PatchMon PATCHMON DOCS

PatchMon Operator Guide

Install, configure, and maintain PatchMon on Docker, Kubernetes, or behind a reverse proxy. Includes OIDC SSO and agent lifecycle.

LAST UPDATED
25 April 2026

CHAPTERS
15

READ ONLINE
https://patchmon.net/docs/patchmon-operator-guide

SOURCE
github.com/PatchMon/PatchMon

Contents

This is the deployment, configuration, and maintenance guide for PatchMon operators running the platform. For day-to-day app usage in the web UI, see the **Admin Guide**. For integrations and the REST API, see the **API & Integrations Guide**.

Table of Contents

Chapter 1: Installing PatchMon Server on Docker

Overview

PatchMon runs as a single container backed by three supporting services. The PatchMon server binary serves both the API and the embedded React frontend. There is no separate frontend container.

Service	Image	Purpose
`server`	`ghcr.io/patchmon/patchmon-server`	PatchMon application (API + frontend + migrations)
`database`	`postgres:17-alpine`	Primary data store
`redis`	`redis:7-alpine`	Background job queues (asynq)
`guacd`	`guacamole/guacd:1.5.5`	RDP gateway (required for in-browser RDP)

All four services communicate over an isolated internal Docker network (`patchmon-internal`). Only the `server` port is exposed to the host.

Prerequisites

- Docker Engine 24+ and Docker Compose v2
- A reverse proxy with a valid TLS certificate (Nginx Proxy Manager, Traefik, Caddy, or similar), strongly recommended for any non-localhost deployment
- Minimum 1 GB RAM, 2 GB recommended

Quick Start

1. Run the setup script

The fastest way to get started is the official setup script. It downloads the compose file, generates secrets, and creates your `.env` in one step:

```
mkdir patchmon && cd patchmon
bash -c "$(curl -fsSL
https://raw.githubusercontent.com/PatchMon/PatchMon/refs/heads/main/docker/setup-env.sh)"
```

Once it completes, skip to step 3.

2. Manual setup (alternative)

If you prefer to set things up yourself:

```
mkdir patchmon && cd patchmon

# Download the compose file and example env
curl -fsSL -o docker-compose.yml
https://raw.githubusercontent.com/PatchMon/PatchMon/refs/heads/main/docker/docker-compose.yml
curl -fsSL -o env.example
https://raw.githubusercontent.com/PatchMon/PatchMon/refs/heads/main/docker/env.example

# Create your .env and generate the three required secrets
cp env.example .env
sed -i "s/^POSTGRES_PASSWORD=$/POSTGRES_PASSWORD=$(openssl rand -hex 32)/" .env
sed -i "s/^REDIS_PASSWORD=$/REDIS_PASSWORD=$(openssl rand -hex 32)/" .env
sed -i "s/^JWT_SECRET=$/JWT_SECRET=$(openssl rand -hex 64)/" .env
```

3. Configure your access URL

Open `.env` and set `CORS_ORIGIN` to the full URL that PatchMon will be reachable at in a browser. This is the only URL-related env var the server reads from `.env`:

```
CORS_ORIGIN=https://patchmon.example.com
```

For a local test without a reverse proxy:

```
CORS_ORIGIN=http://localhost:3000
```

If users reach PatchMon on more than one URL (for example an external domain and an internal LAN address), comma-separate the values with no spaces:

```
CORS_ORIGIN=https://patchmon.example.com,https://patchmon.internal.lan
```

After first login, go to **Settings** in the PatchMon UI to configure the server URL that agents use to connect back (protocol, host, port). These are stored in the database, not in `.env`. The Settings page is also where you configure update intervals, auto-update behaviour, and other server-level options.

> ***Tip:*** *Do not edit the `docker-compose.yml` file to add env vars. The compose file uses `env_file: .env` to pass your entire `.env` into each container. All configuration lives in `.env`.*

For the full list of available environment variables (rate limiting, logging, OIDC SSO, TOTP, database pool tuning, session timeouts, and more), see the PatchMon Environment Variables Reference.

4. Start PatchMon

```
docker compose up -d
```

Docker will pull the images, wait for each service to pass its health check, and start the server. The server automatically runs database migrations on startup.

Once all containers are healthy, open the URL you configured in your browser and complete the first-time setup to create your admin account.

You can check the startup logs at any time with:

```
docker compose logs -f server
```

Container Image

The PatchMon server image is published at:

```
ghcr.io/patchmon/patchmon-server
```

Available Tags

Tag	Description
`latest`	Latest stable release
`x.y.z`	Exact version pin (e.g. `1.5.0`)
`x.y`	Latest patch in a minor series (e.g. `1.5`)
`x`	Latest minor and patch in a major series (e.g. `1`)
`edge`	Latest development build from the main branch. Unstable, for testing only.

Compose File Reference

This is the production `docker-compose.yml` used by PatchMon. You do not need to edit it; all configuration is controlled via your `.env` file.

```
name: patchmon

services:

  server:
    image: ghcr.io/patchmon/patchmon-server:latest
    restart: unless-stopped
    env_file: .env
    ports:
      - "3000:3000"
    networks:
      - patchmon-internal
    depends_on:
      database:
        condition: service_healthy
      redis:
        condition: service_healthy
      guacd:
        condition: service_healthy

  database:
    image: postgres:17-alpine
    restart: unless-stopped
    env_file: .env
    volumes:
      - postgres_data:/var/lib/postgresql/data
    networks:
      - patchmon-internal

  redis:
    image: redis:7-alpine
    restart: unless-stopped
    env_file: .env
    command: redis-server --requirepass ${REDIS_PASSWORD}
    volumes:
      - redis_data:/data
    networks:
      - patchmon-internal

  guacd:
    image: guacamole/guacd:1.5.5
    restart: unless-stopped
    networks:
      - patchmon-internal

volumes:
  postgres_data:
  redis_data:

networks:
```

```
  patchmon-internal:
    driver: bridge
```

Volumes

Volume	Purpose
`postgres_data`	PostgreSQL data directory
`redis_data`	Redis persistence

Agent binaries and SCAP compliance content are embedded directly in the `patchmon-server` image. No additional volumes are required for them.

You can bind either volume to a host path by editing the compose file:

```
volumes:
  postgres_data:
    driver: local
    driver_opts:
      type: none
      o: bind
      device: /opt/patchmon/postgres
```

> ***Note:*** *The server container runs as a non-root user. If you bind volumes to host paths, ensure that user has read/write access to those directories.*

Updating PatchMon

By default the compose file uses the `latest` tag. To update to the newest release:

```
docker compose pull
docker compose up -d
```

This pulls the latest image, recreates the `server` container, and runs any new database migrations automatically on startup. Your data is preserved in the named volumes.

Pinning to a specific version

If you prefer to control when you update, pin the image tag in your `docker-compose.yml`:

```
services:
  server:
    image: ghcr.io/patchmon/patchmon-server:1.5.0
```

Then pull and restart when you are ready to upgrade:

```
docker compose pull
docker compose up -d
```

Check the GitHub releases page (https://github.com/PatchMon/PatchMon/releases) for version-specific changes and migration notes before upgrading.

Reverse Proxy Setup

PatchMon listens on port `3000` inside the container, mapped to port `3000` on the host by default. Point your reverse proxy at `http://<host>:3000`.

When deploying behind a reverse proxy, ensure:

- WebSocket connections are proxied correctly. PatchMon uses WebSockets for agent communication, SSH terminal, and RDP sessions.
- The `X-Forwarded-For`, `X-Forwarded-Proto`, and `Host` headers are forwarded so PatchMon can construct correct URLs

If you use Nginx Proxy Manager, enable "Websockets Support" on the proxy host entry.

Troubleshooting

Server fails to start: database connection refused

The server waits for the database health check before starting, but if it still fails:

```
# Check that the database container is healthy
docker compose ps

# Check database logs
docker compose logs database
```

Verify that `POSTGRES_USER`, `POSTGRES_PASSWORD`, and `POSTGRES_DB` in your `.env` are set and consistent.

Server fails to start: Redis connection refused

```
docker compose logs redis
```

Verify that `REDIS_PASSWORD` in your `.env` is set. The Redis container uses this value in its startup command (`--requirepass`).

Checking server logs

```
# Follow live logs
docker compose logs -f server

# Last 100 lines
docker compose logs --tail=100 server
```

Resetting to a clean state

> ***Warning:*** *This deletes all PatchMon data. Only do this on a fresh install where you want to start over.*

```
docker compose down -v
docker compose up -d
```

Port 3000 already in use

Change the host-side port in `docker-compose.yml` :

```
ports:
  - "8080:3000"   # Expose on host port 8080 instead
```

Update `SERVER_PORT` in your `.env` to match if agents need to reach the server directly on that port.

Chapter 2: Installing PatchMon on Kubernetes with Helm

Overview

The community Helm chart for PatchMon deploys the server on any Kubernetes 1.19+ cluster. It is maintained in a separate repository:

Chart repository: github.com/RuTHlessBEat200/PatchMon-helm (https://github.com/RuTHlessBEat200/PatchMon-helm)

Application repository: github.com/PatchMon/PatchMon (https://github.com/PatchMon/PatchMon)

> ***Verify against the latest chart.*** *This page describes the chart values shape. The chart is community-maintained and may be a version or two behind the latest PatchMon release. Always check the chart's own README (https://github.com/RuTHlessBEat200/PatchMon-helm) for the current value names and defaults before upgrading.*

Important: PatchMon 2.0 architecture

The Helm chart was originally written against the 1.4.x Node.js stack, which shipped a separate `patchmon-backend` and `patchmon-frontend` image and used BullMQ for background jobs. **PatchMon 2.0 consolidates everything into a single Go binary**: the React frontend is embedded, `chi` serves both `/api/*` and the SPA, background jobs run on Asynq (https://github.com/hibiken/asynq), and schema migrations are applied automatically at boot by golang-migrate (https://github.com/golang-migrate/migrate).

Practical implications for the chart:

Concept	1.4.x (Node)	2.0+ (Go)
Containers	`patchmon-backend` + `patchmon-frontend`	Single `patchmon-server`
Queue	BullMQ	Asynq
Migrations	Prisma	golang-migrate (embedded, automatic)
Listen port	backend 3001, frontend 3000	server 3000 (both `/api/*` and SPA)
RDP sidecar	n/a	`guacamole/guacd` (optional, required for in-browser RDP)

If your version of the chart still ships with separate backend and frontend deployments, set the frontend deployment to `enabled: false` and point your Ingress exclusively at the backend image tagged `ghcr.io/patchmon/patchmon-server:2.0.0` or later, exposed on port 3000. Where the chart's values file uses `backend.env.*` keys, those now map to the single `patchmon-server` container's environment.

Prerequisites

Kubernetes 1.19+

Helm 3.0+

A PersistentVolume provisioner in the cluster (for PostgreSQL and Redis data)

An Ingress controller (e.g. NGINX Ingress) for external access (strongly recommended)

cert-manager for automatic TLS certificate management (optional)

Metrics Server for HPA (optional)

Container Images

Component	Image	Default Tag
Server	`ghcr.io/patchmon/patchmon-server`	`2.0.0`
Database	`docker.io/postgres`	`17-alpine`
Redis	`docker.io/redis`	`7-alpine`
guacd (RDP sidecar, optional)	`docker.io/guacamole/guacd`	`1.5.5`

Available tags (server image)

Tag	Description
`latest`	Latest stable release
`x.y.z`	Exact version pin (e.g. `2.0.0`)
`x.y`	Latest patch in a minor series (e.g. `2.0`)
`x`	Latest minor and patch in a major series (e.g. `2`)
`edge`	Latest development build from the main branch. Unstable, for testing only.

Quick Start

The quickest way to try PatchMon on Kubernetes is the provided `values-quick-start.yaml`. It contains placeholder secrets and sensible defaults for a single-command install.

> ***Warning:*** *`values-quick-start.yaml` ships with placeholder secrets and is intended for evaluation only. Never use it in production without replacing all secret values.*

1. Install the chart

```
wget https://raw.githubusercontent.com/RuTHlessBEat200/PatchMon-
helm/refs/heads/main/values-quick-start.yaml

helm install patchmon oci://ghcr.io/ruthlessbeat200/charts/patchmon \
  --namespace patchmon \
  --create-namespace \
  --values values-quick-start.yaml
```

2. Wait for pods to become ready

```
kubectl get pods -n patchmon -w
```

The server pod runs embedded migrations on first boot. Follow the logs to watch them apply:

```
kubectl logs -n patchmon deploy/patchmon-server -f
```

3. Access PatchMon

If an Ingress is configured, open the host you set (e.g. `https://patchmon.example.com`).

Without Ingress, use port-forwarding:

```
kubectl port-forward -n patchmon svc/patchmon-server 3000:3000
```

Then open `http://localhost:3000` and complete the first-time admin setup.

Production Deployment

For production, start from `values-prod.yaml` in the chart repository. The example below demonstrates how to:

- Use an external Kubernetes Secret (managed by SOPS, Sealed Secrets, or External Secrets Operator) instead of inline passwords
- Configure HTTPS with cert-manager
- Set the `CORS_ORIGIN` to the external URL your users access (comma-separate multiple URLs with no spaces if PatchMon is reached from more than one origin)

1. Create your secrets

The chart does **not** auto-generate secrets. You must supply them yourself.

Required secrets for a PatchMon 2.0 deployment:

Key	Description
`postgres-password`	PostgreSQL password
`redis-password`	Redis password
`jwt-secret`	JWT signing secret used by the server
`ai-encryption-key`	Encryption key used for AI provider credentials, bootstrap tokens, and other secrets at rest
`oidc-client-secret`	OIDC client secret (only when OIDC is enabled)

Example: creating a Secret manually

```
kubectl create namespace patchmon

kubectl create secret generic patchmon-secrets \
  --namespace patchmon \
  --from-literal=postgres-password="$(openssl rand -hex 32)" \
  --from-literal=redis-password="$(openssl rand -hex 32)" \
  --from-literal=jwt-secret="$(openssl rand -hex 64)" \
  --from-literal=ai-encryption-key="$(openssl rand -hex 32)"
```

Recommended secret-management tools for production:

- SOPS (https://github.com/getsops/sops): encrypt secrets in Git
- Sealed Secrets (https://github.com/bitnami-labs/sealed-secrets): cluster-only decryption
- External Secrets Operator (https://external-secrets.io/): sync secrets from Vault, AWS Secrets Manager, etc.

2. Create your values file

Start from `values-prod.yaml` and adjust:

```yaml
global:
  storageClass: "your-storage-class"
  imageTag: "2.0.0"

fullnameOverride: "patchmon-prod"

server:
  env:
    # Comma-separate with no spaces to allow multiple origins, e.g.
    # "https://patchmon.example.com,https://patchmon.internal.lan"
    CORS_ORIGIN: "https://patchmon.example.com"
    ENABLE_HSTS: "true"
    TRUST_PROXY: "true"
  existingSecret: "patchmon-secrets"
  existingSecretJwtKey: "jwt-secret"
  existingSecretAiEncryptionKey: "ai-encryption-key"

database:
  auth:
    existingSecret: "patchmon-secrets"
    existingSecretPasswordKey: "postgres-password"

redis:
  auth:
    existingSecret: "patchmon-secrets"
    existingSecretPasswordKey: "redis-password"

secret:
  create: false

ingress:
  enabled: true
  className: nginx
  annotations:
    cert-manager.io/cluster-issuer: letsencrypt-prod
    nginx.ingress.kubernetes.io/proxy-read-timeout: "86400"
    nginx.ingress.kubernetes.io/proxy-send-timeout: "86400"
    nginx.ingress.kubernetes.io/proxy-body-size: "0"
  hosts:
    - host: patchmon.example.com
      paths:
        - path: /
          pathType: Prefix
          service:
            name: server
            port: 3000
  tls:
    - secretName: patchmon-tls
      hosts:
        - patchmon.example.com
```

> ***Verify against the latest chart.*** *The exact value keys (`server.*` vs `backend.*`) depend on whether the chart has been updated for 2.0. If your chart still splits backend and frontend, set `frontend.enabled: false` and expose only the backend on port 3000.*

3. Install

```
helm install patchmon oci://ghcr.io/ruthlessbeat200/charts/patchmon \
  --namespace patchmon \
  --create-namespace \
  --values values-prod.yaml
```

Configuration Reference

> ***Verify against the latest chart.*** *The table below reflects the original chart value names. 2.0 versions of the chart are expected to drop the separate `frontend.*` block and consolidate everything under a single `server.*` block. Always diff against the chart's `values.yaml` before changing things.*

Global Settings

Parameter	Description	Default
`global.imageRegistry`	Override the image registry for all components	`""`
`global.imageTag`	Override the image tag for the PatchMon image (takes priority over individual tags)	`""`
`global.imagePullSecrets`	Image pull secrets applied to all pods	`[]`
`global.storageClass`	Default storage class for all PVCs	`""`
`nameOverride`	Override the chart name used in resource names	`""`
`fullnameOverride`	Override the full resource name prefix	`""`
`commonLabels`	Labels added to all resources	`{}`
`commonAnnotations`	Annotations added to all resources	`{}`

Database (PostgreSQL)

Parameter	Description	Default
`database.enabled`	Deploy the PostgreSQL StatefulSet	`true`
`database.image.registry`	Image registry	`docker.io`
`database.image.repository`	Image repository	`postgres`
`database.image.tag`	Image tag	`17-alpine`
`database.auth.database`	Database name	`patchmon_db`
`database.auth.username`	Database user	`patchmon_user`
`database.auth.password`	Database password (required unless `existingSecret` is set)	`""`
`database.auth.existingSecret`	Existing Secret containing the password	`""`
`database.auth.existingSecretPasswordKey`	Key inside the existing Secret	`postgres-password`
`database.persistence.enabled`	Enable persistent storage	`true`
`database.persistence.size`	PVC size	`5Gi`
`database.resources.requests.cpu`	CPU request	`100m`
`database.resources.requests.memory`	Memory request	`128Mi`
`database.resources.limits.memory`	Memory limit	`1Gi`
`database.service.port`	Service port	`5432`

Redis

Parameter	Description	Default
`redis.enabled`	Deploy the Redis StatefulSet	`true`
`redis.image.tag`	Image tag	`7-alpine`
`redis.auth.password`	Redis password (required unless `existingSecret` is set)	`""`
`redis.auth.existingSecret`	Existing Secret containing the password	`""`
`redis.auth.existingSecretPasswordKey`	Key inside the existing Secret	`redis-password`
`redis.persistence.enabled`	Enable persistent storage	`true`
`redis.persistence.size`	PVC size	`5Gi`
`redis.resources.requests.memory`	Memory request	`10Mi`
`redis.resources.limits.memory`	Memory limit	`512Mi`
`redis.service.port`	Service port	`6379`

Server (PatchMon 2.0)

In 2.0 the server is a single Go binary that serves `/api/*` and the embedded React SPA on port 3000. In chart versions that have not yet been updated for 2.0, the equivalent values live under `backend.*` and `frontend.enabled` should be set to `false`.

Parameter	Description	Default
`server.enabled`	Deploy the PatchMon server	`true`
`server.image.registry`	Image registry	`ghcr.io`
`server.image.repository`	Image repository	`patchmon/patchmon-server`
`server.image.tag`	Image tag (overridden by `global.imageTag` if set)	`2.0.0`
`server.replicaCount`	Number of replicas	`1`
`server.jwtSecret`	JWT signing secret (required unless `existingSecret` is set)	`""`
`server.aiEncryptionKey`	Encryption key for secrets at rest	`""`
`server.existingSecret`	Name of an existing Secret for JWT and encryption key	`""`
`server.existingSecretJwtKey`	Key for `JWT_SECRET` inside the existing Secret	`jwt-secret`
`server.existingSecretAiEncryptionKey`	Key for `AI_ENCRYPTION_KEY` inside the existing Secret	`ai-encryption-key`
`server.resources.requests.cpu`	CPU request	`100m`
`server.resources.requests.memory`	Memory request	`256Mi`
`server.resources.limits.memory`	Memory limit	`1Gi`
`server.service.port`	Service port	`3000`
`server.autoscaling.enabled`	Enable HPA	`false`

> ***Migrations:*** *The PatchMon server runs migrations automatically at boot via embedded* `golang-migrate` *. You do* ***not*** *need a dedicated migration Job; remove it from the chart if one is present.*

Server environment variables

The `server.env.*` keys map directly to the environment variables the PatchMon binary reads. See the Environment Variables Reference for the full list.

Key	Description	Defau
`CORS_ORIGIN`	Allowed origin for CORS (must match the URL users type in their browser; comma-separate with no spaces to allow multiple, e.g. `https://patchmon.example.com,https://patchmon.internal.lan`)	`http`
`ENABLE_HSTS`	Enable HSTS header for HTTPS	`false`
`TRUST_PROXY`	Trust proxy headers when behind an Ingress controller	`false`
`ENABLE_LOGGING`	Enable structured logging to stdout	`false`
`LOG_LEVEL`	Log level (`debug`, `info`, `warn`, `error`)	`info`
`JSON_BODY_LIMIT`	Max JSON body size	`5mb`
`AGENT_UPDATE_BODY_LIMIT`	Max agent update body size	`2mb`
`TZ`	IANA timezone for log timestamps	`UTC`

> ***Tip:*** *Many of these can be changed later from the Settings UI without a restart of the whole cluster. See Settings in the web UI.*

OIDC / SSO

Key	Description	Default
`OIDC_ENABLED`	Enable OIDC authentication	`false`
`OIDC_ISSUER_URL`	OIDC issuer URL	`""`
`OIDC_CLIENT_ID`	OIDC client ID	`""`
`OIDC_CLIENT_SECRET`	OIDC client secret (put this in an existing Secret)	`""`
`OIDC_REDIRECT_URI`	Callback URL (`https://<host>/api/v1/auth/oidc/callback`)	`""`
`OIDC_SCOPES`	Space-separated scopes	`openid email profile groups`
`OIDC_AUTO_CREATE_USERS`	Auto-provision users on first login	`false`
`OIDC_DEFAULT_ROLE`	Default role for new OIDC users	`user`
`OIDC_SYNC_ROLES`	Sync roles from OIDC group claims on each login	`false`
`OIDC_DISABLE_LOCAL_AUTH`	Disable local username/password authentication	`false`

Full OIDC configuration and group-to-role mapping variables are documented in the Environment Variables Reference.

guacd sidecar (optional, for RDP)

PatchMon 2.0 can proxy Windows RDP through the Apache Guacamole daemon (https://guacamole.apache.org/). If you need in-browser RDP, add a `guacd` sidecar and point `GUACD_ADDRESS` at it:

```
server:
  env:
    GUACD_ADDRESS: "guacd:4822"

guacd:
  enabled: true
  image:
    repository: guacamole/guacd
    tag: "1.5.5"
```

> ***Verify against the latest chart.*** *`guacd` support was added after the original 1.4.x chart; if your chart version does not include a `guacd.enabled` option, you can deploy it as a separate Deployment and Service in the same namespace and set `GUACD_ADDRESS` to its ClusterIP hostname.*

Ingress

Parameter	Description	Default
`ingress.enabled`	Enable Ingress resource	`true`
`ingress.className`	Ingress class name	`""`
`ingress.annotations`	Ingress annotations	`{}`
`ingress.hosts`	List of Ingress host rules	see chart `values.yaml`
`ingress.tls`	TLS configuration	`[]`

Required annotations for WebSocket support (agent WS and live patch streaming):

```
ingress:
  annotations:
    nginx.ingress.kubernetes.io/proxy-read-timeout: "86400"
    nginx.ingress.kubernetes.io/proxy-send-timeout: "86400"
    nginx.ingress.kubernetes.io/proxy-body-size: "0"
```

For detailed reverse proxy configuration (Nginx, Caddy, Traefik), see Reverse proxy examples.

Persistent Volumes

PVC	Component	Purpose	Default Size
`postgres-data`	Database	PostgreSQL data directory	`5Gi`
`redis-data`	Redis	Redis data directory	`5Gi`

> ***No agent files volume in 2.0.*** *In 1.4.x, agent binaries and SCAP compliance content were stored on a* `agent-files` *PVC. In 2.0 both are* ***embedded in the server binary****, so no application-specific volume is required. If your chart still declares* `backend.persistence`*, it can be removed.*

Updating PatchMon

Using `global.imageTag`

```
helm upgrade patchmon oci://ghcr.io/ruthlessbeat200/charts/patchmon \
  -n patchmon \
  -f values-prod.yaml \
  --set global.imageTag=2.0.1
```

When the new pod starts, `golang-migrate` applies any pending schema migrations automatically. No manual Job is required.

Pinning individual tags

```
server:
  image:
    tag: "2.0.0"
```

Upgrading the chart version

```
helm upgrade patchmon oci://ghcr.io/ruthlessbeat200/charts/patchmon \
  --namespace patchmon \
  --values values-prod.yaml \
  --wait --timeout 10m
```

Check the chart releases page (https://github.com/RuTHlessBEat200/PatchMon-helm/releases) and the PatchMon releases page (https://github.com/PatchMon/PatchMon/releases) before upgrading.

