser profile accordingly.

Trust lifetime

The default lifetime is **30 days**, configurable server-wide via the `TFA_REMEMBER_ME_EXPIRES_IN` environment variable. Accepts duration strings such as `7d`, `30d`, `90d`. See PatchMon Environment Variables Reference for the full list.

There is a hard cap on how many trusted devices a single user can accumulate, controlled by `TFA_MAX_REMEMBER_SESSIONS` (default `5`). When a sixth device is trusted, the oldest existing trust is removed automatically.

Reviewing your trusted devices

Go to **Profile → Trusted Devices**.

You'll see a list with, for each device:

Label (best-effort device name derived from the user agent)

User agent

IP address at the time it was last used

Created / **Last used** / **Expires** timestamps

A **This device** badge next to the one you're currently logged in from

Revoking a trusted device

To stop a specific device skipping 2FA (for example, an old laptop you're decommissioning):

Profile → Trusted Devices.

Find the device in the list and click **Revoke**.

Confirm.

If the device you're revoking is the **current** browser, its trust cookie is also cleared, so your next login from this browser will require 2FA again.

Revoking every trusted device

Click **Forget all trusted devices** at the top of the panel. This:

Removes every trust record for your account.

Clears the trust cookie on the current browser.

Forces a full 2FA prompt on every device next time you sign in.

Use this after a suspected account compromise or after losing a device.

Disabling 2FA

To turn 2FA back off on your own account:

Profile → Multi-Factor Authentication.

Click **Disable TFA**.

Enter your password to confirm.

Click **Disable TFA**.

Side effects:

The TOTP secret is wiped from the database.

All existing backup codes are invalidated.

All of your trusted devices are revoked. This is intentional. The only purpose of a trust record is to skip 2FA, so with 2FA off they serve no purpose. If you later re-enable 2FA, old trust cookies will not be resurrected and every device must confirm 2FA again.

> *You cannot disable 2FA on an OIDC-only account. The API rejects the request with "Cannot disable TFA for accounts without a password". This is because disabling 2FA requires password confirmation, and OIDC-only accounts have no password set.*

Failed Attempts and Lockout

To prevent brute-forcing the 6-digit code space, the verify-2FA endpoint is rate-limited per user.

Env var	Default	What it does
`MAX_TFA_ATTEMPTS`	`5`	Consecutive wrong codes allowed before a lockout
`TFA_LOCKOUT_DURATION_MINUTES`	`30`	How long the lockout lasts

After the cap is hit, the endpoint returns HTTP `429 Too Many Requests` with the message *"Too many failed TFA attempts. Please try again later."* Wait out the lockout, or ask an admin (see below).

Each failure also returns a `remainingAttempts` counter in the response, so the login UI can tell the user how many tries are left.

First-Time Wizard: Optional 2FA Setup

When you bring up a brand-new PatchMon instance and complete the setup wizard, **Step 2 (Multi-Factor Authentication)** offers two choices:

> **Setup MFA now:** scan a QR code and register an authenticator for the brand-new admin account before finishing the wizard. You'll also capture your backup codes.
>
> **Skip for now:** the admin account is created without 2FA. You can turn it on later from **Profile → Multi-Factor Authentication**.

There is no "enforce for everyone" option in the wizard. This decision is always per-user.

Admin Recovery: User Has Lost Their Authenticator

PatchMon does not have a dedicated "admin reset MFA" button. Recovery is handled through the standard account-recovery flow, which implicitly disables 2FA in a safe way:

Option A: User has a backup code

Ask them to sign in with a backup code (see Using a backup code). Once they're in, they can:

> **Profile → Multi-Factor Authentication → Disable TFA** to remove the old authenticator secret entirely, and then re-enable with the new phone.
>
> Or **Regenerate Codes** to get a fresh set of backup codes without touching the authenticator.

Option B: User has no backup codes and no authenticator

An administrator must reset the account:

> Sign in as a user with `can_manage_users` (admin, superadmin, or any custom role with that permission).
>
> Go to **Settings → Users**.
>
> Find the affected user and click **Reset Password**.
>
> Set a new password and communicate it over an out-of-band secure channel.

> ***Password reset alone does not disable 2FA.*** *The user will still be prompted for a TOTP or backup code after their first login with the new password.*

If the user still cannot produce a code, you have two further options:

Deactivate, then reactivate the account. Edit the user, untick **Active**, save (this also wipes their trusted devices), then tick **Active** again. 2FA is still enabled on the account, so this alone does not solve the missing-authenticator problem.

Delete and re-create the user as a last resort. You lose the user's ID, notification preferences, and any artefacts keyed to their account, so prefer the backup-code route wherever possible.