Uninstalling

```
# Uninstall the release
helm uninstall patchmon -n patchmon

# Clean up PVCs (this deletes all data)
kubectl delete pvc -n patchmon -l app.kubernetes.io/instance=patchmon
```

Advanced Configuration

Custom image registry (air-gapped)

```
global:
  imageRegistry: "registry.example.com"
```

This changes every image pull to use the specified registry:

- `registry.example.com/postgres:17-alpine`
- `registry.example.com/redis:7-alpine`
- `registry.example.com/patchmon/patchmon-server:2.0.0`
- `registry.example.com/guacamole/guacd:1.5.5` (when RDP is enabled)

Horizontal Pod Autoscaling

```
server:
  autoscaling:
    enabled: true
    minReplicas: 2
    maxReplicas: 10
    targetCPUUtilizationPercentage: 70
```

> ***Note:*** *Scaling the server beyond a single replica is safe as of 2.0. There are no writeable local file volumes, and background jobs are coordinated through Redis + Asynq. WebSocket connections (agent WS, SSH terminal WS, live patch streams) do need sticky sessions through the Ingress controller if you run multiple replicas; set* `nginx.ingress.kubernetes.io/affinity: cookie` *in your Ingress annotations.*

Using an external database

Disable the built-in database and set `DATABASE_URL` to an external PostgreSQL instance:

```
database:
  enabled: false

server:
  env:
    DATABASE_URL: "postgresql://patchmon:password@external-
db.example.com:5432/patchmon"
```

OIDC / SSO integration

```
server:
  env:
    OIDC_ENABLED: "true"
    OIDC_ISSUER_URL: "https://auth.example.com/realms/master"
    OIDC_CLIENT_ID: "patchmon"
    OIDC_REDIRECT_URI: "https://patchmon.example.com/api/v1/auth/oidc/callback"
    OIDC_SCOPES: "openid profile email groups"
    OIDC_BUTTON_TEXT: "Login with SSO"
    OIDC_AUTO_CREATE_USERS: "true"
    OIDC_SYNC_ROLES: "true"
    OIDC_ADMIN_GROUP: "patchmon-admins"
```

The client secret should live in a Kubernetes Secret and be mounted as `OIDC_CLIENT_SECRET`, not set inline.

Troubleshooting

Check pod status

```
kubectl get pods -n patchmon
kubectl describe pod <pod-name> -n patchmon
kubectl logs <pod-name> -n patchmon
```

Check init container logs (waiting for DB or Redis)

```
kubectl logs <pod-name> -n patchmon -c wait-for-database
kubectl logs <pod-name> -n patchmon -c wait-for-redis
```

Check migration logs

Migrations run inside the server pod on startup. Follow the startup logs:

```
kubectl logs -n patchmon deploy/patchmon-server --since=5m | grep migrate
```

You should see lines like `[migrate] running migrations from embedded binary` and either `[migrate] applied successfully (version N)` or `[migrate] already up to date`.

Health check

The server exposes a liveness probe at `/health`:

```
kubectl exec -n patchmon -it deploy/patchmon-server -- wget -qO-
http://localhost:3000/health
```

Response is `healthy` (plain text) or a JSON structure when the `Accept: application/json` header is set.

Common issues

Symptom	Likely cause	Fix
Pods stuck in `Init` state	Database or Redis not yet running	`kubectl describe sts -n patchmon`
PVC stuck in `Pending`	No matching StorageClass	Run `kubectl get sc` and set `global.storageClass`
`ImagePullBackOff`	Registry credentials missing	Check `imagePullSecrets` and image path
Ingress returns 404 / 502	Ingress misconfigured or points at wrong port	All traffic should go to `server:3000`
WebSocket connections drop every ~30s	Ingress default read timeout too short	Set `proxy-read-timeout: "86400"`
`secret ... not found`	Required Secret not created before install	Create the Secret or set `secret.create: true`
CORS errors in browser	`CORS_ORIGIN` doesn't match the URL users see	Set it to the exact URL from the Ingress host. If the chart exposes more than one Ingress host, comma-separate them with no spaces, e.g. `https://patchmon.example.com,https://patchmon.internal.lan`

Support

Chart issues: github.com/RuTHlessBEat200/PatchMon-helm/issues (https://github.com/RuTHlessBEat200/PatchMon-helm/issues)

Application issues: github.com/PatchMon/PatchMon (https://github.com/PatchMon/PatchMon)

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

See also

Installing PatchMon Server on Docker: the officially supported deployment method

Reverse proxy examples: Nginx, Caddy, Traefik snippets

PatchMon Environment Variables Reference: every variable the server reads

First-time admin setup: what to do once the pod is running

Chapter 3: Reverse Proxy Examples

Overview

PatchMon 2.0 runs as a single Docker container that listens on **port 3000** inside the container. The REST API, the embedded React frontend, and all WebSocket endpoints are all served from that one port. In front of it you put a reverse proxy for TLS termination, HTTP/2, and a stable public hostname.

This page has ready-to-use snippets for the most common self-hosted reverse proxies:

- Nginx
- Caddy
- Traefik
- Nginx Proxy Manager

All snippets assume the PatchMon container is reachable at `http://patchmon:3000` (Docker service name) or `http://<host>:3000` (bare-metal / VM). Substitute as appropriate.

What Your Proxy Must Do

Every PatchMon reverse proxy, regardless of flavour, must handle four things correctly:

Terminate TLS on the public hostname (`patchmon.example.com`) and forward to the server on plain HTTP.

Upgrade WebSocket connections. PatchMon uses long-lived WebSockets for:

- Agent control channel: `/api/v1/agents/ws`
- Browser SSH terminal: `/api/v1/ssh-terminal/{hostId}`
- Browser RDP tunnel: `/api/v1/rdp/websocket-tunnel`
- Live patch-run log stream: `/api/v1/patching/runs/{id}/stream`

Forward the original protocol via `X-Forwarded-Proto: https`. The server reads this header to know the connection is secure and to construct correct `wss://` URLs for agents.

Use a long read timeout (86400 seconds / 24 hours). The agent control channel is an idle-tolerant connection that sends pings every 30 seconds; most proxies default to a 60-second idle timeout and will drop the connection before the agent can detect the disconnect.

If you also want the server to pick up the real client IP for logging and rate-limiting (instead of the proxy's IP), set `TRUST_PROXY=true` in the PatchMon `.env`. See the Environment Variables Reference for details.

WebSocket endpoints to verify

When you first wire up a proxy, test these four endpoints from a browser or agent and confirm they stay connected. All four require the same upgrade-and-long-timeout treatment:

Endpoint	Used by	Auth
`/api/v1/agents/ws`	PatchMon agent	`X-API-ID` + `X-API-KEY` headers
`/api/v1/ssh-terminal/{hostId}`	Browser SSH terminal	Short-lived ticket in `?ticket=...`
`/api/v1/rdp/websocket-tunnel`	Browser RDP (Guacamole)	Short-lived ticket
`/api/v1/patching/runs/{id}/stream`	Patch-run live log UI	JWT cookie / bearer

If your agents show as "connecting" and then drop every few minutes, the read timeout is almost certainly too short.

Nginx

A minimal production block for a single PatchMon instance behind Nginx with TLS from Let's Encrypt:

```
# /etc/nginx/sites-available/patchmon.conf
#
# Put the WebSocket upgrade map in the http block (e.g. nginx.conf) or at the
# top of this file inside any 'http' context you manage.

map $http_upgrade $connection_upgrade {
    default upgrade;
    ''      close;
}

server {
    listen 80;
    listen [::]:80;
    server_name patchmon.example.com;

    # Redirect everything to HTTPS
    return 301 https://$host$request_uri;
}

server {
    listen 443 ssl http2;
    listen [::]:443 ssl http2;
    server_name patchmon.example.com;

    ssl_certificate     /etc/letsencrypt/live/patchmon.example.com/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/patchmon.example.com/privkey.pem;

    # Modern TLS profile; adjust to taste
    ssl_protocols TLSv1.2 TLSv1.3;
    ssl_prefer_server_ciphers off;

    # Allow large agent reports (packages, Docker inventory) through the proxy.
    # The server also enforces its own JSON_BODY_LIMIT / AGENT_UPDATE_BODY_LIMIT.
    client_max_body_size 20m;

    location / {
        proxy_pass http://127.0.0.1:3000;

        # Required for WebSockets (agent WS, SSH terminal, RDP, patch stream)
        proxy_http_version 1.1;
        proxy_set_header Upgrade    $http_upgrade;
        proxy_set_header Connection $connection_upgrade;

        # Preserve original host + client info
        proxy_set_header Host              $host;
        proxy_set_header X-Real-IP         $remote_addr;
        proxy_set_header X-Forwarded-For   $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
        proxy_set_header X-Forwarded-Host  $host;

        # Long-lived WebSockets — 24h idle timeout so agent connections
```

```
        # aren't dropped by the proxy. PatchMon sends its own keepalive pings.
        proxy_read_timeout  86400s;
        proxy_send_timeout  86400s;
        proxy_connect_timeout 60s;

        # Do not buffer Server-Sent Events or streaming responses
        proxy_buffering off;
        proxy_cache     off;
    }
}
```

Enable and reload:

```
sudo ln -s /etc/nginx/sites-available/patchmon.conf /etc/nginx/sites-enabled/
sudo nginx -t
sudo systemctl reload nginx
```

> ***Tip:*** *If PatchMon is on a different host from Nginx, replace* `127.0.0.1:3000` *with the container's reachable address. Put the PatchMon container and Nginx on the same Docker network and use the service name for the cleanest setup.*

Caddy

Caddy handles TLS certificates, HTTP/2, and WebSocket upgrades automatically. The entire `Caddyfile` is usually four lines:

```
# /etc/caddy/Caddyfile

patchmon.example.com {
    reverse_proxy 127.0.0.1:3000 {
        # 24h timeout for long-lived agent WebSockets.
        # PatchMon sends its own pings; this just keeps Caddy from dropping
        # an otherwise-healthy idle connection.
        transport http {
            read_timeout 86400s
            write_timeout 86400s
        }
    }
}
```

That's the full configuration. Caddy:

- Fetches and renews the TLS certificate from Let's Encrypt automatically.
- Sets `X-Forwarded-Proto` and `X-Forwarded-For` by default.

Upgrades WebSocket connections transparently.

Reload:

```
sudo systemctl reload caddy
```

Docker Compose snippet

If you run Caddy in Docker alongside PatchMon:

```
services:
  caddy:
    image: caddy:2-alpine
    restart: unless-stopped
    ports:
      - "80:80"
      - "443:443"
    volumes:
      - ./Caddyfile:/etc/caddy/Caddyfile:ro
      - caddy_data:/data
      - caddy_config:/config
    networks:
      - patchmon-internal

volumes:
  caddy_data:
  caddy_config:
```

And point the Caddyfile at the compose service name:

```
patchmon.example.com {
    reverse_proxy server:3000 {
        transport http {
            read_timeout 86400s
            write_timeout 86400s
        }
    }
}
```

Traefik

Traefik works well with Docker Compose because it discovers services via container labels.

docker-compose.yml: minimal

```
name: patchmon

services:
  traefik:
    image: traefik:v3
    restart: unless-stopped
    command:
      - "--api.dashboard=false"
      - "--providers.docker=true"
      - "--providers.docker.exposedbydefault=false"
      - "--entrypoints.web.address=:80"
      - "--entrypoints.websecure.address=:443"
      # Redirect http → https
      - "--entrypoints.web.http.redirections.entrypoint.to=websecure"
      - "--entrypoints.web.http.redirections.entrypoint.scheme=https"
      # Let's Encrypt
      - "--certificatesresolvers.le.acme.tlschallenge=true"
      - "--certificatesresolvers.le.acme.email=admin@example.com"
      - "--certificatesresolvers.le.acme.storage=/letsencrypt/acme.json"
    ports:
      - "80:80"
      - "443:443"
    volumes:
      - ./letsencrypt:/letsencrypt
      - /var/run/docker.sock:/var/run/docker.sock:ro
    networks:
      - patchmon-internal

  server:
    image: ghcr.io/patchmon/patchmon-server:latest
    restart: unless-stopped
    env_file: .env
    labels:
      - "traefik.enable=true"
      - "traefik.http.routers.patchmon.rule=Host(`patchmon.example.com`)"
      - "traefik.http.routers.patchmon.entrypoints=websecure"
      - "traefik.http.routers.patchmon.tls.certresolver=le"
      - "traefik.http.services.patchmon.loadbalancer.server.port=3000"
    networks:
      - patchmon-internal
    depends_on:
      - database
      - redis

networks:
  patchmon-internal:
    driver: bridge
```

The long-read-timeout middleware

Traefik defaults to very short per-request timeouts. For the agent WebSocket to stay alive, add a `serversTransport` with a 24-hour read timeout and bind it to the service. Put this in a **static** config (a file referenced by `--providers.file` or a CLI flag, because you cannot set `serversTransport` from container labels):

```
# /etc/traefik/dynamic.yml
http:
  serversTransports:
    patchmon-longtimeout:
      forwardingTimeouts:
        dialTimeout: "30s"
        responseHeaderTimeout: "0s"   # disable response header timeout
        idleConnTimeout: "86400s"

  services:
    patchmon:
      loadBalancer:
        serversTransport: patchmon-longtimeout
        servers:
          - url: "http://server:3000"
```

And tell Traefik to load it:

```
# in the traefik command block
- "--providers.file.filename=/etc/traefik/dynamic.yml"
# mount it
volumes:
  - ./dynamic.yml:/etc/traefik/dynamic.yml:ro
```

> *Traefik's default read/write timeouts are fine for normal HTTP, but they will drop the long-lived agent WebSocket. The 24-hour `idleConnTimeout` and a zeroed `responseHeaderTimeout` are the pieces that make it behave.*

Traefik automatically:

- Terminates TLS at the `websecure` entry point.
- Forwards `X-Forwarded-Proto`, `X-Forwarded-For`, and `X-Forwarded-Host`.
- Upgrades WebSocket connections when the client sends `Upgrade: websocket`.

Nginx Proxy Manager

Nginx Proxy Manager (https://nginxproxymanager.com/) (NPM) is a popular self-hosted web UI for managing Nginx reverse-proxy entries. It handles most of PatchMon's needs in two toggles, but the default read timeout is too short for long-lived agent WebSockets.

Step-by-step

In NPM, create a new **Proxy Host** pointing at the PatchMon container (scheme `http`, hostname `patchmon` or the host IP, port `3000`).

On the **Details** tab, enable:

Block Common Exploits

Websockets Support

Attach your SSL certificate on the **SSL** tab and enable **Force SSL** and **HTTP/2 Support**.

On the **Advanced** tab, paste the following snippet to extend the read timeout for agent WebSockets and for live patch log streaming:

```
# PatchMon — extend read timeout for long-lived WebSockets
# (agent control channel, SSH terminal, RDP tunnel, patch stream)
proxy_read_timeout  86400s;
proxy_send_timeout  86400s;
proxy_buffering     off;
proxy_request_buffering off;

# Ensure X-Forwarded-Proto is set correctly for HTTPS detection inside the app.
proxy_set_header X-Forwarded-Proto $scheme;
proxy_set_header X-Forwarded-Host  $host;
```

Save. Test the URL in a browser and, once the UI loads, enrol an agent and watch it stay online in the **Hosts** page for longer than a few minutes. That is the real test.

> ***Tip:*** *If you use Cloudflare or any other intermediate proxy, it must also be configured for WebSocket pass-through. Cloudflare has WebSockets enabled by default; other CDNs may need an explicit opt-in.*

Verifying Your Setup

Once the proxy is live, verify all four WebSocket endpoints in turn.

1. Browser UI loads over HTTPS

Open `https://patchmon.example.com`. You should see the PatchMon login page served with a valid certificate and no mixed-content warnings.

2. Agent control channel

Enrol one agent using the install command from **Hosts → Add Host → Install**. Within a few seconds the host should appear in the hosts list with a green **online** WebSocket indicator. Leave it running for at least 10 minutes; if it disconnects in that window, your proxy's read

timeout is too short.

Check the agent logs on the target server:

```
sudo journalctl -u patchmon-agent -n 50
```

You want to see `WebSocket connected` and no repeated reconnect loops.

3. Live patch streaming

Trigger a dry-run patch on any host from the UI. The output pane should stream stdout/stderr as the command runs. If it hangs with no output and then prints everything at once at the end, the proxy is buffering. Re-check `proxy_buffering off` (Nginx, NPM) or `proxy_request_buffering off` (NPM).

4. SSH terminal

Open **Hosts → → SSH Terminal**. The terminal should connect and echo keystrokes in real time. If the terminal connects and then hangs after 30–60 seconds, the issue is again the read timeout.

Common Pitfalls

Symptom	Likely cause	Fix
Agents reconnect every ~60 seconds	Proxy read timeout too short	Set to `86400s`
Live patch output arrives all at once	Proxy buffering enabled	`proxy_buffering off`
`wss://` URLs try `http://` inside the agent install script	`X-Forwarded-Proto` missing or wrong	Explicitly set to `$scheme` (Nginx) / default in Caddy + Traefik
Browser console: "CORS policy" errors	`CORS_ORIGIN` does not match the URL in the address bar	Set `CORS_ORIGIN=https://patchmon.example.com` exactly. To allow more than one origin, comma-separate with no spaces, e.g. `CORS_ORIGIN=https://patchmon.example.com,https://patchmon.internal.lan`
Login works but nothing loads	API requests going to a different origin	Send all traffic (API + SPA) to the same hostname/port
Sudden 413 Request Entity Too Large	Proxy body limit smaller than agent report	`client_max_body_size 20m;` (Nginx) or equivalent
Agent page shows "offline" but agent logs say connected	`TRUST_PROXY=false` on the server, agent IP detection wrong	Set `TRUST_PROXY=true` and add `X-Forwarded-For`

See Also

Installing PatchMon Server on Docker: the upstream compose file this page sits in front of

PatchMon Environment Variables Reference: details on `CORS_ORIGIN`, `TRUST_PROXY`, `ENABLE_HSTS`

WebSockets architecture: how PatchMon uses WebSockets under the hood

Chapter 4: First-Time Admin Setup

Overview

The first time you open a fresh PatchMon install in a browser, you see the **first-time setup wizard** instead of the login page. The wizard creates your superadmin account, optionally sets up multi-factor authentication, confirms the URL agents will use to connect back to the server, and redirects you to the dashboard.

The wizard only runs while no admin user exists in the database. Once the superadmin account is created, the wizard disappears forever and the login page takes over.

This page walks through every step so you know what to expect.

Prerequisites

Before starting the wizard, PatchMon must already be running. You should be able to open `http://localhost:3000` (or your configured URL) in a browser and see the wizard's welcome screen. If you don't, check:

- Installing PatchMon Server on Docker: for Docker deployments
- Installing PatchMon on Kubernetes with Helm: for Kubernetes deployments
- Reverse proxy examples: if you're behind Nginx, Caddy, Traefik, or NPM

Wizard Steps at a Glance

The wizard has up to five steps. Some are skipped depending on your deployment mode:

Step	Name	Always shown?
1	Create Admin Account	Yes
2	Multi-Factor Authentication	Yes
3	Confirm Server URL	Only shown on self-hosted
4	Stay Updated (newsletter opt-in)	Hidden when newsletter is disabled
5	Get in Touch (community links)	Yes

Self-hosted users see all five steps. PatchMon Cloud users skip step 3 (the server URL is set for you) and may see fewer of the others depending on branding.

Step 1: Create Admin Account

The first screen collects the superadmin's name, username, email, and password.

Fields

Field	Rules
First name	Required
Last name	Required
Username	Required, at least 2 characters. Used to log in.
Email	Required, valid email format
Password	Must satisfy the active password policy (see below)
Confirm password	Must match the password

Password policy

The password policy is pulled live from the server (`GET /api/v1/settings/login-settings`). Defaults require:

- At least 8 characters
- One uppercase letter
- One lowercase letter
- One number
- One special character

A strength meter and per-rule checklist underneath the password field show which rules are satisfied as you type.

> ***Tip:*** *If you want a weaker or stronger policy before first login, set the `PASSWORD_*` variables in your `.env` before running `docker compose up -d`. See the Environment Variables Reference for the full list.*

What happens when you click Next

The form is validated client-side first, then the server checks that no admin user already exists before accepting the submission. If somebody has already created an admin (for example because two people opened the wizard at the same time), you'll see `Admin users already exist. This endpoint is only for first-time setup.` The first person to submit wins; log in with those credentials instead.

The admin user is created with role `superadmin`, which has every permission in the system. You can demote later or create narrower roles from **Settings → Users → Roles** once you're logged in.

Step 2: Multi-Factor Authentication

After the admin account details are entered, the wizard offers to set up TOTP-based multi-factor authentication (MFA). You have two choices:

Setup MFA now: the admin account is created immediately and the TFA setup UI appears in-line. You scan a QR code, confirm with a 6-digit code, then receive one-time backup codes to save somewhere safe.

Skip (I'll do it later): the wizard moves on and the admin account is created at the end, with MFA disabled. You can enable it any time afterwards from **Settings → My Profile → Two-Factor Authentication**.

The "Setup MFA now" flow

Clicking **Setup MFA now** does three things:

Creates the admin account immediately (rather than at the end of the wizard).

Logs you in automatically (session cookies are set).

Shows a QR code from the standard TOTP setup endpoint (`GET /api/v1/tfa/setup`).

Follow the in-page instructions:

Scan the QR code with an authenticator app (Google Authenticator, Authy, 1Password, Bitwarden, Proton Authenticator, etc.). Most standards-compliant TOTP apps work.

Enter the 6-digit code the app shows you.

Save the backup codes. PatchMon issues a set of one-time backup codes when MFA verification succeeds. Download them as a text file and store them somewhere that is **not** your authenticator app (a password manager is ideal). Each code can be used exactly once if you lose your authenticator; once you run out, you'll need to reset MFA by signing in with another admin account.

Click **Continue** to return to the wizard.

> ***Important:*** *The backup codes are displayed* ***once only****. If you close the tab before copying them, you'll need to disable and re-enable MFA from your profile page to generate a new set.*

Why MFA is recommended

The admin account has full control over your entire fleet: it can trigger patch runs, read every host's inventory, and manage all users. If the admin password is compromised, MFA is the remaining barrier between an attacker and your infrastructure. Enable it on every production deployment.

Step 3: Confirm Server URL

This step only appears in self-hosted deployments. On PatchMon Cloud, the server URL is set for you and this step is skipped.

What the "server URL" means

This is the URL **agents** will use to connect back to the PatchMon server (not the URL users open in their browser, although it is usually the same). It is stored in the database and baked into every agent install command the UI generates. Change it later and every new agent will use the new value; existing agents keep using whatever URL they were installed with until you rerun their install script.

Fields

Field	Description
Protocol	`HTTP` or `HTTPS`. Use `HTTPS` in any non-lab deployment.
Host	The DNS name or IP agents reach, e.g. `patchmon.example.com`.
Port	Public port. Usually `443` for HTTPS, `80` for HTTP, or `3000` for direct-to-container setups without a reverse proxy.

The wizard pre-fills these by calling `GET /api/v1/settings/current-url`, which reflects the URL in the browser's address bar. If you opened the wizard at `https://patchmon.example.com`, those values are already correct.

Self-signed SSL toggle

The toggle **"Will you be using a self-signed SSL certificate?"** controls whether the install command the UI generates passes `-k` to `curl`. Enable it **only** if your TLS certificate is not trusted by the system CA bundle on the target hosts, for example a private-CA internal certificate.

> ***Warning:*** *`skip_ssl_verify` disables TLS verification in the agent as well, which exposes agents to man-in-the-middle attacks on the enrolment network path. The preferred fix is to install your CA into the host's trust store (`/usr/local/share/ca-certificates/` on Debian/Ubuntu, `/etc/pki/ca-trust/source/anchors/` on RHEL/Fedora). Only enable the toggle in lab or air-gapped deployments.*

You can change all of these values later from **Settings → Server URL**.

Step 4: Stay Updated (optional)

The newsletter opt-in step offers to sign you up for PatchMon's security and product update newsletter. Toggle it on to subscribe with the name and email from step 1, or leave it off to skip.

No tracking happens. The only network call is a single subscribe request to the upstream marketing endpoint when you click through, and only when the toggle is on. Your email is never sent anywhere without your explicit opt-in.

This step is hidden if the server's login-settings response includes `show_newsletter: false` (for example, on self-hosted installs that have disabled marketing).

Step 5: Get in Touch

The final screen lists community and support links: Discord, GitHub, documentation, and the public roadmap. Click **Access Dashboard** to finish setup.

What happens when you finish

- If you didn't choose "Setup MFA now" earlier, the admin account is created now via `POST /api/v1/auth/setup-admin`.
- The server URL settings are saved via `PATCH /api/v1/settings` (self-hosted only).
- If you opted in to the newsletter, one subscribe call goes out.
- Cookies are set for your admin session and you're redirected to `/` (the dashboard).

The first-time wizard is complete. The next time you open PatchMon, you'll see the normal login page.

Errors You Might See

"Admin users already exist"

The server already has at least one admin account. The wizard only runs while the database is empty. Log in with the existing admin credentials instead.

"Password does not meet the password policy"

One or more of the password rules isn't satisfied. Re-read the checklist under the password field. Every box that isn't ticked is a failing rule.

"Setting up PatchMon" stuck on a step

If the setup spinner sits on "Creating admin account..." or "Saving server URL..." for more than 10 seconds, something is wrong between the browser and the server. A red error bar appears with two buttons:

Retry: retries the same call. Useful for transient network hiccups.

Skip and continue: if the admin account was created successfully but the URL save failed, this takes you to the dashboard anyway. You can finish the URL configuration from **Settings → Server URL**.

Check the server logs in parallel:

```
docker compose logs -f server
```

Browser says "CORS policy" after clicking Next

Your `CORS_ORIGIN` doesn't match the URL in your browser's address bar. Fix it in `.env`, restart the `server` container, and reload the page. If users access PatchMon from more than one URL, comma-separate the values with no spaces, e.g. `CORS_ORIGIN=https://patchmon.example.com,https://patchmon.internal.lan`. See Environment Variables Reference: `CORS_ORIGIN`.

What to Do Next

With the wizard complete, the usual next steps are:

Enrol your first host. Go to **Hosts → Add Host**, pick the target OS, and copy the generated one-line install command to run on the server.

Confirm the agent reports in. Within a minute, the host should appear on the Hosts page with a green "online" indicator. If it doesn't, see Managing the PatchMon Agent.

Review your settings. The Settings in the web UI page walks through every settings area, including where to configure OIDC SSO, branding, notifications, and alerts.

Secure the deployment. Enable HTTPS via a reverse proxy (Reverse proxy examples), set `ENABLE_HSTS=true`, set `TRUST_PROXY=true`, and consider enabling OIDC SSO or `OIDC_DISABLE_LOCAL_AUTH` for production.

See Also

Installing PatchMon Server on Docker

Settings in the web UI

PatchMon Environment Variables Reference

Reverse proxy examples

Chapter 5: PatchMon Environment Variables Reference

Applies to: PatchMon 2.0+ (Go server)

This document is the authoritative reference for all environment variables supported by the PatchMon server. Configure these in your `.env` file, which Docker reads and passes into the `server` container via `env_file:`.

Variables are loaded from `.env` in the working directory by default. To use a different file, set the `ENV_FILE` variable to the path you want the server to read at startup.

How values are resolved

PatchMon resolves configuration in this order, highest to lowest priority:

1. **Environment variable** (set in `.env` or in your container/Pod spec)
2. **Database value** (set via the Settings UI; only for values marked *Editable in UI* below)
3. **Built-in default**

This means: if a variable is set in `.env`, editing it in **Settings → Environment** has no effect until you remove the env value. The UI flags overridden values with a yellow "env" badge so you can tell at a glance why your change is being ignored. See Settings in the web UI for more on runtime tuning.

Table of Contents

File Loading

1. Required Variables

The server will refuse to start if either of these is missing or empty.

Variable	Default	Required	Description
`DATABASE_URL`	*(none)*	**Yes**	PostgreSQL connection string.
`JWT_SECRET`	*(none)*	**Yes**	Secret key used to sign JWT tokens. Must be a strong, randomly generated value.

Examples:

```
DATABASE_URL="postgresql://patchmon_user:strongpassword@localhost:5432/patchmon_db
"
JWT_SECRET="$(openssl rand -hex 64)"
```

> *Keep `JWT_SECRET` stable across restarts. Changing it invalidates all active sessions and forces every user to log in again.*

2. Server Configuration

General HTTP server and network settings.

Variable	Default	Required	Description
`PORT`	`3000`	No	TCP port the server listens on.
`APP_ENV`	`production`	No	Runtime environment. Accepted values: `production` `NODE_ENV` is also read as a backward-compatibility a takes precedence when both are set.
`CORS_ORIGIN`	`http://localhost:3000`	No	Allowed CORS origin(s). Must match the exact URL yo PatchMon in your browser (protocol, hostname, and p trailing slash). To allow multiple origins, separate them and no spaces (e.g. `https://patchmon.example.com,https://patchmo`
`ENABLE_HSTS`	`false`	No	When `true`, the server adds an `HTTP Strict Trans` header to responses. Enable this only when PatchMon HTTPS.
`TRUST_PROXY`	`false`	No	When `true`, the server trusts `X-Forwarded-For` an from a reverse proxy (nginx, Caddy, etc.). Required fo detection and correct rate limiting when behind a prox

Production example:

```
PORT=3000
APP_ENV=production
CORS_ORIGIN=https://patchmon.example.com
ENABLE_HSTS=true
TRUST_PROXY=true
```

> *Set `CORS_ORIGIN` to the full URL your users type in their browser. A mismatch here is the most common cause of CORS errors after a fresh deployment. If PatchMon is accessed from multiple URLs (e.g. an external domain and an internal LAN address), list them comma-separated with no spaces:*
> *`CORS_ORIGIN=https://patchmon.example.com,https://patchmon.internal.lan`.*

---

3. Database Connection Pool

These variables control how the server manages its PostgreSQL connection pool. The defaults work well for most deployments; adjust them if you are running a large number of monitored hosts or see connection timeout errors.

All timeout values are in seconds unless otherwise noted.

Variable	Default	Required	Description
`PM_DB_CONN_MAX_ATTEMPTS`	`30`	No	How many times the server will retry connecting to the database on startup before giving up. Useful in containerised environments where the database may not be ready immediately.
`PM_DB_CONN_WAIT_INTERVAL`	`2`	No	Seconds to wait between each connection retry attempt on startup.
`DB_CONNECTION_LIMIT`	`30`	No	Maximum number of concurrent database connections in the pool.
`DB_POOL_TIMEOUT`	`20`	No	Seconds to wait for an available connection from the pool before returning a timeout error.
`DB_CONNECT_TIMEOUT`	`10`	No	Seconds to wait when establishing a new individual connection to PostgreSQL.
`DB_IDLE_TIMEOUT`	`300`	No	Seconds an idle connection is kept open before being closed and removed from the pool.
`DB_MAX_LIFETIME`	`1800`	No	Maximum lifetime in seconds of any connection in the pool, regardless of activity. Connections are recycled after this time to prevent stale connections.
`DB_TRANSACTION_MAX_WAIT`	`10000`	No	Milliseconds to wait for a transaction to acquire a database lock before giving up.
`DB_TRANSACTION_TIMEOUT`	`30000`	No	Milliseconds allowed for a standard database transaction to complete.
`DB_TRANSACTION_LONG_TIMEOUT`	`60000`	No	Milliseconds allowed for long-running operations (for example, bulk package imports or compliance scans). Increase this if those operations are timing out.

Sizing guidance:

Deployment size	`DB_CONNECTION_LIMIT`
Small (1–10 hosts)	`15`
Medium (10–50 hosts)	`30` (default)
Large (50+ hosts)	`50` or higher

If you see `connection pool exhausted` errors in the server logs, increase `DB_CONNECTION_LIMIT` in increments of 10 and monitor until the errors stop.

---

4. Authentication and Sessions

Settings for JWT tokens, browser sessions, account lockout, two-factor authentication, and user roles.

JWT and Tokens

Variable	Default	Required	Description
`JWT_SECRET`	*(none)*	**Yes**	See Required Variables.
`JWT_EXPIRES_IN`	`1h`	No	How long an access token is valid. Accepts duration strings: `30m`, `1h`, `2h`, `1d`. Shorter values are more secure but require more frequent token refreshes.
`AUTH_BROWSER_SESSION_COOKIES`	`false`	No	When set to `true`, the `token` and `refresh_token` cookies are issued without a `Max-Age` attribute, making them session cookies that are cleared when the browser is closed rather than persisting across browser restarts.

Account Lockout

Lockout is applied per user account after repeated failed login attempts.

Variable	Default	Required	Description
`MAX_LOGIN_ATTEMPTS`	`5`	No	Number of consecutive failed login attempts before the account is temporarily locked.
`LOCKOUT_DURATION_MINUTES`	`15`	No	How long (in minutes) an account stays locked after exceeding `MAX_LOGIN_ATTEMPTS`.

Session Inactivity

Variable	Default	Required	Description
`SESSION_INACTIVITY_TIMEOUT_MINUTES`	`30`	No	Minutes of inactivity before a user session is automatically invalidated. Each authenticated request resets the timer.

Two-Factor Authentication (TFA)

These settings apply only to users who have TFA enabled on their accounts.

Variable	Default	Required	Description
`MAX_TFA_ATTEMPTS`	`5`	No	Number of consecutive failed TFA code entries before the account is temporarily locked.
`TFA_LOCKOUT_DURATION_MINUTES`	`30`	No	How long (in minutes) a TFA lockout lasts.
`TFA_REMEMBER_ME_EXPIRES_IN`	`30d`	No	How long a "remember this device" TFA exemption is valid. Accepts duration strings such as `7d`, `30d`, `90d`.
`TFA_MAX_REMEMBER_SESSIONS`	`5`	No	Maximum number of remembered devices per user. When the limit is reached, the oldest remembered session is removed.

User Defaults

Variable	Default	Required	Description
`DEFAULT_USER_ROLE`	`user`	No	Role assigned to newly created users. Accepted values: `user`, `admin`, `readonly`. This does not affect existing users.

---

5. Redis Configuration

Redis is used for background job queues (asynq), bootstrap tokens, and TFA lockout state. A running Redis instance is required.

Variable	Default	Required	Description
`REDIS_HOST`	`localhost`	No	Hostname or IP address of the Redis server.
`REDIS_PORT`	`6379`	No	Port the Redis server listens on.
`REDIS_PASSWORD`	*(none)*	No	Redis authentication password. Strongly recommended in any non-local deployment.
`REDIS_USER`	*(none)*	No	Redis username for ACL-based authentication (Redis 6.0+). Leave empty to use password-only authentication.
`REDIS_DB`	`0`	No	Redis logical database number (0–15). Change this if you share a Redis instance with other applications.
`REDIS_TLS`	`false`	No	When `true`, the server connects to Redis over TLS.
`REDIS_TLS_VERIFY`	`true`	No	When set to `false`, the server skips Redis TLS certificate verification. Only use this in testing against self-signed certificates.
`REDIS_TLS_CA`	*(none)*	No	Path to a custom CA certificate file for verifying the Redis TLS connection. Only used when `REDIS_TLS=true`.
`REDIS_CONNECT_TIMEOUT_MS`	`60000`	No	Milliseconds to wait when establishing a new connection to Redis before timing out.
`REDIS_COMMAND_TIMEOUT_MS`	`60000`	No	Milliseconds to wait for a Redis command to complete before timing out.

Generating a secure Redis password:

```
openssl rand -hex 32
```

---

6. Rate Limiting

Rate limits protect the API from abuse. Limits are applied per IP address, split across three endpoint categories. All window values are in milliseconds.

Variable	Default	Required	Description
`RATE_LIMIT_WINDOW_MS`	`900000`	No	Time window for the general API rate limit (default: 15 minutes).
`RATE_LIMIT_MAX`	`5000`	No	Maximum requests per window for general API endpoints (dashboards, hosts, packages, settings).
`AUTH_RATE_LIMIT_WINDOW_MS`	`600000`	No	Time window for authentication endpoints (login, token refresh) (default: 10 minutes).
`AUTH_RATE_LIMIT_MAX`	`500`	No	Maximum requests per window for authentication endpoints.
`AGENT_RATE_LIMIT_WINDOW_MS`	`60000`	No	Time window for agent check-in and reporting endpoints (default: 1 minute).
`AGENT_RATE_LIMIT_MAX`	`1000`	No	Maximum requests per window for agent endpoints. Increase this if you have a large number of agents checking in frequently.
`PASSWORD_RATE_LIMIT_WINDOW_MS`	`900000`	No	Time window for password change and reset operations (default: 15 minutes).
`PASSWORD_RATE_LIMIT_MAX`	`5`	No	Maximum password change attempts per window. Kept deliberately low to limit brute-force attacks on password reset flows.

Quick reference: window conversions

Milliseconds	Human-readable
`60000`	1 minute
`600000`	10 minutes
`900000`	15 minutes

7. Password Policy

Rules applied when a user sets or changes a local account password. These do not apply to OIDC users, who authenticate against their identity provider.

Variable	Default	Required	Description
`PASSWORD_MIN_LENGTH`	`8`	No	Minimum number of characters required in a password.
`PASSWORD_REQUIRE_UPPERCASE`	`true`	No	Require at least one uppercase letter. Set to `false` to disable.
`PASSWORD_REQUIRE_LOWERCASE`	`true`	No	Require at least one lowercase letter. Set to `false` to disable.
`PASSWORD_REQUIRE_NUMBER`	`true`	No	Require at least one numeric digit. Set to `false` to disable.
`PASSWORD_REQUIRE_SPECIAL`	`true`	No	Require at least one special character (e.g. `!`, `@`, `#`). Set to `false` to disable.

> *All four complexity options default to `true`. To disable a rule you must explicitly set it to `false`: omitting the variable leaves the rule enabled.*

---

8. Logging and Profiling

Variable	Default	Required	Description
`ENABLE_LOGGING`	`false`	No	When `true`, enables structured application logging to stdout. Set to `true` in production to capture request and error logs.
`LOG_LEVEL`	`info`	No	Minimum log level to output. Accepted values: `debug`, `info`, `warn`, `error`. Must be one of these exact strings. The server will fail to start if an invalid value is provided.
`ENABLE_PPROF`	`false`	No	When `true`, exposes Go pprof profiling endpoints. For diagnostics only. Do not enable in production unless actively investigating a performance issue.
`MEMSTATS_INTERVAL_SEC`	`60`	No	How often (in seconds) the server logs Go runtime memory statistics when profiling is active. Only relevant when `ENABLE_PPROF=true`.

Log level guide:

Level	When to use
`debug`	Active troubleshooting: very verbose, includes internal operations
`info`	Normal production operation
`warn`	Quieter production operation; only non-critical issues and errors
`error`	Minimal output; critical errors only

9. OIDC / SSO

OpenID Connect configuration for Single Sign-On. When `OIDC_ENABLED=true`, the four marked variables below become required and the server will refuse to start without them.

Core Settings

Variable	Default	Required	Description
`OIDC_ENABLED`	`false`	No	Set to `true` to activate OIDC authentication.
`OIDC_ISSUER_URL`	*(none)*	If OIDC enabled	The issuer URL of your identity provider (e.g. `https://auth.example.com`). The server fetches the OIDC discovery document from this URL.
`OIDC_CLIENT_ID`	*(none)*	If OIDC enabled	The client ID registered in your identity provider.
`OIDC_CLIENT_SECRET`	*(none)*	If OIDC enabled	The client secret from your identity provider.
`OIDC_REDIRECT_URI`	*(none)*	If OIDC enabled	The callback URL registered in your identity provider. Must be: `https://your-patchmon-url/api/v1/auth/oidc/callback`
`OIDC_SCOPES`	`openid email profile groups`	No	Space-separated list of OAuth scopes to request. The `groups` scope is required for group-to-role mapping to work.
`OIDC_ENFORCE_HTTPS`	`true`	No	When `true` (the default), the server rejects OIDC configurations using a non-HTTPS issuer URL. Set to `false` only in a local development environment with a non-TLS identity provider.

User Provisioning

Variable	Default	Required	Description
`OIDC_AUTO_CREATE_USERS`	`false`	No	When `true`, a PatchMon account is automatically created the first time an OIDC user logs in. When `false`, an administrator must create the account first.
`OIDC_DEFAULT_ROLE`	`user`	No	Role assigned to automatically created OIDC users when no group mapping matches. Accepted values: `superadmin`, `admin`, `host_manager`, `user`, `readonly`.
`OIDC_DISABLE_LOCAL_AUTH`	`false`	No	When `true`, local username/password authentication is disabled. Only OIDC login is accepted. Useful when enforcing SSO organisation-wide.

Login Page

Variable	Default	Required	Description
`OIDC_BUTTON_TEXT`	`Login with SSO`	No	Text displayed on the SSO login button on the PatchMon login page.
`OIDC_POST_LOGOUT_URI`	Derived from `FRONTEND_URL`, then `CORS_ORIGIN`	No	URL the user is redirected to after logging out of the identity provider. Defaults to the login page of your PatchMon instance.

Session

Variable	Default	Required	Description
`OIDC_SESSION_TTL`	`600`	No	Lifetime in seconds of the temporary OIDC session state stored during the OAuth flow. Increase this only if users on very slow networks experience session-expired errors mid-login.

Group-to-Role Mapping

Map groups from your identity provider directly to PatchMon roles. Set `OIDC_SYNC_ROLES=true` to keep role assignments in sync with group membership on every login.

Variable	Default	Required	Description
`OIDC_SYNC_ROLES`	`false`	No	When `true`, the user's PatchMon role is updated on every login to match their current IdP group membership. When `false`, roles are managed locally in PatchMon and OIDC login does not change them.
`OIDC_ADMIN_GROUP`	*(none)*	No	Name of the IdP group whose members are granted the `admin` role.
`OIDC_SUPERADMIN_GROUP`	*(none)*	No	Name of the IdP group whose members are granted the `superadmin` role.
`OIDC_HOST_MANAGER_GROUP`	*(none)*	No	Name of the IdP group whose members are granted the `host_manager` role.
`OIDC_READONLY_GROUP`	*(none)*	No	Name of the IdP group whose members are granted the `readonly` role.
`OIDC_USER_GROUP`	*(none)*	No	Name of the IdP group whose members are granted the standard `user` role.

Example: Authentik

```
OIDC_ENABLED=true
OIDC_ISSUER_URL=https://authentik.example.com/application/o/patchmon/
OIDC_CLIENT_ID=patchmon
OIDC_CLIENT_SECRET=your-client-secret
OIDC_REDIRECT_URI=https://patchmon.example.com/api/v1/auth/oidc/callback
OIDC_SCOPES=openid email profile groups
OIDC_AUTO_CREATE_USERS=true
OIDC_DEFAULT_ROLE=user
OIDC_BUTTON_TEXT=Login with Authentik
OIDC_SYNC_ROLES=true
OIDC_ADMIN_GROUP=PatchMon Admins
OIDC_USER_GROUP=PatchMon Users
```

Example: Keycloak

```
OIDC_ENABLED=true
OIDC_ISSUER_URL=https://keycloak.example.com/realms/your-realm
OIDC_CLIENT_ID=patchmon
OIDC_CLIENT_SECRET=your-client-secret
OIDC_REDIRECT_URI=https://patchmon.example.com/api/v1/auth/oidc/callback
OIDC_SCOPES=openid email profile groups
OIDC_AUTO_CREATE_USERS=true
OIDC_DEFAULT_ROLE=user
OIDC_BUTTON_TEXT=Login with Keycloak
```

10. Compliance / SSG

Settings for SCAP Security Guide content used by the compliance scanning feature.

Variable	Default	Required	Description
`SSG_CONTENT_DIR`	`./ssg-content`	No	Path to the directory containing SCAP Security Guide content (`.xml` datastream files). The compliance scanner reads SCAP profiles from this directory. When running in Docker, mount your SSG content to this path.

11. RDP / Remote Access

Configuration for the Guacamole daemon (`guacd`) that powers in-browser RDP sessions.

Variable	Default	Required	Description
`GUACD_PATH`	*(none)*	No	Absolute path to the `guacd` binary. When empty, the server locates `guacd` using the system `PATH`. Set this if `guacd` is installed in a non-standard location.
`GUACD_ADDRESS`	`127.0.0.1:4822`	No	Host and port the server uses to connect to the running `guacd` process. Change this if `guacd` is running on a different host or non-default port.

12. Body Limits

Maximum sizes for request bodies accepted by the API. Increase these only if you encounter HTTP 413 errors caused by legitimate large payloads.

Accepted suffixes: `b`, `kb`, `mb`, `gb`. Examples: `10mb`, `512kb`.

Variable	Default	Required	Description
`JSON_BODY_LIMIT`	`5mb`	No	Maximum size of JSON request bodies for standard API endpoints (user management, settings, host actions, etc.).
`AGENT_UPDATE_BODY_LIMIT`	`2mb`	No	Maximum size of request bodies on agent check-in and package reporting endpoints. Increase this if agents managing a very large number of packages hit the limit.

13. Timezone

Variable	Default	Required	Description
`TZ`	`UTC`	No	IANA timezone name used for timestamps in server logs and scheduled operations. If `TZ` is not set, the server also checks `TIMEZONE` before falling back to `UTC`. All timestamps stored in the database remain in UTC regardless of this setting.

Common values:

```
TZ=UTC                    # Recommended for servers
TZ=Europe/London
TZ=Europe/Paris
TZ=America/New_York
TZ=America/Chicago
TZ=America/Los_Angeles
TZ=Asia/Tokyo
```

14. Encryption Keys

PatchMon encrypts sensitive values at rest: AI provider credentials, bootstrap enrolment tokens, OIDC client secrets, notification destination secrets. The encryption key is resolved from the first non-empty value in this list:

1. `AI_ENCRYPTION_KEY`
2. `SESSION_SECRET`
3. Derived from `DATABASE_URL` (fallback; **not recommended for production**)

If you rotate this value, every encrypted secret in the database becomes unreadable. Set it once at install time and treat it with the same care as `JWT_SECRET`.

Variable	Default	Required	Description
`AI_ENCRYPTION_KEY`	*(none)*	Recommended	32+ byte random secret used to encrypt AI provider keys, bootstrap tokens, OIDC client secrets, and notification destination credentials. Configure via `.env` only (not editable from the Settings UI).
`SESSION_SECRET`	*(none)*	No	Fallback encryption key used if `AI_ENCRYPTION_KEY` is not set. Exists for backward compatibility with early 1.x installs; prefer `AI_ENCRYPTION_KEY` for new deployments.

Generating a secure value:

```
openssl rand -hex 32
```

> *If neither `AI_ENCRYPTION_KEY` nor `SESSION_SECRET` is set, the server derives an encryption key from `DATABASE_URL`. This works but means your encryption key is only as strong as your database connection string. Any change to the DB host, port, or password rotates the encryption key and invalidates every encrypted value. Always set `AI_ENCRYPTION_KEY` explicitly in production.*

---

15. Agent Binary Overrides

Advanced deployments that want to replace the bundled agent binaries (for example, to ship a custom build or host binaries on a different volume) can override the directory the install script serves from. Normally you should leave these unset. The server embeds all supported agent binaries in the image at build time.

Variable	Default	Required	Description
`AGENT_BINARIES_DIR`	*(none)*	No	Absolute path to a directory containing replacement agent binaries (e.g. `patchmon-agent-linux-amd64`). Takes precedence over `AGENTS_DIR` when both are set.
`AGENTS_DIR`	*(none)*	No	Alternative name for `AGENT_BINARIES_DIR`. Kept for compatibility with installs that set this variable before 2.0.

When both are empty, the server serves binaries from the `static/` path embedded in its binary.

16. Telemetry

PatchMon can send anonymous usage heartbeats (version, host count, rough OS distribution) to the upstream metrics endpoint once a day. This is fully opt-in. See Metrics and telemetry for what is sent and the **Settings → Metrics** page to toggle it.

Variable	Default	Required	Description
`METRICS_API_URL`	*(none)*	No	Override the upstream metrics endpoint. Leave empty to send to the default PatchMon telemetry service. Set to an internal URL to collect telemetry privately.

17. File Loading

Variable	Default	Required	Description
`ENV_FILE`	`.env`	No	Path to the `.env` file the server reads at startup. If the file does not exist at the given path, startup continues silently and only actual process environment variables are used.
`FRONTEND_URL`	*(none)*	No	Optional alias used by OIDC when computing `OIDC_POST_LOGOUT_URI`. If set and `OIDC_POST_LOGOUT_URI` is not set, the post-logout redirect defaults to `<FRONTEND_URL>/login`. Otherwise the server falls back to `<CORS_ORIGIN>/login`. Most deployments can ignore this; set `CORS_ORIGIN` correctly and it's not needed.

Complete Minimal Configuration

The smallest valid `.env` for a production deployment:

```
# Required
DATABASE_URL="postgresql://patchmon_user:strongpassword@database:5432/patchmon_db"
JWT_SECRET="paste-output-of-openssl-rand-hex-64-here"

# Encryption (strongly recommended in production)
AI_ENCRYPTION_KEY="paste-output-of-openssl-rand-hex-32-here"

# Server
PORT=3000
APP_ENV=production
CORS_ORIGIN=https://patchmon.example.com
ENABLE_HSTS=true
TRUST_PROXY=true

# Redis
REDIS_HOST=redis
REDIS_PORT=6379
REDIS_PASSWORD="paste-output-of-openssl-rand-hex-32-here"

# Logging
ENABLE_LOGGING=true
LOG_LEVEL=info

# Timezone
TZ=UTC
```

Everything else defaults to a sensible production value and does not need to be set unless you want to change the behaviour described in this document.

> *The `setup-env.sh` script shipped with the Docker compose generates a valid `.env` with all three secrets (`JWT_SECRET`, `REDIS_PASSWORD`, and `POSTGRES_PASSWORD`) pre-populated. See Installing PatchMon Server on Docker. For day-to-day changes to rate limits, logging, password policy, timezone, and similar runtime-tunable values, prefer the Settings UI. See Settings in the web UI.*

Troubleshooting

Server fails to start with "DATABASE_URL is required" or "JWT_SECRET is required" These two variables have no default. Verify they are present in your `.env` file and that the file is being loaded (check the `ENV_FILE` variable if you use a custom path).

CORS errors in the browser `CORS_ORIGIN` must exactly match the URL in your browser's address bar, including the protocol (`http` vs `https`) and port. A common mistake is setting it to `https://patchmon.example.com` while accessing the site on `http://`. If you access PatchMon from multiple URLs, list all of them comma-separated with no spaces (e.g. `CORS_ORIGIN=https://patchmon.example.com,https://patchmon.internal.lan`).

Rate limit errors (HTTP 429) Increase the relevant `*_RATE_LIMIT_MAX` value for the endpoint category hitting the limit. For large agent fleets, `AGENT_RATE_LIMIT_MAX` is the most common one to raise.

Database connection pool exhausted Increase `DB_CONNECTION_LIMIT`. Check your PostgreSQL `max_connections` setting to ensure the total across all PatchMon instances does not exceed it.

OIDC login fails immediately after enabling Verify that `OIDC_ISSUER_URL`, `OIDC_CLIENT_ID`, `OIDC_CLIENT_SECRET`, and `OIDC_REDIRECT_URI` are all set. The server validates these on startup and will not start if any are missing when `OIDC_ENABLED=true`. Also confirm that `OIDC_REDIRECT_URI` is registered as an allowed callback URL in your identity provider.