> ***Feature gap:*** *A "wipe 2FA on another user" admin action is on the roadmap. If you hit this frequently, consider moving your deployment to OIDC / SSO so that MFA is managed by the IdP (see Setting Up OIDC / Single Sign-On).*

Direct database workaround (self-hosted only)

If you are self-hosting and absolutely need to clear 2FA on a user without backup codes, a DBA can clear the user's `tfa_enabled`, `tfa_secret` and `tfa_backup_codes` columns directly in the `users` table, then force a password reset from the UI. This is a last resort. Make a backup first, and never do this on PatchMon Cloud (where direct database access is not available).

```
-- Replace 'alice' with the affected username. Make a backup first.
UPDATE users
SET tfa_enabled = false,
    tfa_secret = NULL,
    tfa_backup_codes = NULL
WHERE username = 'alice';
```

After running this the user can sign in with just a password; they should immediately re-enrol in 2FA from their profile.

Environment Variables Reference

All of these are read once at server start. Changes require a restart to take effect. The full table lives in PatchMon Environment Variables Reference; reproduced here for convenience:

Variable	Default	Description
`MAX_TFA_ATTEMPTS`	`5`	Consecutive wrong 2FA codes before the account is temporarily locked
`TFA_LOCKOUT_DURATION_MINUTES`	`30`	How long a 2FA lockout lasts
`TFA_REMEMBER_ME_EXPIRES_IN`	`30d`	How long a "Remember me" trusted-device record is valid. Accepts `7d`, `30d`, `90d`, etc.
`TFA_MAX_REMEMBER_SESSIONS`	`5`	Maximum number of trusted devices per user; the oldest is evicted when the limit is reached

Troubleshooting

"Invalid verification code" when I know the code is correct

Clock skew. TOTP codes are time-based. If your phone's clock is more than ~30 seconds out of sync with the server, codes will be rejected. Enable automatic date/time on your phone. PatchMon already tolerates a small drift window server-side, but not more than that.

Using a used code. TOTP codes roll every 30 seconds. If you paste a stale code from 60+ seconds ago it will fail. Wait for a fresh code.

Used backup code. Backup codes are single-use. If you've already used one, try a different one.

"Too many failed TFA attempts"

You've hit `MAX_TFA_ATTEMPTS`. Wait `TFA_LOCKOUT_DURATION_MINUTES` (default 30) and try again. There is no admin "unlock" button; the lockout key in Redis expires automatically. Self-hosters can flush the key by restarting Redis.

I ticked "Remember me" but I'm still being asked for 2FA

Three likely causes:

The trust record has expired. The default lifetime is 30 days; check `TFA_REMEMBER_ME_EXPIRES_IN` on your server.

You're signing in on a different browser, or in a private / incognito window, which doesn't have the cookie.

A password reset was performed on your account. Password resets (whether self-service or admin-initiated) revoke every trusted device as part of the security response. You'll need to tick **Remember me** again on the next 2FA prompt.

My MFA tab is missing on the profile page

You signed in via OIDC. PatchMon defers MFA to your IdP in that case. Enable MFA in your IdP (Entra ID, Authentik, Keycloak, etc.) if you want it.

I regenerated backup codes but the old ones still work

The old codes are invalidated at the same moment the new batch is displayed. If a stale code still seems to work, make sure you're looking at the right account. Backup codes are not user-transferable.

Chapter 26: Metrics and Telemetry

What we collect and why

We collect three pieces of information about PatchMon instances in the field:

- Quantity of installations / live setups
- Quantity of hosts being monitored
- Version number of the instance

This lets us produce a live statistic on patchmon.net (https://patchmon.net) showing adoption across the community, and (more importantly) lets us know how many instances are running an older version if a security issue is found.

This was discussed with the community on Discord; the original conversation is pinned in the **Security** channel.

What we do **not** collect

- **IP addresses.** IPs are not written to any log or stored when your instance reaches out to us.
- **Host, user, or package data.** Only the three fields above, plus a random instance UUID that identifies your install across reports.

How to opt out

Go to **Settings → Metrics** in the web UI and toggle the schedule off. From that moment, your instance stops sending telemetry.

FAQ

How do I delete the information you have about my instance?

Email support@patchmon.net with your UUID and we will remove your entry from the database. This is the only time we can associate your UUID with your instance, so once it is deleted we have no further link back to you.

What happens if I regenerate my instance ID?

A new instance ID appears in our reports and is counted as a new instance. We have no way to know which instance it replaced. Our website metric counts only instances active in the last 7 days, so old UUIDs drop out after a week.

Can I see the code for this?

Yes, PatchMon is open source. You can inspect the metrics collector in the PatchMon repository (https://github.com/PatchMon/PatchMon).

PatchMon

ABOUT PATCHMON

The open source Linux patch management platform

PatchMon gives sysadmins one dashboard for patching across Linux, FreeBSD, and Windows fleets. Run it as a managed SaaS on PatchMon Cloud (per-host billing, 14-day trial, no fleet minimum) or self-host the AGPLv3 Community Edition on your own infrastructure.

Start a trial: patchmon.net/pricing