Sessions lost after server restart Verify `JWT_SECRET` has not changed. Rotating this value invalidates all existing tokens.

PatchMon 2.0+ (Go server)

Chapter 6: Setting Up OIDC SSO

Overview

PatchMon supports OpenID Connect (OIDC) authentication, allowing users to log in via an external Identity Provider (IdP) instead of, or in addition to, local username/password credentials.

Supported Providers

Any OIDC-compliant provider works, including:

- Authentik
- Keycloak
- Okta
- Azure AD (Entra ID)
- Google Workspace

What You Get

- **SSO login** via a configurable button on the login page
- **Automatic user provisioning** on first login (no need to create accounts manually)
- **Group-based role mapping** so your IdP controls who is an admin, user, or readonly viewer
- **Optional:** disable local password login entirely and enforce SSO for all users

---

Prerequisites

- PatchMon already installed and running
- An OIDC-compatible Identity Provider with an OAuth2/OIDC application configured
- HTTPS in production (OIDC routes enforce HTTPS when `OIDC_ENFORCE_HTTPS=true`, which is the default)

Step 1 - Create an OIDC Application in Your IdP

Create a new OAuth2 / OIDC application in your Identity Provider with the following settings:

Setting	Value
Application type	Web application / Confidential client
Redirect URI	`https://patchmon.example.com/api/v1/auth/oidc/callback`
Scopes	`openid`, `email`, `profile`, `groups`
Grant type	Authorization Code
Token endpoint auth	Client Secret (Basic)

After creating the application, note the **Client ID** and **Client Secret** as you'll need both.

> ***Tip:*** *If you plan to use group-based role mapping, ensure your IdP includes the* `groups` *claim in the ID token. In Authentik, this is enabled by default. In Keycloak, you may need to add a "Group Membership" mapper to the client scope.*

Provider-Specific Notes

Authentik:

- Create an OAuth2/OIDC Provider, then create an Application linked to it
- Issuer URL format: `https://auth.example.com/application/o/patchmon/`
- Groups are included via the `groups` or `ak_groups` claim (both are supported)

Keycloak:

- Create a Client with Access Type `confidential`
- Issuer URL format: `https://keycloak.example.com/realms/your-realm`
- Add a "Group Membership" protocol mapper to include groups in the token

Okta / Azure AD:

- Create an OIDC Web Application

Ensure groups are included in the ID token claims

Step 2 - Configure PatchMon

Add the following environment variables to your `.env` file (for Docker deployments) or your server environment.

Required Variables

```
OIDC_ENABLED=true
OIDC_ISSUER_URL=https://auth.example.com/application/o/patchmon/
OIDC_CLIENT_ID=your-client-id
OIDC_CLIENT_SECRET=your-client-secret
OIDC_REDIRECT_URI=https://patchmon.example.com/api/v1/auth/oidc/callback
```

Variable	Description
`OIDC_ENABLED`	Set to `true` to enable OIDC
`OIDC_ISSUER_URL`	Your IdP's issuer / discovery URL
`OIDC_CLIENT_ID`	Client ID from your IdP application
`OIDC_CLIENT_SECRET`	Client secret from your IdP application
`OIDC_REDIRECT_URI`	Must match exactly what you configured in your IdP

Optional Variables

```
OIDC_SCOPES=openid email profile groups
OIDC_AUTO_CREATE_USERS=false
OIDC_DEFAULT_ROLE=user
OIDC_DISABLE_LOCAL_AUTH=false
OIDC_BUTTON_TEXT=Login with SSO
OIDC_SESSION_TTL=600
OIDC_POST_LOGOUT_URI=https://patchmon.example.com/login
OIDC_ENFORCE_HTTPS=true
OIDC_SYNC_ROLES=false
```

Variable	Default	Description
`OIDC_SCOPES`	`openid email profile groups`	Space-separated scopes to request. Include `groups` for role mapping
`OIDC_AUTO_CREATE_USERS`	`false`	When `true`, automatically creates a PatchMon account on first OIDC login. When `false`, the user must already exist in PatchMon (matched by email)
`OIDC_DEFAULT_ROLE`	`user`	Role assigned when a user doesn't match any group mapping
`OIDC_DISABLE_LOCAL_AUTH`	`false`	When `true`, hides the username/password fields and only shows the SSO button
`OIDC_BUTTON_TEXT`	`Login with SSO`	Label shown on the SSO login button
`OIDC_SESSION_TTL`	`600`	Seconds the OIDC login state is valid. If the user takes longer than this at the IdP, the session expires and they must try again
`OIDC_POST_LOGOUT_URI`	`<CORS_ORIGIN>/login`	Where to redirect after a logout. Defaults to the PatchMon login page
`OIDC_ENFORCE_HTTPS`	`true`	When `true`, enforces HTTPS on OIDC login and callback routes. Set to `false` only for local development
`OIDC_SYNC_ROLES`	`false`	When `true`, the user's role is updated on every login based on current group membership. When `false`, roles are managed locally in PatchMon and OIDC login does not change them

> ***Note on*** `APP_ENV` ***:*** *PatchMon reads* `APP_ENV` *to determine the runtime environment (e.g.* `production`*).* `NODE_ENV` *is accepted as a backward-compatibility alias but* `APP_ENV` *is preferred.*

Step 3 - Group-Based Role Mapping (Optional)

Map your IdP groups to PatchMon roles so that role assignments stay in sync with your directory. Group matching is **case-insensitive**.

Role Hierarchy

PatchMon checks group membership in this order (highest priority first):

PatchMon Role	Required IdP Group(s)	Description
Super Admin	Member of `OIDC_SUPERADMIN_GROUP`	Full access including managing other superadmins
Admin	Member of `OIDC_ADMIN_GROUP`	Full access
Host Manager	Member of `OIDC_HOST_MANAGER_GROUP`	Manage hosts and groups
User	Member of `OIDC_USER_GROUP`	Standard access with data export
Readonly	Member of `OIDC_READONLY_GROUP`	View-only access
Default	None of the above	Gets `OIDC_DEFAULT_ROLE` (defaults to `user`)

> ***Priority:*** *If a user is in multiple groups, the highest-priority role wins. The priority order from highest to lowest is: Super Admin > Admin > Host Manager > Readonly > User.*

Environment Variables

```
OIDC_ADMIN_GROUP=PatchMon Admins
OIDC_USER_GROUP=PatchMon Users
OIDC_SUPERADMIN_GROUP=PatchMon SuperAdmins
OIDC_HOST_MANAGER_GROUP=PatchMon Host Managers
OIDC_READONLY_GROUP=PatchMon Readonly
OIDC_SYNC_ROLES=true
```

Variable	Description
`OIDC_ADMIN_GROUP`	IdP group name that maps to Admin role
`OIDC_USER_GROUP`	IdP group name that maps to User role
`OIDC_SUPERADMIN_GROUP`	IdP group name that maps to Super Admin
`OIDC_HOST_MANAGER_GROUP`	IdP group name that maps to Host Manager role
`OIDC_READONLY_GROUP`	IdP group name that maps to Readonly role
`OIDC_SYNC_ROLES`	When `true`, the user's role is updated on every login based on current group membership. When `false` (default), roles are managed locally in PatchMon and OIDC login does not change them

You only need to define the groups you intend to use. Any variables left unset are simply ignored.

Step 4 - Restart PatchMon

After updating your `.env` file, restart the server so it discovers your OIDC provider on startup:

```
# Docker
docker compose restart patchmon-server

# Or if rebuilding
docker compose up -d --force-recreate patchmon-server

# Native systemd installation
sudo systemctl restart <your-domain>
```

Check the logs to confirm OIDC initialised:

```
# Docker
docker compose logs patchmon-server | grep -i oidc

# Native systemd
journalctl -u <your-domain> | grep -i oidc
```

You should see:

```
Discovering OIDC configuration from: https://auth.example.com/...
OIDC Issuer discovered: https://auth.example.com/...
OIDC client initialized successfully
```

If you see `OIDC is enabled but missing required configuration`, double-check your environment variables.

Step 5 - Test the Login

1. Open PatchMon in your browser
2. You should see a **"Login with SSO"** button (or your custom `OIDC_BUTTON_TEXT`)
3. Click it and you'll be redirected to your IdP
4. Authenticate with your IdP credentials
5. You'll be redirected back to PatchMon and logged in

If `OIDC_AUTO_CREATE_USERS` is `true`, a PatchMon account is created automatically using your email address. The username is derived from the email prefix (e.g. `john.doe@example.com` becomes `john.doe`).

First-Time Setup (No Users Exist Yet)

When PatchMon has no users in the database, it displays a setup wizard. You have two options:

Option A - Use the Setup Wizard (Recommended)

Complete the setup wizard to create your first admin account. This account is created as a **Super Admin** with full access. You can then enable OIDC from Settings afterwards.

Option B - Log In via OIDC Directly

If you've pre-configured OIDC via environment variables before first boot:

- Set `OIDC_AUTO_CREATE_USERS=true`
- The setup wizard is automatically bypassed when OIDC with auto-create is enabled
- The **first user** to log in via OIDC is automatically promoted to **Super Admin**, regardless of group mapping, to ensure the system always has an admin
- Subsequent OIDC users get roles based on group mapping or the default role as normal

> ***Note:*** *You do not need to configure group mapping for the first user. The auto-promotion happens because PatchMon detects no admin exists yet.*

What Syncs from Your IdP

On every OIDC login, PatchMon automatically syncs the following from your Identity Provider:

- **Avatar / profile picture:** synced if the `picture` claim is present
- **First name and last name:** from `given_name` and `family_name` claims
- **Email:** used for matching and account linking

The following is **only synced when `OIDC_SYNC_ROLES=true`:**

- **Role:** based on group membership. When sync is off, roles are managed locally in PatchMon and OIDC login does not change them. You can use OIDC for authentication while still managing roles manually.

Account Linking

If a local PatchMon user already exists with the same email as the OIDC user, PatchMon will automatically link the accounts, but only if the email is marked as **verified** by the IdP. This prevents account takeover via unverified emails.

Disabling Local Authentication

To enforce SSO for all users, set:

```
OIDC_DISABLE_LOCAL_AUTH=true
```

This hides the username/password fields on the login page and only shows the SSO button. Local authentication is only actually disabled if OIDC is also enabled and successfully initialised. This safety check prevents you from being locked out if OIDC is misconfigured.

> ***Important:*** *Ensure at least one OIDC user has admin access before enabling this, or you may lose the ability to manage PatchMon.*

Complete Example Configuration

Authentik

```
# .env
APP_ENV=production
OIDC_ENABLED=true
OIDC_ISSUER_URL=https://authentik.example.com/application/o/patchmon/
OIDC_CLIENT_ID=patchmon
OIDC_CLIENT_SECRET=your-client-secret-here
OIDC_REDIRECT_URI=https://patchmon.example.com/api/v1/auth/oidc/callback
OIDC_SCOPES=openid email profile groups
OIDC_AUTO_CREATE_USERS=true
OIDC_DEFAULT_ROLE=user
OIDC_BUTTON_TEXT=Login with Authentik
OIDC_ADMIN_GROUP=PatchMon Admins
OIDC_USER_GROUP=PatchMon Users
OIDC_SYNC_ROLES=true
```

Keycloak

```
# .env
APP_ENV=production
OIDC_ENABLED=true
OIDC_ISSUER_URL=https://keycloak.example.com/realms/your-realm
OIDC_CLIENT_ID=patchmon
OIDC_CLIENT_SECRET=your-client-secret-here
OIDC_REDIRECT_URI=https://patchmon.example.com/api/v1/auth/oidc/callback
OIDC_SCOPES=openid email profile groups
OIDC_AUTO_CREATE_USERS=true
OIDC_DEFAULT_ROLE=user
OIDC_BUTTON_TEXT=Login with Keycloak
OIDC_ADMIN_GROUP=PatchMon Admins
OIDC_USER_GROUP=PatchMon Users
OIDC_SYNC_ROLES=true
```

Troubleshooting

OIDC Not Initialising

Logs show: `OIDC is enabled but missing required configuration`

All four required variables must be set: `OIDC_ISSUER_URL`, `OIDC_CLIENT_ID`, `OIDC_CLIENT_SECRET`, `OIDC_REDIRECT_URI`. Check for typos or empty values.

SSO Button Not Appearing

The button only appears if OIDC is both enabled (`OIDC_ENABLED=true`) **and** successfully initialised. Check server logs for OIDC errors. Common causes:

- PatchMon cannot reach the IdP (DNS / firewall issue)
- Issuer URL is incorrect
- IdP's `.well-known/openid-configuration` endpoint is not accessible

"Authentication Failed" After Redirect

- Verify the **Redirect URI** in your IdP matches `OIDC_REDIRECT_URI` exactly (including trailing slashes)
- Ensure cookies are not being blocked (OIDC uses httpOnly cookies for session state)
- Check that your IdP supports PKCE (PatchMon uses S256 code challenge)

"Session Expired" Error

The OIDC login state has a configurable window (default 600 seconds via `OIDC_SESSION_TTL`). If the user takes longer than this at the IdP, the session expires. Simply try logging in again, or increase `OIDC_SESSION_TTL` if this is happening frequently.

User Gets Wrong Role

- Check that the `groups` scope is included in `OIDC_SCOPES`
- Verify your IdP is including groups in the ID token (not just the access token)
- Check server logs as they show which groups were received: `OIDC groups found: [...]`
- If logs show `No groups found in OIDC token`, configure your IdP to include the groups claim
- Group matching is case-insensitive, so `patchmon admins` matches `PatchMon Admins`

OIDC Banners / Restrictions Appearing When They Shouldn't

If you see "OIDC Authentication Enabled" banners on the Users or Roles settings pages, or the "Add User" / "Add Role" buttons are missing, `OIDC_SYNC_ROLES` is enabled. These restrictions only apply when role sync is active. If you want to use OIDC for login but manage roles locally, set `OIDC_SYNC_ROLES=false` (or leave it unset; it defaults to `false`).

"User Not Found" Error

`OIDC_AUTO_CREATE_USERS` is `false` (the default) and no matching PatchMon account exists. Either set `OIDC_AUTO_CREATE_USERS=true` or create the user account manually in PatchMon first (the email must match).

Debug Logging

For detailed OIDC troubleshooting, enable debug logging:

```
LOG_LEVEL=debug
```

Then check the server logs:

```
# Docker
docker compose logs -f patchmon-server | grep -i oidc

# Native systemd
journalctl -u <your-domain> -f | grep -i oidc
```

Security Notes

- **HTTPS is enforced** for OIDC login and callback routes when `OIDC_ENFORCE_HTTPS=true` (the default). Set `APP_ENV` to `production` in your environment. `NODE_ENV` is also accepted as a backward-compatibility alias
- **PKCE (S256)** is used for all authorization code exchanges
- **Tokens are stored in httpOnly cookies**, not localStorage, to prevent XSS attacks

Client secrets should never be committed to version control

Account linking only occurs when the IdP reports the email as verified

Role sync can be disabled (`OIDC_SYNC_ROLES=false` , which is the default) if you prefer to manage roles manually in PatchMon after first login

---

Chapter 7: Setting Up Azure Entra ID SSO

This is a step-by-step guide for configuring **Microsoft Azure Entra ID** (formerly Azure Active Directory) as the Single Sign-On provider for PatchMon using the **Settings UI**. No `.env` editing is required.

What You'll End Up With

Users sign in to PatchMon with their Microsoft work account.

PatchMon accounts are created automatically on first login.

PatchMon roles (Super Admin / Admin / Host Manager / User / Readonly) are driven by Entra ID **security groups**.

Optionally, local username/password login is disabled, so SSO is the only way in.

Everything is configured through **Settings → OIDC / SSO** in the PatchMon web interface.

Before You Begin

You'll need:

Item	Notes
A running PatchMon instance	Reachable at a fixed URL, e.g. `https://patchmon.example.com`
HTTPS on your PatchMon URL	Entra ID **will not** accept plain `http://` redirect URIs (except `http://localhost`)
An existing admin account in PatchMon	So you can sign in and open Settings. If you don't have one, complete the normal setup wizard first
Access to the Microsoft Entra admin center	`https://entra.microsoft.com` . You need **Application Administrator** or **Global Administrator** role on the tenant

Open two browser tabs side-by-side:

Tab 1: PatchMon → sign in as admin → **Settings → OIDC / SSO**

Tab 2: https://entra.microsoft.com (https://entra.microsoft.com)

You will collect **six values** in Tab 2 and paste them into Tab 1:

Tenant ID
Application (client) ID
Client secret (the **Value**, not the Secret ID)
Admin group Object ID
User group Object ID
(Optional) any additional role group Object IDs

Part A: Configure Entra ID (Tab 2)

Step 1: Get the Callback URL from PatchMon First

Before you start in Entra, grab the callback URL PatchMon will use. You'll paste it into Entra.

In Tab 1, go to **Settings → OIDC / SSO**.
Scroll down to the **OAuth2 Configuration** section.
Look at the **Callback URL** field. It will say something like:

```
https://patchmon.example.com/api/v1/auth/oidc/callback
```

Copy it. You'll need this in the next step.

> ***Note:*** *This field is read-only and is derived from the PatchMon server URL setting. If it looks wrong (e.g.* `http://localhost:3000` *when you're running in production), fix your* ***Server URL*** *in Settings → General first.*

Step 2: Register an Application in Entra ID

In Tab 2, open **Identity → Applications → App registrations**.
Click **+ New registration**.
Fill in the form:

Name: `PatchMon` (purely cosmetic, shown on the consent screen)

Supported account types: choose **Accounts in this organizational directory only (Single tenant)** for most deployments. Only pick multi-tenant if you explicitly want users from other Entra tenants to sign in.

Redirect URI:

Platform: **Web**

URL: paste the callback URL you copied in Step 1

Click **Register**.

You'll land on the app's **Overview** page. Copy these two values into a scratch note:

Application (client) ID

Directory (tenant) ID

Step 3: Create a Client Secret

In the left menu, open **Certificates & secrets**.

Under **Client secrets**, click **+ New client secret**.

Description: `PatchMon`. Expiry: pick a duration that fits your rotation policy (up to 24 months).

Click **Add**.

Copy the `Value` column immediately. This is the only time Entra will show it.

> ***Do not*** *copy the `Secret ID`. That is a metadata GUID, not the secret. You want the `Value` column.*

Save this value in your scratch note as **Client Secret**.

Step 4: Configure Token Claims (Add Groups)

PatchMon maps Entra ID groups to PatchMon roles, so Entra must include group information in the ID token.

In the left menu, open **Token configuration**.

Click **+ Add groups claim**.

Tick **Security groups**. Leave the other checkboxes unticked unless you specifically use Directory roles or Distribution lists.

Expand each of the three sections (**ID**, **Access**, **SAML**) and make sure **Group ID** is selected. This is the default. **Do not change it to sAMAccountName** for cloud-only Entra groups (sAMAccountName only works for groups synced from on-prem AD).

Click **Add**.

> ***What PatchMon receives:*** *With this configuration, Entra ID sends groups as an array of GUIDs (the group Object IDs) in the `groups` claim of the ID token. You will paste those GUIDs (not group names) into PatchMon's Role Mapping table.*

Optional but recommended: add standard user claims

Entra doesn't always include every OIDC-standard claim by default.

Still on **Token configuration**, click **+ Add optional claim**.

Token type: **ID**.

Tick `email`, `family_name`, `given_name`, `preferred_username`.

Click **Add**. If prompted to enable the Microsoft Graph `email` permission, accept.

Step 5: API Permissions

Open **API permissions** in the left menu.

You should already see `User.Read` listed under **Microsoft Graph**. That's enough. If it's missing, click **+ Add a permission → Microsoft Graph → Delegated permissions** and add `User.Read`, `openid`, `profile`, `email`.

Click **Grant admin consent for** at the top and confirm. Without admin consent, users will be prompted to consent individually on first login.

Step 6: Create Security Groups for Role Mapping

Decide which PatchMon roles you'll use. At minimum you probably want **Admin** and **User**. You can add more later.

For **each** role:

In Entra, go to **Identity → Groups → All groups**.

Click **+ New group**.

Fill in:

Group type: `Security`

Group name: e.g. `PatchMon Admins` (the name is for humans; PatchMon matches on Object ID)

Membership type: `Assigned` (simplest)

Add the users who should hold that role as **Members**.

Click **Create**.

After creation, open the group and **copy its Object ID** (a GUID like `11111111-2222-3333-4444-555555555555`) into your scratch note.

Repeat for each role you want to use.

Mapping table

PatchMon role	Entra group (example)	Where you'll paste the Object ID
Super Admin	`PatchMon SuperAdmins`	Role Mapping table → `superadmin` row
Admin	`PatchMon Admins`	Role Mapping table → `admin` row
Host Manager	`PatchMon Host Managers`	Role Mapping table → `host_manager` row
User	`PatchMon Users`	Role Mapping table → `user` row
Readonly	`PatchMon Readonly`	Role Mapping table → `readonly` row

> *You only need to fill in the rows you use. Empty rows are ignored. Users who match none of the groups get the **Default (fallback)** role.*

Part B: Configure PatchMon (Tab 1)

Go back to Tab 1: **Settings → OIDC / SSO**.

Step 7: Fill in the OAuth2 Configuration Section

Scroll to the **OAuth2 Configuration** panel and fill in the fields using the values from your scratch note:

Field in PatchMon	What to put in it
Issuer URL	`https://login.microsoftonline.com/<TENANT_ID>/v2.0`. Replace `<TENANT_ID>` with the Directory (tenant) ID from Step 2. The `/v2.0` suffix is required.
Client ID	The Application (client) ID from Step 2
Client Secret	Paste the client secret Value from Step 3, then click the **Save** button next to the field. The badge will change from "Not set" to "Set"
Callback URL	Read-only, already populated. This is the URL you registered in Entra in Step 2
Redirect URI (optional override)	Leave empty. Only use this if your PatchMon is behind a reverse proxy that presents a different public URL
Scopes	Change the default `openid email profile groups` to `openid email profile User.Read`: remove the trailing `groups` and add `User.Read`. Entra rejects `groups` as an unknown scope. `User.Read` is required if you want PatchMon to fetch the user's Entra profile photo
Button Text	`Sign in with Microsoft` (or anything you like)

Click **Apply** at the bottom of the panel. You should see a toast saying **"OIDC settings saved"**.

> ***Why no `groups` scope for Entra?*** *Other IdPs (Authentik, Keycloak) use a `groups` scope to request group claims. Entra does not. It uses the app's **Token configuration** instead (which you configured in Step 4). Including `groups` in the Scopes field will cause Entra to reject the authorisation request with an "invalid scope" error.*
>
> ***Why add `User.Read`?*** *PatchMon uses `User.Read` to call Microsoft Graph and fetch the signed-in user's profile photo. Without it, SSO still works, but Entra profile pictures cannot be imported.*

Step 8: Configure the Toggles

At the top of the OIDC / SSO page there's a **Configuration** panel with five toggles. Recommended settings for Entra ID:

Toggle	Recommended	Why
Enable OIDC / SSO	**Leave OFF for now.** You'll turn it on in Step 10 after everything else is set	Flipping it on too early will expose a broken SSO button on the login page
Enforce HTTPS	**ON**	Entra will not work over plain HTTP anyway
Sync roles from IdP	**ON**	Required if you want Entra security groups to drive PatchMon roles
Disable local auth	**OFF** (for now)	Leave this off until you've confirmed SSO works. You can enable it later
Auto-create users	**ON**	Creates PatchMon accounts automatically on first login so you don't have to pre-provision users

No Save button is needed for the toggles at the top (except **Enable OIDC / SSO**, which saves immediately). The other four are applied when you click **Apply** in the OAuth2 Configuration panel.

Step 9: Fill in the Role Mapping Table

Scroll to **Role Mapping** and click the header to expand it.

You'll see a table with a **Default (fallback)** row and one row per PatchMon role.

For each role you created an Entra group for, paste the group's **Object ID** (from Step 6) into the **OIDC Mapped Role (IdP Group Name)** column.

PatchMon Role	Paste here
Default (fallback)	Leave as `user`, or change to `readonly` if you want unmatched users to have no write access
superadmin	Entra Object ID of `PatchMon SuperAdmins` (or leave blank if you don't want anyone promoted to superadmin via SSO)
admin	Entra Object ID of `PatchMon Admins`
host manager	Entra Object ID of `PatchMon Host Managers`
user	Entra Object ID of `PatchMon Users`
readonly	Entra Object ID of `PatchMon Readonly`

Scroll back up to the **OAuth2 Configuration** panel and click **Apply** to save the role mapping. (The role mapping fields are saved together with the OAuth2 fields by the Apply

button.)

> ***Important:*** *The label reads "IdP Group Name" but for Entra ID you must paste the group's* ***Object ID (GUID)****, not the display name. Entra sends GUIDs in the token, not names.*

> ***Amber warning:*** *If Sync Roles is on but the Superadmin row is empty, you'll see an amber warning. That is expected: it means no one will be promoted to superadmin via SSO. Existing local superadmins will keep their role. If that's what you want, ignore the warning.*

Step 10: Turn On OIDC and Test

At the top of the page, flip **Enable OIDC / SSO** to **ON**. It saves immediately.

Open PatchMon in a **private/incognito browser window** (so you're not using your existing session).

You should see a **Sign in with Microsoft** button on the login page (or whatever text you set).

Click it. You'll be redirected to `login.microsoftonline.com`.

Sign in with an Entra account that's a member of one of your PatchMon groups.

You'll be redirected back and logged in.

First-login behaviour:

A PatchMon account is created automatically. The username is derived from the email prefix (e.g. `alice@contoso.com` → `alice`).

The role is determined by group membership; if no group matches, the **Default (fallback)** role is used.

If **no admin exists yet in PatchMon**, the very first OIDC user is automatically promoted to **Super Admin** regardless of groups, so you cannot lock yourself out.

Optional: Enforce SSO Only (Disable Password Login)

Once you've confirmed at least one OIDC user has Admin or Super Admin:

Go back to **Settings → OIDC / SSO**.

Turn **Disable local auth** to **ON**.

Click **Apply** at the bottom of the OAuth2 Configuration panel.

The login page will now only show the **Sign in with Microsoft** button. Local username/password fields are hidden.

> ***Safety:*** *PatchMon only enforces this flag if OIDC is **also** enabled and successfully initialised. If OIDC breaks for any reason, local login is automatically re-enabled so you're not locked out.*

Troubleshooting

"OIDC is configured via .env" amber banner at the top

You'll see this if OIDC environment variables were set in `.env` before the UI was used. Click **Load from .env** to import those values into the database, then remove the `OIDC_*` lines from `.env` and restart the server. From then on, everything is managed from the UI.

The "Sign in with Microsoft" button doesn't appear on the login page

The button only shows when OIDC is both **enabled** and **successfully initialised** at runtime. Most common causes:

- **Issuer URL is wrong:** it must end in `/v2.0`. Double-check for typos in the tenant GUID.
- **Client Secret is empty or wrong:** the label will say "Not set". Re-enter it and click **Save** next to the secret field.
- **PatchMon cannot reach `login.microsoftonline.com`:** an egress firewall or proxy is blocking it.

Check the server logs; search for `oidc`:

```
# Docker
docker compose logs patchmon-server | grep -i oidc

# Native systemd
journalctl -u <your-service-name> | grep -i oidc
```

`AADSTS50011: Reply URL does not match`

The redirect URI in Entra does not match the callback URL PatchMon is sending. Go to the Entra app's **Authentication** page and verify:

- Protocol is `https://`
- Host and port exactly match PatchMon's public URL
- Path is `/api/v1/auth/oidc/callback` with **no** trailing slash
- There are no hidden whitespace characters (paste into a plain editor to check)

If you're behind a reverse proxy and PatchMon is generating the wrong callback URL, fix the **Server URL** in **Settings → General** first. Do not use the "Redirect URI (optional override)" field unless you really know the proxy is presenting a different public URL.

`AADSTS70011: The provided value for scope ... is not valid`

Your Scopes field includes `groups`. Entra rejects unknown scopes. Change the Scopes field to:

```
openid email profile User.Read
```

Click **Apply**.

`AADSTS700016: Application with identifier ... was not found`

The **Client ID** field doesn't match the Application (client) ID in Entra. Copy it again from the app's **Overview** page and click **Apply**.

`AADSTS7000215: Invalid client secret provided`

The secret is wrong, was rotated, or has expired. Create a new one in Entra (**Certificates & secrets**), paste the new Value into the Client Secret field, and click **Save** next to the field.

Logged in but got the wrong role (or default role)

Make sure **Sync roles from IdP** toggle is ON.

Confirm you pasted the Entra group **Object ID (GUID)**, not the display name, into the Role Mapping table.

Check the server logs. PatchMon logs which groups it received:

```
docker compose logs patchmon-server | grep -i "oidc groups"
```

If logs show `oidc no groups in token`, revisit Step 4 and make sure the groups claim was added under Token configuration with **Security groups** → **Group ID**.

Logged in but no profile photo appears

Make sure the **Scopes** field includes `User.Read`.

Confirm the Entra app has **Microsoft Graph → Delegated permission → User.Read** and that **admin consent** was granted.

Check whether the user actually has a profile photo set in Microsoft 365 / Entra.

Sign out and sign back in after changing scopes or permissions so PatchMon gets a fresh access token.

"Too many groups": user belongs to more than 200 groups

If a user is a member of 200+ groups in Entra, the token switches to a `_claim_names` overage indicator and omits the `groups` array. PatchMon does not currently follow the overage pointer.

Workaround: In Entra's **Token configuration → Edit groups claim**, select **Groups assigned to the application**. This limits the claim to groups explicitly assigned to the PatchMon app, which almost always keeps the total well under 200.

"Session Expired" after clicking the SSO button

The state cookie has a 10-minute TTL by default. If users take too long on the Microsoft login page (MFA, password reset), it expires. They just need to click the SSO button again and complete the login faster. If this happens often, the TTL is configurable via `OIDC_SESSION_TTL` in `.env` (this one is not yet in the UI).

Quick Reference: Where Each Value Comes From

PatchMon UI field	Where to find it in Entra
Issuer URL	`https://login.microsoftonline.com/<Directory (tenant) ID>/v2.0`. Tenant ID is on the Entra app's **Overview** page
Client ID	Entra app **Overview** → **Application (client) ID**
Client Secret	Entra app → **Certificates & secrets** → client secret **Value** (shown once, at creation time)
Callback URL	Already filled in by PatchMon. Copy it **to** Entra, not from it
Scopes	`openid email profile User.Read` (no `groups`)
Role Mapping → each row	Entra → **Groups → All groups → → Overview → Object ID**

Chapter 8: Installing the PatchMon Agent

Overview

The PatchMon agent is enrolled through the web UI's **Add Host** wizard. The wizard provisions a host record in the database, issues a one-time bootstrap token, and builds a ready-to-paste one-liner. Running that command on the target host downloads the platform-specific agent binary over an authenticated HTTPS channel, writes config and credentials, registers a system service, and opens a persistent WebSocket connection back to the server.

This page covers the UI-driven install flow end to end. For everything that happens after the first check-in (CLI commands, service management, logs, updates), see Managing the PatchMon Agent. For bulk enrolment on container hosts, see Proxmox LXC Auto-Enrollment Guide.

How it Works

The web UI calls `POST /api/v1/hosts` (admin-authenticated) to create a host row and returns a plaintext `api_id` / `api_key` pair. This is the only time the plaintext key is ever exposed.

The wizard builds an install URL (`GET /api/v1/hosts/install?os=<linux|freebsd|windows>`) and pre-shares the credentials via `X-API-ID` / `X-API-KEY` headers.

The server responds with an OS-specific install script. A short-lived **bootstrap token** (5-minute TTL, single-use) is embedded at the top of the script so the host can exchange it for its permanent `api_id` / `api_key` via `POST /api/v1/hosts/bootstrap/exchange`.

The installer auto-detects architecture (via `uname -m` or `PROCESSOR_ARCHITECTURE`), downloads the matching agent binary from `GET /api/v1/hosts/agent/download`, writes `/etc/patchmon/config.yml` and `/etc/patchmon/credentials.yml` (or the Windows equivalents), and starts the service.

On first `serve`, the agent opens a WebSocket to `/api/v1/agents/ws` and sends an initial report. The wizard polls `/api/v1/ws/status/{apiId}` every 2 seconds and moves through four states: **Waiting for connection → Connected → Receiving initial report → Done**. It then redirects you to the host detail page.

Prerequisites

Before you enrol a host, make sure:

You can log into PatchMon as a user with the `can_manage_hosts` permission (admin, superadmin, or a custom role with that permission).

The target host can reach the PatchMon server over HTTPS (TCP/443 or whatever port your reverse proxy exposes).

The target host's clock is correct. The installer checks this and will warn (or abort interactively) if the system time looks wrong. TLS and signed tokens will both fail silently if the clock drifts more than a few minutes.

On Linux / FreeBSD you have `root` access (or `sudo`). On Windows you have an elevated PowerShell (Run as Administrator).

Step-by-Step Walkthrough

Step 1: Open the Add Host Wizard

Navigate to **Hosts → Add Host** in the web UI. A four-step wizard appears:

Step	Label	What you do
1	Choose OS	Pick Linux, FreeBSD, or Windows
2	Host details	Friendly name, host groups, optional integrations
3	Copy command	Copy the generated install one-liner
4	Connection	Wait for the agent to check in

Step 2: Choose the Operating System

Pick the OS of the target host. This determines which installer the server serves and which binary is downloaded.

OS	Installer	Binary format
Linux	POSIX shell script (`patchmon_install.sh`)	`patchmon-agent-linux-<arch>`
FreeBSD	POSIX shell script (same as Linux, with `os=freebsd` query parameter)	`patchmon-agent-freebsd-<arch>`
Windows	PowerShell script (`patchmon_install_windows.ps1`)	`patchmon-agent-windows-<arch>.exe`

The **architecture is auto-detected at install time**: you do not pick it in the UI. The installer maps `uname -m` (or `PROCESSOR_ARCHITECTURE` on Windows) onto one of `amd64` , `arm64` , `arm` , or `386` and downloads the matching binary.

> ***Windows 32-bit (x86) is not supported.*** *The installer aborts with a clear error. All Microsoft-supported Windows versions as of 2026 are 64-bit only.*

Step 3: Host Details

Fill in the form:

Field	Required	Notes
Friendly Name	Yes	Free-form label shown in the UI (e.g. `web-01.prod`). Does not have to match the real hostname; the real hostname is learned from the agent's first report.
Host Groups	No	Tick one or more groups to pre-tag the host. Groups can be changed later.
Docker integration	No	Enables Docker container/volume/image reporting. Can be toggled later from the host detail page.
Compliance integration	No	Enables the OpenSCAP compliance scanner. Can be toggled later.

Click **Next**. The UI calls `POST /api/v1/hosts` to create the host record and receives a plaintext `api_id` and `api_key` back. These are only rendered into the copy-paste command on the next step. They are **never stored or shown again** in the UI. If you lose the command before running it, regenerate credentials from the host detail page (see Managing the PatchMon Agent).

Step 4: Copy the Install Command

The wizard now shows a one-liner tailored to the OS you picked. Examples:

Linux one-liner:

```
curl -s "https://patchmon.example.com/api/v1/hosts/install" \
  -H "X-API-ID: patchmon_a1b2c3d4" \
  -H "X-API-KEY: <64-char-key>" | sudo sh
```

FreeBSD one-liner (note: no `sudo`; FreeBSD installs run as root directly; use `su -` first if you are not root):

```
curl -s "https://patchmon.example.com/api/v1/hosts/install?os=freebsd" \
  -H "X-API-ID: patchmon_a1b2c3d4" \
  -H "X-API-KEY: <64-char-key>" | sh
```

Windows one-liner (elevated PowerShell, single line):

```
$r = Invoke-WebRequest -Uri "https://patchmon.example.com/api/v1/hosts/install?
os=windows" -Headers @{"X-API-ID"="patchmon_a1b2c3d4"; "X-API-KEY"="<64-char-
key>"} -UseBasicParsing; $r.Content | Set-Content "$env:TEMP\patchmon-install.ps1"
-Encoding UTF8; & "$env:TEMP\patchmon-install.ps1"
```

Click **Copy command**. The wizard advances to **Step 5: Connection** automatically.

Windows Options

On the Windows step, two optional tick-boxes are shown:

Option	When to use
Self-signed certificate (SSL bypass)	Your PatchMon server uses a private/internal CA that Windows does not trust. The emitted command sets `[Net.ServicePointManager]::ServerCertificateValidationCallback = { $true }` before calling the server.
Use curl instead of Invoke-WebRequest	Some hardened / older Windows hosts fail with `Invoke-WebRequest` due to TLS 1.0/1.1 negotiation or chunked-transfer issues. Ticking this box uses `curl.exe` (shipped with Windows 10 1803+ and Server 2019+) instead.

If your server is using a commercially signed TLS certificate (Let's Encrypt, commercial CA), leave both unticked.

The `--force` Flag (Linux only)

The Linux installer supports a `--force` flag that bypasses broken `apt` packages during dependency installation. The wizard does not surface this by default. If `apt-get update` or `apt-get install curl` fails on the target host, re-run the install command with `--force` appended after `sh`:

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

Step 5: Run the Command on the Target Host

Paste the command into a terminal on the target host.

Linux / FreeBSD: Run as `root` or with `sudo`. The installer refuses to run otherwise.

Windows: Right-click PowerShell → **Run as Administrator**, then paste.

The installer will:

- Verify the system date/time (interactive confirmation when run on a TTY; silent continue when piped).
- Detect the package manager (`apt`, `dnf`, `yum`, `zypper`, `pacman`, `apk`, or `pkg`) and install `curl` if missing.
- Exchange the bootstrap token for the real `api_id` / `api_key` via `POST /api/v1/hosts/bootstrap/exchange`.
- Create `/etc/patchmon/` (Linux/FreeBSD) or `C:\ProgramData\PatchMon\` (Windows) with `config.yml` and `credentials.yml`, both with `0600` / Administrator-only permissions.

Download the matching agent binary from `GET /api/v1/hosts/agent/download?arch=<arch>&os=<os>`.

Run `patchmon-agent ping` to confirm credentials work.

Register the service:

systemd on most Linux distros: `/etc/systemd/system/patchmon-agent.service`

OpenRC on Alpine: `/etc/init.d/patchmon-agent`

rc.d on FreeBSD: `/usr/local/etc/rc.d/patchmon_agent`

Crontab fallback if no init system is detected

Windows Service Control Manager on Windows: service name `PatchMonAgent`, startup type Automatic

Start the service, which opens the WebSocket and sends the initial system report.

On a clean host with a working network, the whole process takes 10–30 seconds.

Step 6: Watch the "Waiting for Connection" Screen

The wizard now shows the connection progress. It polls `/api/v1/ws/status/{apiId}` every 2 seconds and transitions through four states:

State	Meaning
Waiting for connection	No WebSocket has been opened yet. The installer is still running on the host, or the host cannot reach the server.
Connected	The agent has opened a WebSocket but has not yet sent a report. The initial report runs in the background right after `serve` starts.
Receiving initial report	The agent sent a report and the server is processing it (OS type, hostname, IP, architecture, packages).
Done	Enrolment is complete. After a brief "Done" state the wizard redirects you to **Hosts** →.

If you close the wizard before reaching **Done**, the enrolment still completes in the background. The host appears in the **Hosts** list with status "Pending" until the first report lands, then flips to "Active".

What Gets Installed

Path (Linux / FreeBSD)	Path (Windows)	Purpose
`/usr/local/bin/patchmon-agent`	`C:\Program Files\PatchMon\patchmon-agent.exe`	Agent binary
`/etc/patchmon/config.yml`	`C:\ProgramData\PatchMon\config.yml`	Agent configuration
`/etc/patchmon/credentials.yml`	`C:\ProgramData\PatchMon\credentials.yml`	API credentials (`api_id` , `api_key`)
`/etc/patchmon/logs/patchmon-agent.log`	`C:\ProgramData\PatchMon\patchmon-agent.log`	Rolling log file
`/etc/systemd/system/patchmon-agent.service` *(systemd)*		systemd unit
`/etc/init.d/patchmon-agent` *(OpenRC)*		OpenRC init script
`/usr/local/etc/rc.d/patchmon_agent` *(FreeBSD)*		FreeBSD rc.d script
	Windows Service `PatchMonAgent`	Service Control Manager entry

For the full reference of every `config.yml` parameter, see Agent Configuration Reference (config.yml).

Troubleshooting First Check-in

If the "Waiting for connection" screen never moves past its initial state, work through the checks below in order.

1. Host cannot reach the server

From the target host:

```
curl -v https://patchmon.example.com/health
# expected: HTTP/1.1 200 and body "healthy"
```

If this fails:

DNS: `nslookup patchmon.example.com` or `dig patchmon.example.com`. Fix `/etc/resolv.conf` or update your internal DNS.

Routing / firewall: `traceroute patchmon.example.com` and check outbound TCP/443 is allowed. Corporate firewalls frequently block egress to new hostnames.

Proxy: if the host is behind an outbound HTTP proxy, set `HTTPS_PROXY` and `HTTP_PROXY` in the environment before pasting the install command.

2. Certificate validation fails

Symptoms: `curl: (60) SSL certificate problem` on Linux, or `Could not establish trust relationship for the SSL/TLS secure channel` on Windows.

Preferred fix: install your CA into the host's system trust store:

Debian/Ubuntu: copy the CA into `/usr/local/share/ca-certificates/` and run `update-ca-certificates`.

RHEL/Rocky/Fedora: copy the CA into `/etc/pki/ca-trust/source/anchors/` and run `update-ca-trust`.

Alpine: `apk add ca-certificates`, then copy and `update-ca-certificates`.

Windows: import the CA into **Local Computer → Trusted Root Certification Authorities** via `certlm.msc`.

Quick bypass (lab only): in the PatchMon web UI, go to **Settings → Server → Ignore SSL self-signed** and toggle it on. The server will then serve install scripts with `curl -sk` and inject `skip_ssl_verify: true` into the generated `config.yml`. On Windows, tick **Self-signed certificate (SSL bypass)** on Step 3 of the wizard before copying the command.

> *Do not use `skip_ssl_verify` in production. It disables TLS verification entirely and exposes the agent to man-in-the-middle attacks. See the Agent Configuration Reference for more on `skip_ssl_verify`.*

3. "CORS error" in the browser (wizard-side only)

The wizard itself calls `/api/v1/hosts/install` and `/api/v1/ws/status/{apiId}` from your browser. If either fails with a CORS error, your server's `CORS_ORIGIN` env var does not match the URL you are accessing PatchMon with.

Fix: set `CORS_ORIGIN` in the server `.env` to the **exact origin** (protocol + host + port) the browser uses. For example:

```
CORS_ORIGIN=https://patchmon.example.com
```

If users reach PatchMon from more than one URL (e.g. an external domain and an internal LAN address), comma-separate the values with no spaces:

```
CORS_ORIGIN=https://patchmon.example.com,https://patchmon.internal.lan
```

Then restart the server container (`docker compose restart server`). See Server Troubleshooting for the full CORS section.

4. Outbound port 443 blocked

Many cloud and enterprise networks allow port 80 but block 443 to arbitrary hosts. Test with:

```
# TCP reachability
nc -vz patchmon.example.com 443
# or
timeout 5 bash -c "</dev/tcp/patchmon.example.com/443" && echo "open" || echo
"blocked"
```

If blocked, allowlist outbound TCP/443 to your PatchMon server on the egress firewall.

5. Bootstrap token expired

The bootstrap token in the install command is valid for **5 minutes** and is **single-use**. If you copy the command, wait too long, and then run it, you will see:

```
ERROR: Failed to fetch credentials. Bootstrap token may have expired.
Please request a new installation script.
```

Go back to **Step 4** of the wizard and click **Copy command** again. This reuses the same host record but mints a fresh token.

6. Installer aborts on clock skew

If the host's clock is more than a few minutes off UTC, TLS handshakes will fail. On Linux:

```
sudo timedatectl set-ntp true
sudo timedatectl set-timezone Europe/London
```

On Windows (elevated PowerShell):

```
w32tm /resync
Set-TimeZone -Name "GMT Standard Time"
```

Then re-run the install command.

7. Agent installed but stays "Pending"

The binary installed and the service started, but the host never flips to "Active" in the UI. Run on the target host:

```
sudo patchmon-agent diagnostics
sudo patchmon-agent ping
sudo systemctl status patchmon-agent        # or: rc-service patchmon-agent status
sudo journalctl -u patchmon-agent -n 50     # or: tail -n 50
/etc/patchmon/logs/patchmon-agent.log
```

For the detailed checks, see Managing the PatchMon Agent: Common Troubleshooting and Agent Troubleshooting.

8. Reverse proxy drops the WebSocket

If the "Connected" state never appears but HTTP requests work fine, your reverse proxy is not forwarding the `Upgrade: websocket` handshake. The agent opens its WebSocket against `/api/v1/agents/ws`. See Server Troubleshooting: Agent cannot connect over WebSocket for Nginx / Traefik / Caddy config snippets.

Re-running the Installer on an Already-Enrolled Host

The Linux installer is idempotent. If you paste the install command on a host that is already enrolled and healthy, it will:

- Detect existing `config.yml`, `credentials.yml`, and binary.
- Run `patchmon-agent ping`.
- Exit early with `Agent is already configured and ping successful` and leave everything untouched.

To force a full reinstall:

```
sudo rm -f /etc/patchmon/config.yml /etc/patchmon/credentials.yml
# then paste the install command again
```

Or regenerate credentials from the UI first (this rotates the API key, and the next installer run will pick up the new one).

Next Steps

- Managing the PatchMon Agent: CLI commands, service control, logs, updates, removal.
- Agent Configuration Reference (config.yml): every config parameter, with defaults.
- Proxmox LXC Auto-Enrollment Guide: bulk-enrol containers via the auto-enrolment token API.
- Uninstalling the PatchMon Agent: remove the agent from a host.

Agent Troubleshooting: quick decision tree for common symptoms.

Chapter 9: Managing the PatchMon Agent

Overview

The PatchMon agent is a compiled Go binary (`patchmon-agent`) that runs as a persistent service on monitored hosts. It maintains a WebSocket connection to the PatchMon server for real-time communication, sends periodic package and system reports, collects integration data (Docker, compliance), and supports remote commands such as SSH proxy sessions.

This guide covers everything you need to manage the agent after installation: CLI commands, service management, log access, troubleshooting, updates, and removal.

Key Facts: Linux / FreeBSD

Property	Value
Binary location	`/usr/local/bin/patchmon-agent`
Configuration directory	`/etc/patchmon/`
Config file	`/etc/patchmon/config.yml`
Credentials file	`/etc/patchmon/credentials.yml`
Log file	`/etc/patchmon/logs/patchmon-agent.log`
Service name	`patchmon-agent` (systemd or OpenRC)
Runs as	`root`
Primary mode	`patchmon-agent serve` (long-lived service)

Key Facts: Windows

Property	Value
Binary location	`C:\Program Files\PatchMon\patchmon-agent.exe`
Configuration directory	`C:\ProgramData\PatchMon\`
Config file	`C:\ProgramData\PatchMon\config.yml`
Credentials file	`C:\ProgramData\PatchMon\credentials.yml`
Log file	`C:\ProgramData\PatchMon\patchmon-agent.log`
Service name	`PatchMonAgent` (Windows Service Control Manager)
Runs as	`LocalSystem`
Primary mode	`patchmon-agent serve` (long-lived service)
Architectures	amd64, arm64 (Surface Pro X / Copilot+ PCs). 32-bit Windows is not supported.

Table of Contents

CLI Command Reference

All commands must be run with elevated privileges:

Linux / FreeBSD: run as `root` or with `sudo`. The agent will refuse to run without root privileges.

Windows: run from an elevated PowerShell or `cmd` prompt (Run as Administrator). The agent requires Administrator rights to read installed-package inventory and manage its own service.

Quick Reference

```
patchmon-agent [command] [flags]
```

Command	Description	Requires Root
`serve`	Run the agent as a long-lived service (primary mode)	Yes
`report`	Collect and send a one-off system/package report	Yes
`report --json`	Output the report payload as JSON to stdout (does not send)	Yes
`ping`	Test connectivity and validate API credentials	Yes
`diagnostics`	Show comprehensive system and agent diagnostics	Yes
`config show`	Display current configuration and credential status	No
`config set-api`	Configure API credentials and server URL	Yes
`check-version`	Check if an agent update is available	Yes
`update-agent`	Download and install the latest agent version	Yes
`version`	Print the agent version	No

Global Flags

These flags can be used with any command:

Flag	Default	Description
`--config <path>`	`/etc/patchmon/config.yml`	Path to the configuration file
`--log-level <level>`	`info`	Override log level (`debug` , `info` , `warn` , `error`)
`--version`		Print the agent version and exit
`--help`		Show help for any command

`serve` : Run as a Service

```
sudo patchmon-agent serve
```

This is the primary operating mode. It is what the systemd/OpenRC service unit executes. When started, it:

- Loads configuration and credentials from `/etc/patchmon/`
- Sends a startup ping to the PatchMon server

- Establishes a persistent WebSocket connection (real-time commands)
- Sends an initial system report in the background
- Starts periodic reporting on the configured interval (default: 60 minutes)
- Syncs integration status and update interval from the server
- Listens for server-initiated commands (report now, update, compliance scan, etc.)

You should **not** normally run `serve` manually. It is managed by the system service. If you need to test it interactively, stop the service first to avoid duplicate instances.

Example: running interactively for debugging:

```
# Stop the service first
sudo systemctl stop patchmon-agent

# Run with debug logging to see all output
sudo patchmon-agent serve --log-level debug

# When finished, restart the service
sudo systemctl start patchmon-agent
```

`report`: Send a One-Off Report

```
sudo patchmon-agent report
```

Collects system information, installed packages, repository data, hardware info, network details, and integration data (Docker containers, compliance scans), then sends everything to the PatchMon server.

After sending the report, the agent also:

- Checks for available agent updates and applies them if auto-update is enabled
- Collects and sends integration data (Docker, compliance) separately

Output:

The command logs its progress to the configured log file. To see output directly, run with `--log-level debug` or check the log file.

`report --json`: Output Report as JSON

```
sudo patchmon-agent report --json
```

Collects the same system and package data but **outputs the full JSON payload to stdout** instead of sending it to the server. Useful for:

Debugging: see exactly what data the agent would send

Validation: verify package detection is correct

Integration: pipe JSON to other tools for analysis

Example: inspect the report payload:

```
sudo patchmon-agent report --json | jq .
```

Example: check which packages need updates:

```
sudo patchmon-agent report --json | jq '[.packages[] | select(.needsUpdate == true)] | length'
```

Example: save a snapshot for later comparison:

```
sudo patchmon-agent report --json > /tmp/patchmon-report-$(date +%Y%m%d).json
```

> ***Note:*** *The* `--json` *flag does* ***not*** *send data to the server and does* ***not*** *require valid API credentials. It only requires root access to read system package information.*

`ping`: Test Connectivity

```
sudo patchmon-agent ping
```

Tests two things:

Network connectivity: can the agent reach the PatchMon server?

API credentials: are the `api_id` and `api_key` valid?

Success output:

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

Failure output example:

```
Error: connectivity test failed: server returned 401
```

Use this command immediately after installation or whenever you suspect credential or network issues.

`diagnostics`: Full System Diagnostics

```
sudo patchmon-agent diagnostics
```

Displays a comprehensive diagnostic report covering:

Section	Details
System Information	OS type/version, architecture, kernel version, hostname, machine ID
Agent Information	Agent version, config file path, credentials file path, log file path, log level
Configuration Status	Whether config and credentials files exist (✅/❌)
Network Connectivity	Server URL, TCP reachability test, API credential validation
Recent Logs	Last 10 log entries from the agent log file

Example output:

```
PatchMon Agent Diagnostics v1.5.0

System Information:
  OS: ubuntu 22.04
  Architecture: amd64
  Kernel: 5.15.0-91-generic
  Hostname: webserver-01
  Machine ID: a1b2c3d4e5f6...

Agent Information:
  Version: 1.5.0
  Config File: /etc/patchmon/config.yml
  Credentials File: /etc/patchmon/credentials.yml
  Log File: /etc/patchmon/logs/patchmon-agent.log
  Log Level: info

Configuration Status:
  ✓ Config file exists
  ✓ Credentials file exists

Network Connectivity & API Credentials:
  Server URL: https://patchmon.example.com
  ✓ Server is reachable
  ✓ API is reachable and credentials are valid

Last 10 log entries:
  2026-02-12T10:30:00 level=info msg="Report sent successfully"
  ...
```

This is the best single command for troubleshooting agent issues.

`config show`: View Current Configuration

```
sudo patchmon-agent config show
```

Displays the current configuration values and credential status:

```
Configuration:
  Server: https://patchmon.example.com
  Agent Version: 1.5.0
  Config File: /etc/patchmon/config.yml
  Credentials File: /etc/patchmon/credentials.yml
  Log File: /etc/patchmon/logs/patchmon-agent.log
  Log Level: info

Credentials:
  API ID: patchmon_a1b2c3d4
  API Key: Set ✓
```

> ***Security:*** *The API key is never shown. The output only confirms whether it is set.*

`config set-api`: Configure Credentials

```
sudo patchmon-agent config set-api <API_ID> <API_KEY> <SERVER_URL>
```

Sets up the agent's API credentials and server URL. This command:

- Validates the inputs (non-empty, valid URL format)
- Saves the server URL to `/etc/patchmon/config.yml`
- Saves the credentials to `/etc/patchmon/credentials.yml` (with `600` permissions)
- Runs an automatic connectivity test (`ping`)

Example:

```
sudo patchmon-agent config set-api \
  patchmon_a1b2c3d4 \
  abcdef1234567890abcdef1234567890abcdef1234567890abcdef1234567890 \
  https://patchmon.example.com
```

> ***Note:*** *This command is primarily useful for manual installations or credential rotation. The standard install script sets credentials automatically.*

`check-version`: Check for Updates

```
sudo patchmon-agent check-version
```

Queries the PatchMon server to see if a newer agent version is available.

Output when up to date:

```
Agent is up to date (version 1.5.0)
```

Output when update is available:

```
Agent update available!
  Current version: 1.4.0
  Latest version: 1.5.0

To update, run: patchmon-agent update-agent
```

Output when auto-update is disabled on the server:

```
Current version: 1.4.0
Latest version: 1.5.0
Status: Auto-update disabled by server administrator

To update manually, run: patchmon-agent update-agent
```

`update-agent` : Update to Latest Version

```
sudo patchmon-agent update-agent
```

Downloads the latest agent binary from the PatchMon server and performs an in-place update. The process:

- Checks for recent updates (prevents update loops within 5 minutes)
- Queries the server for the latest version
- Downloads the new binary
- **Verifies binary integrity** via SHA-256 hash comparison (mandatory)
- Creates a timestamped backup of the current binary (e.g., `patchmon-agent.backup.20260212_143000`)
- Writes the new binary to a temporary file and validates it
- Atomically replaces the current binary
- Cleans up old backups (keeps the last 3)
- Restarts the service (systemd or OpenRC) via a helper script

Security features:

Binary hash verification is **mandatory**: the agent refuses to update if the server does not provide a hash

Hash mismatch (possible tampering) blocks the update

`skip_ssl_verify` is blocked in production environments for binary downloads

Backup files use `0700` permissions (owner-only)

> ***Note:*** *In normal operation, the agent auto-updates when the server signals a new version. You only need to run* `update-agent` *manually when auto-update is disabled or if you want to force an immediate update.*

`version`: Print Version

```
patchmon-agent version
# or
patchmon-agent --version
```

Prints the agent version:

```
PatchMon Agent v1.5.0
```

This does not require root access.

Service Management

The PatchMon agent runs as a system service managed by **systemd** (most Linux distributions) or **OpenRC** (Alpine Linux). In environments where neither is available, a **crontab** fallback is used.

Systemd (Ubuntu, Debian, CentOS, RHEL, Rocky, Alma, Fedora, etc.)

Service File Location

```
/etc/systemd/system/patchmon-agent.service
```

Service File Contents

The installer creates this unit file automatically:

```
[Unit]
Description=PatchMon Agent Service
After=network.target
Wants=network.target

[Service]
Type=simple
User=root
ExecStart=/usr/local/bin/patchmon-agent serve
Restart=always
RestartSec=10
WorkingDirectory=/etc/patchmon

# Logging
StandardOutput=journal
StandardError=journal
SyslogIdentifier=patchmon-agent

[Install]
WantedBy=multi-user.target
```

Key properties:

- `Restart=always` : the service automatically restarts if it crashes or is killed
- `RestartSec=10` : waits 10 seconds before restarting (prevents rapid restart loops)
- `After=network.target` : ensures the network is up before starting
- Logs go to the **systemd journal** as well as the agent's own log file

Common systemd Commands

```
# Check if the agent is running
sudo systemctl status patchmon-agent

# Start the agent
sudo systemctl start patchmon-agent

# Stop the agent
sudo systemctl stop patchmon-agent

# Restart the agent (e.g., after config changes)
sudo systemctl restart patchmon-agent

# Enable auto-start on boot
sudo systemctl enable patchmon-agent

# Disable auto-start on boot
sudo systemctl disable patchmon-agent

# Check if enabled
sudo systemctl is-enabled patchmon-agent

# Check if active
sudo systemctl is-active patchmon-agent

# Reload systemd after editing the service file manually
sudo systemctl daemon-reload
```

Reading systemd Journal Logs

```
# Follow logs in real-time (like tail -f)
sudo journalctl -u patchmon-agent -f

# Show last 50 log entries
sudo journalctl -u patchmon-agent -n 50

# Show logs since last boot
sudo journalctl -u patchmon-agent -b

# Show logs from the last hour
sudo journalctl -u patchmon-agent --since "1 hour ago"

# Show logs from a specific date
sudo journalctl -u patchmon-agent --since "2026-02-12 10:00:00"

# Show only errors
sudo journalctl -u patchmon-agent -p err

# Show logs without pager (useful for scripts)
sudo journalctl -u patchmon-agent --no-pager -n 100

# Export logs to a file
sudo journalctl -u patchmon-agent --no-pager > /tmp/patchmon-logs.txt
```

OpenRC (Alpine Linux)

Service File Location

```
/etc/init.d/patchmon-agent
```

Service File Contents

```
#!/sbin/openrc-run

name="patchmon-agent"
description="PatchMon Agent Service"
command="/usr/local/bin/patchmon-agent"
command_args="serve"
command_user="root"
pidfile="/var/run/patchmon-agent.pid"
command_background="yes"
working_dir="/etc/patchmon"

depend() {
    need net
    after net
}
```

Common OpenRC Commands

```
# Check if the agent is running
sudo rc-service patchmon-agent status

# Start the agent
sudo rc-service patchmon-agent start

# Stop the agent
sudo rc-service patchmon-agent stop

# Restart the agent
sudo rc-service patchmon-agent restart

# Add to default runlevel (auto-start on boot)
sudo rc-update add patchmon-agent default

# Remove from default runlevel
sudo rc-update del patchmon-agent default

# List services in default runlevel
sudo rc-update show default
```

Reading Logs on Alpine/OpenRC

OpenRC does not have a journal. Logs are written only to the agent's log file:

```
# Follow logs in real-time
sudo tail -f /etc/patchmon/logs/patchmon-agent.log

# Show last 50 lines
sudo tail -n 50 /etc/patchmon/logs/patchmon-agent.log

# Search logs for errors
sudo grep -i "error\|fail" /etc/patchmon/logs/patchmon-agent.log
```

Crontab Fallback (No Init System)

In minimal containers or environments without systemd or OpenRC, the installer sets up a crontab entry:

```
@reboot /usr/local/bin/patchmon-agent serve >/dev/null 2>&1
```

The agent is also started immediately in the background during installation.

Managing the Crontab Fallback

```
# Check for PatchMon crontab entries
crontab -l | grep patchmon

# Stop the agent manually
sudo pkill -f 'patchmon-agent serve'

# Start the agent manually
sudo /usr/local/bin/patchmon-agent serve &

# Restart the agent
sudo pkill -f 'patchmon-agent serve' && sudo /usr/local/bin/patchmon-agent serve &
```

Windows Service Manager

On Windows the agent runs as a native Windows Service managed by Service Control Manager (SCM). The installer registers it as `PatchMonAgent` with `StartupType=Automatic` so it starts with the OS.

Service File Location

Windows has no service file equivalent to systemd. The service is stored in the SCM registry. Service metadata:

Property	Value
Service name	`PatchMonAgent`
Display name	PatchMon Agent
Binary path	`"C:\Program Files\PatchMon\patchmon-agent.exe" serve`
Startup type	Automatic
Account	`LocalSystem` (default)
Config directory	`C:\ProgramData\PatchMon\`
Config file	`C:\ProgramData\PatchMon\config.yml`
Credentials file	`C:\ProgramData\PatchMon\credentials.yml`
Log file	`C:\ProgramData\PatchMon\patchmon-agent.log`

The binary lives under `C:\Program Files\PatchMon\` and is added to the system `PATH`, so `patchmon-agent` works from any elevated PowerShell or `cmd` prompt without the full path.

Common Commands (run in elevated PowerShell)

```
# Service status
Get-Service -Name PatchMonAgent

# Start / stop / restart
Start-Service   -Name PatchMonAgent
Stop-Service    -Name PatchMonAgent -Force
Restart-Service -Name PatchMonAgent

# Disable auto-start (still able to start manually)
Set-Service -Name PatchMonAgent -StartupType Manual

# Re-enable auto-start
Set-Service -Name PatchMonAgent -StartupType Automatic

# Tail the log file
Get-Content 'C:\ProgramData\PatchMon\patchmon-agent.log' -Tail 50 -Wait

# Service events (Event Viewer)
Get-WinEvent -LogName System -MaxEvents 20 |
  Where-Object { $_.ProviderName -eq 'Service Control Manager' -and $_.Message -
like '*PatchMonAgent*' }
```

The agent CLI itself is also available from any elevated prompt (since the install path is in `PATH`):

```
# Test connectivity
patchmon-agent ping

# Force a one-off report
patchmon-agent report

# Print detailed diagnostics
patchmon-agent diagnostics

# Show current config
patchmon-agent config show
```

Troubleshooting on Windows

Service won't start. Check the latest application-log entries: `Get-WinEvent -LogName Application -MaxEvents 20 | Where-Object ProviderName -like '*PatchMon*'`. Common causes: missing `credentials.yml`, stale `skip_ssl_verify` setting after a server cert change, firewall blocking outbound 443/WSS.

ARM64 device showing x64 in `Get-ComputerInfo`. If you ran an older (pre-2.0.0) installer the amd64 binary may be installed and running under x64 emulation. Re-run the latest install script; it detects `PROCESSOR_ARCHITEW6432=ARM64` and swaps in the native `patchmon-agent-windows-arm64.exe`.

SmartScreen / Defender blocking the `.exe`. The binary is unsigned as of v2.0.0. Use `Unblock-File 'C:\Program Files\PatchMon\patchmon-agent.exe'` or authorise it via Windows Security → Virus & threat protection → Allowed threats. Code-signing is planned for a future release.

TLS errors against the PatchMon server. Windows 10 pre-1903 defaults to TLS 1.0/1.1; the installer explicitly enables TLS 1.2 at the session level. For older hosts, update the .NET Framework or set the registry keys documented by Microsoft in the `SchUseStrongCrypto` KB article.

Viewing Logs

The agent writes logs to two locations depending on the service manager:

Service Manager	Journal / Event Log	Log File
systemd	✅ `journalctl -u patchmon-agent`	✅ `/etc/patchmon/logs/patchmon-agent.log`
OpenRC	❌	✅ `/etc/patchmon/logs/patchmon-agent.log`
Crontab	❌	✅ `/etc/patchmon/logs/patchmon-agent.log`
Windows SCM	✅ Event Viewer (Application log, source `PatchMonAgent`) for service lifecycle events	✅ `C:\ProgramData\PatchMon\patchmon-agent.log`

Log File Details: Linux / FreeBSD

Property	Value
Location	`/etc/patchmon/logs/patchmon-agent.log`
Max size	10 MB per file
Max backups	5 rotated files
Max age	14 days
Compression	Yes (old logs compressed automatically)
Rotation	Automatic (handled by the agent, not logrotate)

Log File Details: Windows

Property	Value
Location	`C:\ProgramData\PatchMon\patchmon-agent.log`
Max size	10 MB per file
Max backups	5 rotated files
Max age	14 days
Compression	Yes (old logs compressed automatically)
Rotation	Automatic (handled by the agent)
Tail live	`Get-Content 'C:\ProgramData\PatchMon\patchmon-agent.log' -Tail 50 -Wait`

The agent uses the lumberjack (https://github.com/natefinsh/lumberjack.v2) library for built-in log rotation. You do not need to configure logrotate separately.

Log Levels

Set the log level in `/etc/patchmon/config.yml` or via the `--log-level` flag:

Level	Description	Use Case
`debug`	Verbose: every operation, request/response bodies, package details	Active troubleshooting
`info`	Normal: key events, report summaries, connectivity status	Default / production
`warn`	Warnings: non-critical failures, retries, degraded operation	Noise reduction
`error`	Errors only: critical failures that need attention	Minimal logging

Change log level temporarily (until service restart):

```
sudo patchmon-agent report --log-level debug
```

Change log level permanently:

Edit `/etc/patchmon/config.yml`:

```
log_level: "debug"
```

Then restart the service:

```
sudo systemctl restart patchmon-agent
# or
sudo rc-service patchmon-agent restart
```

On Windows, edit `C:\ProgramData\PatchMon\config.yml` and restart the service from an elevated PowerShell:

```
Restart-Service -Name PatchMonAgent
```

Log Format

Logs use structured text format with timestamps:

```
2026-02-12T10:30:00 level=info msg="Detecting operating system..."
2026-02-12T10:30:00 level=info msg="Detected OS" osType=ubuntu osVersion=22.04
2026-02-12T10:30:01 level=info msg="Found packages" count=247
2026-02-12T10:30:02 level=info msg="Sending report to PatchMon server..."
2026-02-12T10:30:03 level=info msg="Report sent successfully"
2026-02-12T10:30:03 level=info msg="Processed packages" count=247
2026-02-12T10:30:08 level=info msg="Agent is up to date" version=1.5.0
```

Testing and Diagnostics

Quick Health Check

Run these commands in order to verify the agent is working correctly:

```
# 1. Is the service running?
sudo systemctl status patchmon-agent     # systemd
# or
sudo rc-service patchmon-agent status    # OpenRC

# 2. Can the agent reach the server?
sudo patchmon-agent ping

# 3. Full diagnostics
sudo patchmon-agent diagnostics

# 4. What data would the agent send?
sudo patchmon-agent report --json | jq '.hostname, .os_type, .os_version,
.packages | length'
```

Debugging a Problem

If the agent is not reporting data or appears offline:

```
# Step 1: Check service status
sudo systemctl status patchmon-agent

# Step 2: Check recent logs for errors
sudo journalctl -u patchmon-agent -n 30 --no-pager
# or
sudo tail -n 30 /etc/patchmon/logs/patchmon-agent.log

# Step 3: Run diagnostics for full picture
sudo patchmon-agent diagnostics

# Step 4: Test connectivity explicitly
sudo patchmon-agent ping

# Step 5: If needed, restart with debug logging temporarily
sudo systemctl stop patchmon-agent
sudo patchmon-agent serve --log-level debug
# (Ctrl+C to stop, then restart the service normally)
sudo systemctl start patchmon-agent
```

Manual Reporting

While the agent sends reports automatically on its configured interval, you can trigger a report at any time:

```
# Send a report immediately
sudo patchmon-agent report
```

This is useful after:

- Making system changes (installing/removing packages)
- Verifying the agent can communicate after a network change
- Testing after reconfiguring the agent

The `report` command also triggers integration data collection (Docker, compliance) and checks for agent updates, identical to a scheduled report.

Inspecting Report Data

To see exactly what the agent collects without sending anything:

```
# Full JSON output
sudo patchmon-agent report --json

# Pretty-print with jq
sudo patchmon-agent report --json | jq .

# Just the package count and update summary
sudo patchmon-agent report --json | jq '{
  total_packages: (.packages | length),
  needs_update: [.packages[] | select(.needsUpdate)] | length,
  security_updates: [.packages[] | select(.isSecurityUpdate)] | length,
  hostname: .hostname,
  os: "\(.osType) \(.osVersion)"
}'
```

Configuration Management

For comprehensive documentation on all configuration parameters, see the Agent Configuration Reference (config.yml).

Quick Configuration Tasks

View current config:

```
sudo patchmon-agent config show
```

Set or change API credentials:

```
sudo patchmon-agent config set-api <API_ID> <API_KEY> <SERVER_URL>
```

Edit config file directly:

```
sudo nano /etc/patchmon/config.yml
sudo systemctl restart patchmon-agent  # restart to apply changes
```

When do changes require a restart?

Change	Restart Needed?
`patchmon_server`	Yes
`log_level`	Yes
`skip_ssl_verify`	Yes
`update_interval`	No (synced from server via WebSocket)
`integrations.docker`	No (synced from server)
`integrations.compliance`	No (synced from server)
`integrations.ssh-proxy-enabled`	Yes (manual config only)
Credentials (`api_id` / `api_key`)	Yes

Agent Updates

How Auto-Update Works

The agent checks for updates in two ways:

- **After each report**: the agent queries the server for the latest version and updates automatically if one is available
- **Server-initiated**: the server can push an `update_notification` or `update_agent` command via WebSocket

When an update is detected:

- The new binary is downloaded from the PatchMon server
- SHA-256 hash is verified against the server-provided hash (mandatory)
- The current binary is backed up (last 3 backups are kept)
- The new binary replaces the old one atomically
- The service is restarted via a helper script

Manual Update

Linux / FreeBSD:

```
# Check what version is available
sudo patchmon-agent check-version

# Apply the update
sudo patchmon-agent update-agent
```

Windows (elevated PowerShell):

```
patchmon-agent check-version
patchmon-agent update-agent
```

On Windows the `update-agent` flow stops the `PatchMonAgent` service, atomically replaces `C:\Program Files\PatchMon\patchmon-agent.exe` with the verified new binary (keeping the last 3 timestamped `.backup.*` files alongside), and restarts the service.

Update Safety Features

- **Hash verification**: refuses to install if the binary hash does not match
- **Update loop prevention**: blocks re-updates within 5 minutes of a previous update
- **Automatic backup**: creates a timestamped backup before replacing the binary
- **Rollback**: if the new binary fails validation, the update is aborted
- **Version verification**: checks that the downloaded binary reports the expected version

Backup Files

Update backups are stored alongside the binary.

Linux / FreeBSD:

```
/usr/local/bin/patchmon-agent                          # current binary
/usr/local/bin/patchmon-agent.backup.20260212_143000  # backup from update
/usr/local/bin/patchmon-agent.backup.20260210_090000  # older backup
/usr/local/bin/patchmon-agent.backup.20260201_120000  # oldest backup (3 kept)
```

Windows:

```
C:\Program Files\PatchMon\patchmon-agent.exe                          # current
binary
C:\Program Files\PatchMon\patchmon-agent.exe.backup.20260212_143000  # backup from
update
C:\Program Files\PatchMon\patchmon-agent.exe.backup.20260210_090000  # older
backup
C:\Program Files\PatchMon\patchmon-agent.exe.backup.20260201_120000  # oldest
backup (3 kept)
```

The agent automatically removes backups beyond the most recent 3.

Agent Removal

There are two methods to remove the PatchMon agent from a host.

Method 1: Server-Provided Removal Script (Recommended)

Linux / FreeBSD:

```
curl -s https://patchmon.example.com/api/v1/hosts/remove | sudo sh
```

Windows (run in elevated PowerShell):

```
$ProgressPreference = 'SilentlyContinue'
irm https://patchmon.example.com/api/v1/hosts/remove?os=windows | iex
```

(Or download first, inspect, then run: `irm ... -OutFile remove.ps1; .\remove.ps1`.)

The Linux/FreeBSD script handles everything:

- Stops the service (systemd, OpenRC, or crontab)
- Removes the service file and reloads the daemon
- Kills any remaining agent processes
- Removes the agent binary and legacy scripts
- Removes configuration files and directories (`/etc/patchmon/`)
- Removes log files
- Cleans up crontab entries

The Windows script handles the equivalent on Windows:

- Stops the `PatchMonAgent` service
- Kills any residual `patchmon-agent.exe` processes
- Deletes the service via `sc.exe delete PatchMonAgent`
- Removes `C:\Program Files\PatchMon\` and `C:\ProgramData\PatchMon\`
- Removes the install path from the system `PATH`

Options:

Environment Variable	Default	Description
`REMOVE_BACKUPS`	`0`	Set to `1` to also remove backup files
`SILENT`	not set	Set to `1` for silent mode (minimal output)

Examples:

```
# Standard removal (preserves backups)
curl -s https://patchmon.example.com/api/v1/hosts/remove | sudo sh

# Remove everything including backups
curl -s https://patchmon.example.com/api/v1/hosts/remove | sudo REMOVE_BACKUPS=1
sh

# Silent removal (for automation)
curl -s https://patchmon.example.com/api/v1/hosts/remove | sudo SILENT=1 sh

# Silent removal with backup cleanup
curl -s https://patchmon.example.com/api/v1/hosts/remove | sudo REMOVE_BACKUPS=1
SILENT=1 sh
```

Method 2: Manual Removal

If the server is unreachable, you can remove the agent manually.

Linux / FreeBSD:

```
# 1. Stop and disable the service
sudo systemctl stop patchmon-agent
sudo systemctl disable patchmon-agent
sudo rm -f /etc/systemd/system/patchmon-agent.service
sudo systemctl daemon-reload
# or for OpenRC:
sudo rc-service patchmon-agent stop
sudo rc-update del patchmon-agent default
sudo rm -f /etc/init.d/patchmon-agent

# 2. Kill any remaining processes
sudo pkill -f patchmon-agent

# 3. Remove the binary and backups
sudo rm -f /usr/local/bin/patchmon-agent
sudo rm -f /usr/local/bin/patchmon-agent.backup.*

# 4. Remove configuration and logs
sudo rm -rf /etc/patchmon/

# 5. Remove crontab entries (if any)
crontab -l 2>/dev/null | grep -v "patchmon-agent" | crontab -

# 6. Verify removal
which patchmon-agent          # should return nothing
ls /etc/patchmon/ 2>/dev/null # should show "No such file or directory"
systemctl status patchmon-agent 2>&1 | head -1  # should show "not found"
```

Windows (elevated PowerShell):

```
# 1. Stop and delete the service
Stop-Service  -Name PatchMonAgent -Force -ErrorAction SilentlyContinue
sc.exe delete PatchMonAgent

# 2. Kill any remaining processes
Get-Process -Name patchmon-agent -ErrorAction SilentlyContinue | Stop-Process -
Force

# 3. Remove the binary and data directories
Remove-Item -Recurse -Force 'C:\Program Files\PatchMon'
Remove-Item -Recurse -Force 'C:\ProgramData\PatchMon'

# 4. Remove the install path from the system PATH
$installPath = 'C:\Program Files\PatchMon'
$currentPath = [Environment]::GetEnvironmentVariable('Path',
[EnvironmentVariableTarget]::Machine)
$newPath = ($currentPath -split ';' | Where-Object { $_ -and $_ -ne $installPath
}) -join ';'
[Environment]::SetEnvironmentVariable('Path', $newPath,
[EnvironmentVariableTarget]::Machine)

# 5. Verify
Get-Service -Name PatchMonAgent -ErrorAction SilentlyContinue   # should return
nothing
Test-Path 'C:\Program Files\PatchMon'                           # should be False
```

> ***Important:*** *Removing the agent from the host does* ***not*** *remove the host entry from PatchMon. To fully decommission a host, also delete it from the PatchMon web UI (Hosts page).*

Common Troubleshooting

Agent Shows "Pending" in PatchMon

The host was created but the agent has not yet sent its first report.

```
# Check service is running
sudo systemctl status patchmon-agent

# Test connectivity
sudo patchmon-agent ping

# If ping fails, check the server URL
sudo patchmon-agent config show

# Force an immediate report
sudo patchmon-agent report
```

Agent Shows "Offline" in PatchMon

The agent's WebSocket connection is down.

```
# Check if the service is running
sudo systemctl is-active patchmon-agent

# If not running, check why it stopped
sudo journalctl -u patchmon-agent -n 50 --no-pager

# Restart the service
sudo systemctl restart patchmon-agent
```

"Permission Denied" Errors

```
# All agent commands require root
sudo patchmon-agent <command>

# Verify file permissions
ls -la /etc/patchmon/config.yml        # should be -rw------- root
ls -la /etc/patchmon/credentials.yml   # should be -rw------- root
ls -la /usr/local/bin/patchmon-agent   # should be -rwxr-xr-x root
```

"Credentials File Not Found"

```
# Check if credentials exist
ls -la /etc/patchmon/credentials.yml

# If missing, reconfigure
sudo patchmon-agent config set-api <API_ID> <API_KEY> <SERVER_URL>
```

"Connectivity Test Failed"

```
# Run full diagnostics
sudo patchmon-agent diagnostics

# Test network connectivity manually
curl -I https://patchmon.example.com

# Check DNS resolution
nslookup patchmon.example.com
# or
dig patchmon.example.com

# Check firewall rules
sudo iptables -L -n | grep -i drop
```

SSL Certificate Errors

```
# For self-signed certificates in non-production environments:
# Edit /etc/patchmon/config.yml
skip_ssl_verify: true

# Then restart
sudo systemctl restart patchmon-agent
```

> ***Warning:*** *`skip_ssl_verify: true` disables TLS certificate verification entirely, exposing the agent to man-in-the-middle attacks. Only use in lab or air-gapped deployments. The preferred fix for self-signed or internal CA certificates is to install the CA into the host's system trust store (e.g. `/usr/local/share/ca-certificates/` + `update-ca-certificates` on Debian/Ubuntu, `/etc/pki/ca-trust/source/anchors/` + `update-ca-trust` on RHEL/Fedora) rather than disabling verification. You can also set `PATCHMON_SKIP_SSL_VERIFY=true` as an environment variable instead of editing `config.yml`.*

Service Keeps Restarting

Check for crash loops:

```
# See restart count and recent failures
sudo systemctl status patchmon-agent

# Check logs around restart times
sudo journalctl -u patchmon-agent --since "30 minutes ago" --no-pager

# Common causes:
# - Invalid config.yml (syntax error)
# - Invalid credentials
# - Server unreachable (agent retries but logs errors)
```

Agent Not Auto-Updating

```
# Check current version
patchmon-agent version

# Check if update is available
sudo patchmon-agent check-version

# Check if auto-update was recently performed
ls -la /etc/patchmon/.last_update_timestamp

# Try manual update
sudo patchmon-agent update-agent

# Check for update loop prevention (5-minute cooldown)
# If you see "update was performed X ago", wait 5 minutes
```

Architecture and Supported Platforms

Supported Architectures: Linux

Architecture	Binary Name	Common Devices
`amd64`	`patchmon-agent-linux-amd64`	Standard servers, VMs, most cloud instances
`arm64`	`patchmon-agent-linux-arm64`	ARM servers, Raspberry Pi 4+, AWS Graviton
`arm` (v6/v7)	`patchmon-agent-linux-arm`	Raspberry Pi 2/3, older ARM boards
`386`	`patchmon-agent-linux-386`	32-bit x86 systems (legacy)

Supported Architectures: FreeBSD

Architecture	Binary Name	Common Devices
`amd64`	`patchmon-agent-freebsd-amd64`	Standard FreeBSD servers and jails
`arm64`	`patchmon-agent-freebsd-arm64`	ARM FreeBSD servers
`arm` (v6/v7)	`patchmon-agent-freebsd-arm`	Older ARM FreeBSD boards
`386`	`patchmon-agent-freebsd-386`	32-bit x86 FreeBSD (legacy)

Supported Architectures: Windows

Architecture	Binary Name	Common Devices
`amd64`	`patchmon-agent-windows-amd64.exe`	Standard Windows PCs and servers (Intel/AMD 64-bit)
`arm64`	`patchmon-agent-windows-arm64.exe`	Surface Pro X, Surface Pro 9/11 with Snapdragon, Surface Laptop 7, Windows Dev Kit 2023, Copilot+ PCs (Snapdragon X Elite/Plus. Lenovo, Dell, HP, Samsung, etc.)

32-bit Windows (x86) is not supported. All Microsoft-supported Windows versions as of 2026 are 64-bit only. Windows 10 32-bit reached EOL on 14 October 2025, and Windows 11 has never had a 32-bit edition. The PowerShell installer aborts with a clear error on 32-bit hosts rather than installing an unusable binary.

The installer detects the OS architecture via `PROCESSOR_ARCHITECTURE` / `PROCESSOR_ARCHITEW6432` and downloads the matching binary automatically. ARM64 devices get a native ARM64 binary rather than the x64 emulation fallback.

Supported Operating Systems: Linux

Distribution	Init System	Package Manager	Notes
Ubuntu	systemd	apt	All LTS versions supported
Debian	systemd	apt	10+
CentOS	systemd	yum/dnf	7+
RHEL	systemd	yum/dnf	7+
Rocky Linux	systemd	dnf	All versions
AlmaLinux	systemd	dnf	All versions
Fedora	systemd	dnf	Recent versions
Alpine Linux	OpenRC	apk	3.x+

Supported Operating Systems: Windows

Version	Service Manager	Package Manager	Notes
Windows 10 (64-bit)	Service Control Manager (SCM)	winget / chocolatey (reporting only)	amd64. Windows 10 32-bit reached EOL on 14 October 2025 and is not supported.
Windows 11	SCM	winget / chocolatey (reporting only)	amd64 and native ARM64 on Copilot+ PCs
Windows Server 2019	SCM	winget / chocolatey (reporting only)	amd64
Windows Server 2022	SCM	winget / chocolatey (reporting only)	amd64
Windows Server 2025	SCM	winget / chocolatey (reporting only)	amd64 and ARM64

The Windows agent reports installed packages (via `winget`, `chocolatey`, and MSI inventory) but does not execute patch deployments. PatchMon integrates with third-party patching vendors for Windows rollout.

Supported Operating Systems: FreeBSD

Version	Init System	Package Manager	Notes
FreeBSD 13.x	rc.d	pkg	amd64, arm64, arm, 386
FreeBSD 14.x	rc.d	pkg	amd64, arm64, arm, 386

FreeBSD base-system updates are detected via `freebsd-update fetch`.

Resource Usage

The agent is lightweight:

Resource	Typical Usage
Memory	~15-30 MB RSS
CPU	Near zero when idle; brief spikes during report collection
Disk	~15 MB (binary) + logs
Network	WebSocket keepalive (~1 KB/min); report payloads vary by package count

See Also:

- Agent Configuration Reference (config.yml): detailed documentation on every config parameter
- Proxmox LXC Auto-Enrollment Guide: bulk agent deployment on Proxmox
- Integration API Documentation: API endpoints used by the agent

---

Chapter 10: Uninstalling the PatchMon Agent

Overview

There are two sides to decommissioning a host in PatchMon:

- **Remove the agent from the host**: stops the service, deletes the binary, config, credentials, logs, and service unit.
- **Remove the host record from the PatchMon server**: deletes the row in the database, its API credentials, historical reports, and any group memberships.

You almost always want to do both. This page covers the script-driven removal, the manual fallback for each platform, and how to clean up the UI side.

The agent binary does **not** have a built-in `uninstall` subcommand. Removal is driven by a server-generated shell script (Linux/FreeBSD) or PowerShell script (Windows). The `Agent Removal` section in Managing the PatchMon Agent contains the same commands in a more compact reference form: this page is the detailed walkthrough with manual fallbacks for when the script-driven approach is not viable.

What Gets Removed

The server-provided removal scripts delete everything the installer wrote:

Artefact (Linux / FreeBSD)	Artefact (Windows)
Running `patchmon-agent` processes	Running `patchmon-agent.exe` processes
systemd service + `/etc/systemd/system/patchmon-agent.service`	`PatchMonAgent` Windows Service
OpenRC service + `/etc/init.d/patchmon-agent`	-
FreeBSD rc.d script at `/usr/local/etc/rc.d/patchmon_agent`	-
Crontab entries containing `patchmon-agent`	-
Agent binary `/usr/local/bin/patchmon-agent`	`C:\Program Files\PatchMon\`
Configuration directory `/etc/patchmon/` (config, credentials, logs)	`C:\ProgramData\PatchMon\`
Log file `/var/log/patchmon-agent.log` (legacy path, if present)	-
Backup files (`*.backup.*`), only when `REMOVE_BACKUPS=1` is set	Backup `.exe.backup.*` files
	Install path removed from system `PATH`

The host record on the server is **not** removed by these scripts. Use the **Delete Host** button in the web UI (covered below).

Method 1: Server-Provided Removal Script (Recommended)

The server exposes a public, unauthenticated endpoint that returns the removal script: `GET /api/v1/hosts/remove`. Pass `?os=windows` to get the PowerShell version.

Linux / FreeBSD

```
curl -s https://patchmon.example.com/api/v1/hosts/remove | sudo sh
```

The script is idempotent, running it twice is harmless. It prints a progress log and finishes with a "Removal Summary" block.

Options (environment variables set before `sh`):

Variable	Default	Effect
`REMOVE_BACKUPS`	`0`	Set to `1` to also delete `*.backup.*` files (config backups, binary backups, log rotations).
`SILENT`	unset	Set to `1` for minimal output (useful in automation / Ansible).

Examples:

```
# Standard removal, keep backups (safest)
curl -s https://patchmon.example.com/api/v1/hosts/remove | sudo sh

# Nuke everything including backups
curl -s https://patchmon.example.com/api/v1/hosts/remove | sudo REMOVE_BACKUPS=1
sh

# Silent removal (for Ansible / cron)
curl -s https://patchmon.example.com/api/v1/hosts/remove | sudo SILENT=1 sh

# Silent + full cleanup
curl -s https://patchmon.example.com/api/v1/hosts/remove | sudo REMOVE_BACKUPS=1
SILENT=1 sh
```

> *If your PatchMon server uses a self-signed certificate and the target host does not trust it, the server automatically serves the script with `-sk` (insecure) baked in when `Settings → Server → Ignore SSL self-signed` is enabled. If it is not, add `-k` to the initial `curl` yourself: `curl -sk https://patchmon.example.com/api/v1/hosts/remove | sudo sh`.*

Windows (elevated PowerShell)

```
$ProgressPreference = 'SilentlyContinue'
irm https://patchmon.example.com/api/v1/hosts/remove?os=windows | iex
```

Or download first, inspect, then run:

```
irm https://patchmon.example.com/api/v1/hosts/remove?os=windows -OutFile
patchmon_remove.ps1
# inspect patchmon_remove.ps1
.\patchmon_remove.ps1 -RemoveAll -Force
```

Script parameters:

Parameter	Default	Effect
`-RemoveConfig`	off	Remove `C:\ProgramData\PatchMon\` (config, credentials, logs).
`-RemoveLogs`	off	Remove log files.
`-RemoveAll`	off	Shortcut for `-RemoveConfig` + `-RemoveLogs`.
`-Force`	off	Skip interactive confirmation prompts.
`-InstallPath`	`C:\Program Files\PatchMon`	Override install location (rare).
`-ConfigPath`	`C:\ProgramData\PatchMon`	Override config location (rare).

By default the Windows script removes the service and the binary but **keeps** config and logs. Pass `-RemoveAll` to wipe everything.

Method 2: Manual Removal

Use the manual steps when:

- The PatchMon server is unreachable and you cannot pull the removal script.
- You are removing a legacy / corrupted install where the script is failing.
- You want to script removal yourself with a configuration management tool.

Linux: systemd

```
# 1. Stop and disable the service
sudo systemctl stop patchmon-agent
sudo systemctl disable patchmon-agent
sudo rm -f /etc/systemd/system/patchmon-agent.service
sudo systemctl daemon-reload

# 2. Kill any stragglers
sudo pkill -f patchmon-agent

# 3. Remove binary and timestamped backups
sudo rm -f /usr/local/bin/patchmon-agent
sudo rm -f /usr/local/bin/patchmon-agent.backup.*

# 4. Remove config, credentials, and logs
sudo rm -rf /etc/patchmon/

# 5. Remove any stale crontab entries
crontab -l 2>/dev/null | grep -v "patchmon-agent" | crontab -

# 6. Verify
which patchmon-agent                            # should print nothing
systemctl status patchmon-agent 2>&1 | head -1  # should show "not found"
ls /etc/patchmon/ 2>/dev/null                   # should be absent
```

Linux: OpenRC (Alpine)

```
# 1. Stop, remove from runlevel, delete init script
sudo rc-service patchmon-agent stop
sudo rc-update del patchmon-agent default
sudo rm -f /etc/init.d/patchmon-agent

# 2. Kill any stragglers
sudo pkill -f patchmon-agent

# 3. Remove binary and config
sudo rm -f /usr/local/bin/patchmon-agent /usr/local/bin/patchmon-agent.backup.*
sudo rm -rf /etc/patchmon/

# 4. Verify
rc-service patchmon-agent status 2>&1 | head -1
```

FreeBSD: rc.d

```
# 1. Stop the service
service patchmon_agent stop

# 2. Disable auto-start (remove or comment the enable line in rc.conf.local)
sysrc -x patchmon_agent_enable 2>/dev/null || true
sed -i '' '/patchmon_agent_enable/d' /etc/rc.conf.local 2>/dev/null || true

# 3. Remove the rc.d script
rm -f /usr/local/etc/rc.d/patchmon_agent

# 4. Kill any stragglers
pkill -f patchmon-agent || true
rm -f /var/run/patchmon_agent.pid

# 5. Remove binary, config, and backups
rm -f /usr/local/bin/patchmon-agent /usr/local/bin/patchmon-agent.backup.*
rm -rf /etc/patchmon/

# 6. Verify
service patchmon_agent status 2>&1 | head -1
```

> *The FreeBSD rc.d script name uses an underscore (`patchmon_agent`), not a hyphen. This matches FreeBSD's rc convention. Do not be surprised by the inconsistency with Linux.*

Crontab-Only Hosts (minimal containers)

On systems without systemd, OpenRC, or rc.d, the installer adds a `@reboot` crontab entry and starts the agent in the background. To remove:

```
# 1. Kill the running agent
sudo pkill -f 'patchmon-agent serve'

# 2. Strip the crontab entry
crontab -l 2>/dev/null | grep -v "patchmon-agent" | crontab -

# 3. Remove binary and config
sudo rm -f /usr/local/bin/patchmon-agent /usr/local/bin/patchmon-agent.backup.*
sudo rm -rf /etc/patchmon/

# 4. Verify no processes remain
pgrep -f patchmon-agent
```

Windows: elevated PowerShell

```
# 1. Stop and delete the service
Stop-Service -Name PatchMonAgent -Force -ErrorAction SilentlyContinue
sc.exe delete PatchMonAgent

# 2. Kill any stragglers
Get-Process -Name patchmon-agent -ErrorAction SilentlyContinue | Stop-Process -
Force

# 3. Remove binary and data directories
Remove-Item -Recurse -Force 'C:\Program Files\PatchMon'
Remove-Item -Recurse -Force 'C:\ProgramData\PatchMon'

# 4. Strip the install path from the system PATH
$installPath = 'C:\Program Files\PatchMon'
$currentPath = [Environment]::GetEnvironmentVariable('Path',
[EnvironmentVariableTarget]::Machine)
$newPath = ($currentPath -split ';' | Where-Object { $_ -and $_ -ne $installPath
}) -join ';'
[Environment]::SetEnvironmentVariable('Path', $newPath,
[EnvironmentVariableTarget]::Machine)

# 5. Verify
Get-Service -Name PatchMonAgent -ErrorAction SilentlyContinue   # should print
nothing
Test-Path 'C:\Program Files\PatchMon'                           # should be False
Test-Path 'C:\ProgramData\PatchMon'                             # should be False
```

Step 3: Delete the Host Record from PatchMon

Removing the agent from the host does **not** delete the host record in PatchMon's database. The host will show as offline / stale. To fully decommission:

1. Log in as a user with `can_manage_hosts`.
2. Navigate to **Hosts**.
3. Find the host (or multi-select several) and click **Delete Host** (single) or the **Delete** bulk action (multi).
4. Confirm in the modal. This removes:
   - The host row
   - Its API credentials (hashed `api_id` / `api_key`)
   - Its report history, package inventory, repository list, compliance scans, and Docker inventory
   - Its host-group memberships

If you intend to re-enrol the same machine later, it will come back as a brand new host with a new `api_id`.

> ***Caveat for credential reuse.*** *The agent's `credentials.yml` contains the old API ID. If you removed the host record but left the binary installed, the agent will start logging `401 Unauthorized` every check-in because the server no longer knows that API ID. Always pair UI deletion with one of the removal methods above.*

Troubleshooting Removal

Script fails with "Permission denied"

You forgot `sudo`:

```
curl -s https://patchmon.example.com/api/v1/hosts/remove | sudo sh
```

On Windows, open PowerShell with **Run as Administrator**: a non-elevated shell will error with `This script must be run as Administrator`.

Service is still running after removal

On systemd:

```
sudo systemctl status patchmon-agent
sudo systemctl stop patchmon-agent
sudo systemctl disable patchmon-agent
sudo rm -f /etc/systemd/system/patchmon-agent.service
sudo systemctl daemon-reload
sudo pkill -9 -f patchmon-agent   # force-kill stragglers
```

On Windows, if `sc.exe delete PatchMonAgent` reports "The specified service has been marked for deletion" and the service is still listed, reboot once. SCM cannot purge a service while the binary's file handle is still open. Alternatively, close any Event Viewer / Services.msc windows and retry.

Config files still present after script ran

The Linux/FreeBSD script removes `/etc/patchmon/` unconditionally. The Windows script only removes `C:\ProgramData\PatchMon\` when `-RemoveConfig` (or `-RemoveAll`) is passed. Re-run the Windows script with `-RemoveAll -Force`.

Backups persist after removal

Backup files (config, credentials, binary, logs) are **preserved by default** on Linux as a safety net. They live under:

- `/etc/patchmon/credentials.yml.backup.*` (removed with `/etc/patchmon/` directory)
- `/etc/patchmon/config.yml.backup.*` (removed with `/etc/patchmon/` directory)

`/usr/local/bin/patchmon-agent.backup.*` - **not** removed unless `REMOVE_BACKUPS=1`

`/etc/patchmon/logs/patchmon-agent.log.old.*` (removed with `/etc/patchmon/` directory)

Pass `REMOVE_BACKUPS=1` when invoking the removal script, or delete them manually:

```
sudo rm -f /usr/local/bin/patchmon-agent.backup.*
```

Host still appears as "Connected" briefly after uninstall

The WebSocket status can take up to ~60 seconds to reflect the disconnect. If you delete the host record in the UI while the agent is still running, the agent will immediately see `401` on its next ping and the UI will update. The residual "Connected" display is cosmetic and self-corrects.

See Also

Managing the PatchMon Agent: especially the "Agent Removal" section for a condensed reference.

Installing the PatchMon Agent: how to re-enrol a host after removal.

Agent Configuration Reference (config.yml): what's in the config files that get deleted.

Agent Troubleshooting: quick decision tree for agent-side problems.

Chapter 11: Agent config.yml Reference

Overview

The PatchMon agent is configured through a YAML file. On Linux the default path is `/etc/patchmon/config.yml`. On Windows it is `C:\ProgramData\PatchMon\config.yml`. This file controls how the agent communicates with the PatchMon server, where logs are stored, which integrations are active, and other runtime behaviour. A separate credentials file stores the host's API authentication details (`/etc/patchmon/credentials.yml` on Linux, `C:\ProgramData\PatchMon\credentials.yml` on Windows).

On Linux, these files are owned by root and set to `600` permissions (read/write by owner only) to protect sensitive information.

File Locations

File	Default Path	Purpose
Configuration	Linux: `/etc/patchmon/config.yml` Windows: `C:\ProgramData\PatchMon\config.yml`	Agent settings, server URL, integrations
Credentials	Linux: `/etc/patchmon/credentials.yml` Windows: `C:\ProgramData\PatchMon\credentials.yml`	API ID and API Key for host authentication
Log File	Linux: `/etc/patchmon/logs/patchmon-agent.log` Windows: `C:\ProgramData\PatchMon\patchmon-agent.log`	Agent log output
Cron File	`/etc/cron.d/patchmon-agent`	Scheduled reporting (fallback for non-systemd systems)

Full Configuration Reference

Below is a complete `config.yml` with all available parameters, their defaults, and descriptions:

```
# PatchMon Agent Configuration
# Location: /etc/patchmon/config.yml

# ─── Server Connection ──────────────────────────────────────────
# The URL of the PatchMon server this agent reports to.
# Required. Must start with http:// or https://
patchmon_server: "https://patchmon.example.com"

# API version to use when communicating with the server.
# Default: "v1". Do not change unless instructed.
api_version: "v1"

# ─── File Paths ─────────────────────────────────────────────────
# Path to the credentials file containing api_id and api_key.
# Default: "/etc/patchmon/credentials.yml"
credentials_file: "/etc/patchmon/credentials.yml"

# Path to the agent log file. Logs are rotated automatically
# (max 10 MB per file, 5 backups, 14-day retention, compressed).
# Default: "/etc/patchmon/logs/patchmon-agent.log"
log_file: "/etc/patchmon/logs/patchmon-agent.log"

# ─── Logging ────────────────────────────────────────────────────
# Log verbosity level.
# Options: "debug", "info", "warn", "error"
# Default: "info"
log_level: "info"

# ─── SSL / TLS ──────────────────────────────────────────────────
# Skip SSL certificate verification when connecting to the server.
# Set to true only if using self-signed certificates.
# Default: false
skip_ssl_verify: false

# ─── Reporting Schedule ─────────────────────────────────────────
# How often (in minutes) the agent sends a full report to the server.
# This value is synced from the server on startup. If the server has
# a different value, the agent updates config.yml automatically.
# Default: 60
update_interval: 60

# Report offset (in seconds). Automatically calculated from the host's
# api_id to stagger reporting across hosts and avoid thundering-herd.
# You should not need to set this manually. The agent calculates and
# persists it automatically.
# Default: 0 (auto-calculated on first run)
report_offset: 0

# ─── Integrations ───────────────────────────────────────────────
# Integration toggles control optional agent features.
# Most integrations can be toggled from the PatchMon UI and the server
```

```
# will push the change to the agent via WebSocket. The agent then
# updates config.yml and restarts the relevant service.
#
# EXCEPTION: ssh-proxy-enabled and rdp-proxy-enabled CANNOT be pushed from the
server.
# It must be manually set in this file (see below).
integrations:
  # Docker integration: monitors containers, images, volumes, networks.
  # Can be toggled from the PatchMon UI (Settings → Integrations).
  # Default: false
  docker: false

  # Compliance integration: OpenSCAP and Docker Bench security scanning.
  #   enabled: false | "on-demand" | true
  #     false       - Disabled. No scans run.
  #     "on-demand"  - Scans only run when triggered from the PatchMon UI.
  #     true        - Enabled with automatic scheduled scans every report cycle.
  #   openscap_enabled: enable/disable OpenSCAP scanning (default: true)
  #   docker_bench_enabled: enable/disable Docker Bench scanning (default: false)
  # Can be toggled from the PatchMon UI.
  compliance:
    enabled: "on-demand"
    openscap_enabled: true
    docker_bench_enabled: false

  # SSH Proxy: allows browser-based SSH sessions through the agent.
  #     SECURITY: This setting can ONLY be enabled by manually editing
  #     this file. It cannot be pushed from the server to the agent.
  #     This is intentional. Enabling remote shell access should require
  #     deliberate action by someone with root access on the host.
  # Default: false
  ssh-proxy-enabled: false

  # RDP Proxy: allows browser-based RDP sessions through the agent.
  #     SECURITY: Same as SSH proxy. Requires manual configuration.
  #     Cannot be pushed from the server. Requires guacd sidecar on
  #     the server and RDP enabled on the Windows host.
  # Default: false
  rdp-proxy-enabled: false
```

Parameters In Detail

`patchmon_server`

Type	String (URL)
Required	Yes
Default	None (required)
Example	`https://patchmon.example.com`

The full URL of the PatchMon server. Must include the protocol (`http://` or `https://`). Do not include a trailing slash or path.

`api_version`

Type	String
Required	No
Default	`v1`

The API version string appended to API calls. Leave as `v1` unless directed otherwise by PatchMon documentation or release notes.

`credentials_file`

Type	String (file path)
Required	No
Default	`/etc/patchmon/credentials.yml`

Path to the YAML file containing the host's `api_id` and `api_key`. The credentials file has this structure:

```
api_id: "patchmon_abc123def456"
api_key: "your_api_key_here"
```

`log_file`

Type	String (file path)
Required	No
Default	`/etc/patchmon/logs/patchmon-agent.log`

Path to the agent's log file. The directory is created automatically if it does not exist. Logs are rotated using the following policy:

- **Max file size**: 10 MB
- **Max backups**: 5 rotated files
- **Max age**: 14 days
- **Compression**: Enabled (gzip)

`log_level`

Type	String
Required	No
Default	`info`
Options	`debug`, `info`, `warn`, `error`

Controls the verbosity of agent logging. Use `debug` for troubleshooting: it includes detailed execution flow and extra diagnostics. Enable it briefly and disable it again once you have the logs you need. Can also be overridden at runtime with the `--log-level` CLI flag.

`skip_ssl_verify`

Type	Boolean
Required	No
Default	`false`

When `true`, the agent skips TLS certificate verification when connecting to the PatchMon server. Use this only for internal or testing environments with self-signed certificates. **Not recommended for production.**

`update_interval`

Type	Integer (minutes)
Required	No
Default	`60`

How frequently the agent sends a full system report (installed packages, updates, etc.) to the server. This value is **synced from the server**: if you change the global or per-host reporting interval in the PatchMon UI, the agent will update this value in `config.yml` automatically on its

next startup or when it receives a settings update via WebSocket.

If the value is `0` or negative, the agent falls back to the default of 60 minutes.

`report_offset`

Type	Integer (seconds)
Required	No
Default	`0` (auto-calculated)

A stagger offset calculated from the host's `api_id` and the current `update_interval`. This ensures agents across your fleet do not all report at the same moment (avoiding a thundering-herd problem on the server).

You should not set this manually. The agent calculates it on first run and saves it. If the `update_interval` changes, the offset is recalculated automatically.

`integrations`

A map of integration names to their enabled/disabled state. See the Integrations section below for details on each.

Integrations

Docker (`docker`)

Type	Boolean
Default	`false`
Server-pushable	Yes

When enabled, the agent monitors Docker containers, images, volumes, and networks on the host. It sends real-time container status events and periodic inventory snapshots to the PatchMon server.

Requirements: Docker must be installed and the Docker socket must be accessible.

Toggle from UI: Go to a host's detail page, then Integrations tab, and toggle Docker on or off. The server pushes the change to the agent via WebSocket, the agent updates `config.yml`, and the service restarts automatically.

Compliance (`compliance`)

Type	Boolean or String
Default	`"on-demand"`
Server-pushable	Yes
Valid values	`false`, `"on-demand"`, `true`

Controls OpenSCAP and Docker Bench security compliance scanning.

Value	Behaviour
`false`	Compliance scanning is fully disabled. No scans run.
`"on-demand"`	Scans only run when manually triggered from the PatchMon UI. Tools are installed but no automatic scheduled scans occur.
`true`	Fully enabled. Scans run automatically on every report cycle in addition to being available on-demand.

When first enabled, the agent automatically installs the required compliance tools (OpenSCAP, SSG content packages, Docker Bench image if Docker is also enabled).

SSH Proxy (`ssh-proxy-enabled`)

Type	Boolean
Default	`false`
Server-pushable	No (manual edit required)

Enables browser-based SSH terminal sessions proxied through the PatchMon agent. When a user opens the SSH terminal in the PatchMon UI, the server sends the SSH connection request to the agent via WebSocket, and the agent establishes a local SSH connection on behalf of the user.

Why SSH Proxy Requires Manual Configuration
This is a deliberate security design decision. Enabling SSH proxy allows remote shell access to the host through the PatchMon agent. Unlike Docker or compliance integrations, this has direct security implications:

- It opens an SSH connection path through the agent.
- It could be exploited if a PatchMon server or user account were compromised.
- The host administrator should make an informed, deliberate choice to enable it.

For these reasons, `ssh-proxy-enabled` **cannot be toggled from the PatchMon UI or pushed from the server**. If the server attempts to initiate an SSH proxy session while this is disabled, the agent rejects the request and returns an error message explaining how to enable it.

How to Enable SSH Proxy

SSH into the host where the PatchMon agent is installed.

Open the config file:

```
sudo nano /etc/patchmon/config.yml
```

Find the `integrations` section and change `ssh-proxy-enabled` to `true`:

```
integrations:
  docker: false
  compliance:
    enabled: "on-demand"
    openscap_enabled: true
    docker_bench_enabled: false
  ssh-proxy-enabled: true    # ← Change from false to true
```

Save the file and restart the agent:

```
# Systemd
sudo systemctl restart patchmon-agent.service

# OpenRC (Alpine)
sudo rc-service patchmon-agent restart
```

The SSH terminal feature is now available for this host in the PatchMon UI.

How to Disable SSH Proxy

Set `ssh-proxy-enabled` back to `false` in `config.yml` and restart the agent service. Existing SSH sessions will be terminated.

RDP Proxy (`rdp-proxy-enabled`)

Type	Boolean
Default	`false`
Server-pushable	No (manual edit required)

Enables browser-based RDP (Remote Desktop Protocol) sessions proxied through the PatchMon agent. When a user opens the RDP tab for a Windows host in the PatchMon UI, the server sends the RDP connection request to the agent via WebSocket, and the agent establishes a local RDP connection (default: `localhost:3389`) on behalf of the user via `guacd` (Apache Guacamole) running on the PatchMon server.

Why RDP Proxy Requires Manual Configuration

Same security rationale as SSH proxy. Enabling RDP proxy allows remote desktop access to the host through the PatchMon agent. This requires a deliberate decision by the host administrator:

- It opens an RDP connection path through the agent to port 3389.
- It could be exploited if a PatchMon server or user account were compromised.
- The Windows host must have RDP enabled and `guacd` must be available on the PatchMon server.

For these reasons, `rdp-proxy-enabled` **cannot be toggled from the PatchMon UI or pushed from the server**. If the server attempts to initiate an RDP proxy session while this is disabled, the agent rejects the request and returns an error message explaining how to enable it.

Prerequisites

- The PatchMon **server** must have `guacd` available (the default Docker Compose stack includes `guacamole/guacd:1.5.5` as a sidecar).
- The **Windows host** must have Remote Desktop enabled.
- The PatchMon agent must be installed on the Windows host.
- PatchMon only needs RDP listening on `localhost:3389` on the Windows host. You do not need to expose RDP publicly.
- If the browser reaches the host but the session still fails, retry with explicit Windows credentials first. The PatchMon UI distinguishes port-unreachable, auth, security-negotiation, and generic post-connect setup failures when guacd provides an explicit upstream message.

How to Enable RDP Proxy

- Connect to the host where the PatchMon agent is installed.
- Open the config file on the host:

```
# Windows (PowerShell as Administrator)
notepad "C:\ProgramData\PatchMon\config.yml"
```

- Find the `integrations` section and change `rdp-proxy-enabled` to `true`:

```
integrations:
  docker: false
  compliance:
    enabled: "on-demand"
    openscap_enabled: true
    docker_bench_enabled: false
  ssh-proxy-enabled: false
  rdp-proxy-enabled: true    # ← Change from false to true
```

Save the file and restart the agent:

```
# Systemd
sudo systemctl restart patchmon-agent.service

# Windows (PowerShell as Administrator)
Restart-Service -Name PatchMonAgent
```

The RDP tab is now available for this host in the PatchMon UI.

How to Disable RDP Proxy

Set `rdp-proxy-enabled` back to `false` in `config.yml` and restart the agent service. Existing RDP sessions will be terminated.

How `config.yml` Is Generated

Initial Generation (Installation)

The `config.yml` file is created during agent installation by the `patchmon_install.sh` script. The installer generates a fresh config with:

- `patchmon_server` set to the server URL used during installation
- `skip_ssl_verify` set based on whether `-k` curl flags were used
- All integrations defaulted to `false` (Docker, SSH proxy, RDP proxy) or `"on-demand"` (compliance)
- Standard file paths for credentials and logs

```
# What the installer generates:
cat > /etc/patchmon/config.yml << EOF
# PatchMon Agent Configuration
# Generated on $(date)
patchmon_server: "https://patchmon.example.com"
api_version: "v1"
credentials_file: "/etc/patchmon/credentials.yml"
log_file: "/etc/patchmon/logs/patchmon-agent.log"
log_level: "info"
skip_ssl_verify: false
integrations:
  docker: false
  compliance:
    enabled: "on-demand"
    openscap_enabled: true
    docker_bench_enabled: false
  ssh-proxy-enabled: false
  rdp-proxy-enabled: false
EOF

chmod 600 /etc/patchmon/config.yml
```

Reinstallation Behaviour

If the agent is reinstalled on a host that already has a working configuration:

- The installer **checks if the existing configuration is valid** by running `patchmon-agent ping`.
- If the ping succeeds, the installer **exits without overwriting**. The existing configuration is preserved.
- If the ping fails (or the binary is missing), the installer:
  - Creates a timestamped backup: `config.yml.backup.YYYYMMDD_HHMMSS`
  - Keeps only the last 3 backups (older ones are deleted)
  - Writes a fresh `config.yml`

A reinstall on a healthy agent is safe and will not destroy your configuration.

How `config.yml` Is Regenerated / Updated at Runtime

The agent updates `config.yml` automatically in several scenarios. These are in-place updates: the agent reads the file, modifies the relevant field, and writes it back. Your other settings (including `ssh-proxy-enabled`) are preserved.

Server-Driven Updates

Trigger	What Changes	How
Agent startup	`update_interval`, `report_offset`	Agent fetches the current interval from the server. If it differs from config, the agent updates config.yml.
Agent startup	`integrations.docker`, `integrations.compliance`	Agent fetches integration status from the server. If it differs from config, the agent updates config.yml.
WebSocket: `settings_update`	`update_interval`, `report_offset`	Server pushes a new interval. Agent saves it and recalculates the report offset.
WebSocket: `integration_toggle`	`integrations.*` (except SSH/RDP proxy)	Server pushes a toggle for Docker or compliance. Agent saves the change and restarts the relevant service.

Agent-Calculated Updates

Trigger	What Changes	How
First run	`report_offset`	Calculated from `api_id` hash and `update_interval` to stagger reports.
Interval change	`report_offset`	Recalculated whenever `update_interval` changes.
CLI: `config set-api`	`patchmon_server`, credentials	Running `patchmon-agent config set-api` overwrites the server URL and saves new credentials.

What Is Never Changed Automatically

Parameter	Why
`ssh-proxy-enabled`	Security: requires manual host-level action
`rdp-proxy-enabled`	Security: requires manual host-level action
`log_level`	Only changed by manual edit or `--log-level` CLI flag
`log_file`	Only changed by manual edit
`credentials_file`	Only changed by manual edit or `config set-api`
`skip_ssl_verify`	Only changed by manual edit

Important: How SaveConfig Works

When the agent calls `SaveConfig()` internally, it writes **all parameters** back to the file. This means:

- Your `ssh-proxy-enabled` and `rdp-proxy-enabled` settings are **preserved** across server-driven updates.
- New integrations added in agent updates are **automatically added** to the file with their defaults (you will see them appear after an agent update).
- The file format may be slightly reorganised by the YAML serialiser (key ordering may change), but all values are preserved.

CLI Configuration Commands

The agent provides CLI commands for configuration management:

View Current Configuration

```
sudo patchmon-agent config show
```

Output:

```
Configuration:
  Server: https://patchmon.example.com
  Agent Version: 1.4.0
  Config File: /etc/patchmon/config.yml
  Credentials File: /etc/patchmon/credentials.yml
  Log File: /etc/patchmon/logs/patchmon-agent.log
  Log Level: info

Credentials:
  API ID: patchmon_abc123def456
  API Key: Set ✓
```

Set API Credentials

```
sudo patchmon-agent config set-api <API_ID> <API_KEY> <SERVER_URL>
```

Avoid pasting a real API key into a shell with persistent history or session recording. If your environment records command lines, use a temporary shell with history disabled, or update `credentials.yml` directly.

This command:

- Validates the server URL format

Saves the server URL to `config.yml`

Saves the credentials to `credentials.yml`

Tests connectivity with a ping to the server

Reports success or failure

Custom Config File Path

All commands support a `--config` flag to use an alternative config file:

```
sudo patchmon-agent --config /path/to/custom/config.yml serve
```

Credentials File (`credentials.yml`)

The credentials file is separate from the config file for security isolation. It contains:

```
api_id: "patchmon_abc123def456"
api_key: "your_api_key_here"
```

Permissions: `600` (root read/write only)

Written using atomic rename: The agent writes to a temp file first, then atomically renames it. This prevents partial writes or race conditions.

Never contains the hashed key: The plain-text API key is stored here; the server stores only the bcrypt hash.

Troubleshooting

Config File Missing

If `/etc/patchmon/config.yml` does not exist, the agent uses built-in defaults. This means it will not know which server to connect to. Reinstall the agent or create the file manually.

Config File Permissions

```
# Check permissions (should be 600, owned by root)
ls -la /etc/patchmon/config.yml

# Fix if needed
sudo chmod 600 /etc/patchmon/config.yml
sudo chown root:root /etc/patchmon/config.yml
```

SSH Proxy Not Working

If the SSH terminal in the PatchMon UI shows an error like:

> *SSH proxy is not enabled. To enable SSH proxy, edit the file /etc/patchmon/config.yml...*

This means `ssh-proxy-enabled` is set to `false` (the default). Follow the How to Enable SSH Proxy instructions above.

Config Gets Overwritten

If you notice settings being changed unexpectedly, check:

Server sync: The `update_interval` and integration toggles (Docker, compliance) are synced from the server on startup and via WebSocket. Changes made in the PatchMon UI will override local values for these fields.

Agent updates: After an agent update, new integration keys may appear in the file with default values.

Reinstallation: A reinstall only overwrites config if the existing ping test fails.

Your `ssh-proxy-enabled`, `rdp-proxy-enabled`, `log_level`, `skip_ssl_verify`, and file path settings are **never overwritten** by server sync.

Viewing Debug Logs

```
# Temporarily enable debug logging
sudo patchmon-agent --log-level debug serve

# Or set permanently in config.yml
sudo nano /etc/patchmon/config.yml
# Change: log_level: "debug"
# Then restart the service
sudo systemctl restart patchmon-agent.service
```

Only enable `debug` briefly for troubleshooting. Avoid it during active SSH or RDP sessions, since remote-access traffic is more sensitive than normal agent telemetry. Switch back to `info` once you have captured what you need.

Example Configurations

Minimal Configuration

```
patchmon_server: "https://patchmon.example.com"
```

All other values use defaults. The agent will function with just the server URL (and valid credentials in `credentials.yml`).

Full Configuration with SSH and RDP Proxy Enabled

```
patchmon_server: "https://patchmon.internal.company.com"
api_version: "v1"
credentials_file: "/etc/patchmon/credentials.yml"
log_file: "/etc/patchmon/logs/patchmon-agent.log"
log_level: "info"
skip_ssl_verify: false
update_interval: 30
report_offset: 847
integrations:
  docker: true
  compliance:
    enabled: "on-demand"
    openscap_enabled: true
    docker_bench_enabled: false
  ssh-proxy-enabled: true
  rdp-proxy-enabled: true
```

Self-Signed SSL with Debug Logging

```
patchmon_server: "https://patchmon.lab.local"
api_version: "v1"
credentials_file: "/etc/patchmon/credentials.yml"
log_file: "/etc/patchmon/logs/patchmon-agent.log"
log_level: "debug"
skip_ssl_verify: true
update_interval: 60
integrations:
  docker: false
  compliance:
    enabled: false
    openscap_enabled: true
    docker_bench_enabled: false
  ssh-proxy-enabled: false
  rdp-proxy-enabled: false
```

Chapter 12: Server Troubleshooting

Overview

This page covers diagnosing and fixing PatchMon 2.0 **server** problems on a Docker Compose deployment. It covers container startup failures, database and Redis issues, CORS, WebSocket, reverse-proxy, and admin recovery scenarios.

If the problem is on the **agent** side (won't start, can't reach the server, credentials file missing), jump to Agent Troubleshooting instead.

Reference Architecture

A stock `docker-compose.yml` deployment runs four containers on the `patchmon-internal` bridge network:

Service	Image	Port (exposed)	Depends on
`server`	`ghcr.io/patchmon/patchmon-server:latest`	`3000:3000`	`database`, `redis`, `guacd`
`database`	`postgres:17-alpine`	not exposed	n/a
`redis`	`redis:7-alpine`	not exposed	n/a
`guacd`	`guacamole/guacd:1.5.5`	not exposed	n/a

The `server` container embeds the Go HTTP server, the frontend, the queue worker, and the migration runner. No separate migration job is needed. In front of it you typically run Nginx / Traefik / Caddy / Cloudflare that terminates TLS and forwards to `server:3000`.

Gathering Diagnostic Info

Before attempting any fix, capture the current state of the stack. These commands are safe and non-destructive.

```
# In the directory where your docker-compose.yml lives

# 1. Show container state (Up / Restarting / Exit)
docker compose ps

# 2. Server logs (last 200 lines)
docker compose logs --tail 200 server

# 3. Everything (server + database + redis + guacd)
docker compose logs --tail 200 --timestamps

# 4. Follow logs live
docker compose logs -f server

# 5. Health check (from the Docker host)
curl http://localhost:3000/health
# expected: 200 OK, body "healthy"

# 6. Health check as JSON
curl -H 'Accept: application/json' http://localhost:3000/health
# expected: {"status":"healthy","database":"healthy","redis":"healthy"}

# 7. Resource usage
docker stats --no-stream
```

The health endpoint at `/health` is **public and unauthenticated** and reports the status of both the Postgres connection and the Redis connection. A `503 Service Unavailable` from `/health` means at least one dependency is down.

Check the values of your `.env` without leaking secrets:

```
# Show keys only (no values) -- safe to share in a bug report
grep -v '^#' .env | grep '=' | cut -d= -f1 | sort
```

Check what the server sees at runtime (Settings UI → Server tab):

1. Log into PatchMon as a superadmin.
2. Navigate to **Settings → Server**.
3. The "Effective value" column shows the resolved value (env → database → default) and "Source" tells you which layer won.
4. "Conflict" flags any setting defined in both `.env` and the database. The env always wins, but it is worth resolving.

1. Container Won't Start / Crashes on Boot

Symptoms

`docker compose ps` shows `server` in state `Restarting` or `Exited (1)`.

`docker compose logs server` shows an error immediately and the container loops.

Diagnose

```
docker compose logs --tail 100 server
```

Look at the first 30 lines. The crash cause is almost always logged right there.

Common Causes and Fixes

Log line	Cause	Fix
`config: DATABASE_URL is required`	`DATABASE_URL` is empty or missing from `.env`	Set `DATABASE_URL=postgresql://user:pass@database:5432/patch sslmode=disable` in `.env` and `docker compose up -d`.
`config: JWT_SECRET is required`	`JWT_SECRET` is missing	Generate one: `openssl rand -base64 48`. Add to `.env`.
`migrations failed: ...`	Database migration error at boot	See **Database Migration Errors** below.
`database: ...: connect: connection refused`	Postgres is not yet healthy or `DATABASE_URL` hostname is wrong	`docker compose ps database` and ensure it shows `healthy`. Th hostname in `DATABASE_URL` must match the service name in `dock compose.yml` (`database`, not `localhost`).
`redis: ... NOAUTH Authentication required`	Server is connecting to Redis without a password but Redis has one set	Set `REDIS_PASSWORD` to the same value as the one passed to `red server --requirepass` in `docker-compose.yml`. Both are read f the same `.env`.
`encryption init failed`	Bootstrap tokens / OIDC secrets will not work. Set at least one of `DATABASE_URL`, `SESSION_SECRET`, or `AI_ENCRYPTION_KEY`.	Set `SESSION_SECRET` in `.env` (32+ characters, random).

Escape Hatch: Start a Shell in the Server Image

If the container crashes too fast to inspect, run it with an override command:

```
docker compose run --rm --entrypoint /bin/sh server
```

From the resulting shell you can `env | grep -E 'DATABASE_URL|REDIS|JWT'` and test connectivity with `nc -zv database 5432` and `redis-cli -h redis -a "$REDIS_PASSWORD" ping`.

2. Database Migration Errors at Boot

Symptoms

```
[fatal] migrations failed: Dirty database version N. Fix and force version.
```

or

```
[fatal] migrations failed: migration file XXXX is corrupted
```

Background

PatchMon uses `golang-migrate` with embedded SQL files. On every server start, the server runs pending migrations before opening its HTTP listener. A **dirty** state means a previous migration started and crashed mid-way. The `schema_migrations` table records the version but the `dirty` column is `true`.

Fix: Stuck on Dirty

PatchMon ships a standalone `migrate` binary alongside the server binary. Use it to inspect and unstick the migration state.

```
# 1. Stop the server so nothing is writing
docker compose stop server

# 2. Open a shell inside a fresh server container (database keeps running)
docker compose run --rm --entrypoint /bin/sh server

# Inside the container:
migrate version
# prints: Version: 42 (dirty: true)

# 3. Review what migration 42 did -- read the SQL file if you have the source
#    (in the image, migrations are embedded -- you may need to check the repo)

# 4. Manually fix the partial change in Postgres if needed, then force-reset:
migrate force 42          # tells migrate the schema is at 42, not dirty
migrate up                # re-run from 42 onwards (idempotent if SQL is safe)

exit
docker compose up -d server
```

> ***Only use*** `migrate force` *after you've verified the schema is actually consistent with version N. Forcing onto an inconsistent schema hides the problem until the next migration.*

Fix: Run Migrations Manually

The server runs migrations automatically at startup, so you normally never need to run them by hand. If you do (e.g. scripted rollout, debugging):

```
# Up (apply all pending)
docker compose run --rm --entrypoint migrate server up

# Down one step (rollback the most recent migration)
docker compose run --rm --entrypoint migrate server down

# Show current version
docker compose run --rm --entrypoint migrate server version
```

3. "CORS Error" in the Browser

Symptoms

- Browser DevTools network panel shows `OPTIONS /api/v1/...` returning 403/404 with `No 'Access-Control-Allow-Origin' header`.
- Login or wizard requests fail silently.

Cause

`CORS_ORIGIN` does not match the URL the user's browser is hitting. `CORS_ORIGIN` accepts a single origin or a comma-separated list of origins (no spaces between entries), and each entry must be the exact origin the browser uses (protocol + host + port, no path, no trailing slash). Common mismatches:

- `.env` has `CORS_ORIGIN=http://localhost:3000` but users access PatchMon at `https://patchmon.example.com`.
- `CORS_ORIGIN=https://patchmon.example.com` but users access via `https://patchmon.example.com:8443`.
- Trailing slash in `CORS_ORIGIN` (`https://patchmon.example.com/`). Strip the slash.
- PatchMon is reached from more than one URL (e.g. an external domain and an internal LAN address) but only one is listed.

Fix

Set `CORS_ORIGIN` to the **exact origin** the browser uses (protocol + host + port, no path, no trailing slash):

```
# .env
CORS_ORIGIN=https://patchmon.example.com
```

For multiple allowed origins (e.g. staging and production share a database during migration, or users reach PatchMon on both an external and internal URL), comma-separate them with no spaces between entries:

```
CORS_ORIGIN=https://patchmon.example.com,https://staging.patchmon.example.com
```

Then restart the server:

```
docker compose restart server
```

> ***Requires a server restart.*** *The Settings UI flags* `CORS_ORIGIN` *as "Requires a server restart to take effect".*

Alternative: Settings UI

You can also set `CORS_ORIGIN` in **Settings → Server → CORS_ORIGIN** via the UI. If both the env var and the UI value are set, the env wins and the Settings UI shows a "Conflict" flag. Pick one source of truth.

4. Agent Can't Connect Over WebSocket

Symptoms

- Agents check in via HTTP reports (host turns "Active") but show **Offline** in the Hosts list.
- Agent log shows repeated `websocket: bad handshake` or reconnection loops.
- The "Waiting for Connection" screen after enrolment gets past **Waiting** to **Connected** slowly or never.

Cause

The agent opens a WebSocket at `GET /api/v1/agents/ws` with `Upgrade: websocket` / `Connection: Upgrade`. If your reverse proxy is **not forwarding those headers**, the upgrade handshake fails and the connection falls back to HTTP, which the agent then drops.

Fix: Nginx

```
location / {
    proxy_pass http://127.0.0.1:3000;
    proxy_http_version 1.1;

    # Required for WebSocket upgrade
    proxy_set_header Upgrade    $http_upgrade;
    proxy_set_header Connection $connection_upgrade;

    proxy_set_header Host              $host;
    proxy_set_header X-Real-IP         $remote_addr;
    proxy_set_header X-Forwarded-For   $proxy_add_x_forwarded_for;
    proxy_set_header X-Forwarded-Proto $scheme;

    # Don't time out long-lived connections (WS, SSE, patching streams)
    proxy_read_timeout 86400s;
    proxy_send_timeout 86400s;
}

# Top of the config:
map $http_upgrade $connection_upgrade {
    default upgrade;
    ''      close;
}
```

Fix: Traefik

Traefik forwards the required headers by default. You normally do not need extra config. If you have a custom `headers` middleware, make sure it does not strip `Upgrade` / `Connection`.

Fix: Caddy

```
patchmon.example.com {
    reverse_proxy 127.0.0.1:3000
    # Caddy auto-handles WebSocket upgrades -- nothing extra needed.
}
```

Fix: Cloudflare

Cloudflare's free tier supports WebSocket but check the dashboard: **Network → WebSockets → On**.

Verify the Fix

From the server host (not the agent host), this should return `101 Switching Protocols`:

```
curl -i -N \
  -H "Connection: Upgrade" \
  -H "Upgrade: websocket" \
  -H "Sec-WebSocket-Version: 13" \
  -H "Sec-WebSocket-Key: $(openssl rand -base64 16)" \
  https://patchmon.example.com/api/v1/agents/ws?apiId=dummy
```

If you see `101`, the proxy is forwarding the handshake. A `4xx` back from `/api/v1/agents/ws` is fine here. The agent uses real credentials; the point is that the proxy did not interfere with the upgrade attempt.

`X-Forwarded-Proto` Matters

Set `X-Forwarded-Proto $scheme` (Nginx) or equivalent in other proxies. PatchMon uses the `TRUST_PROXY=true` env setting to read this header when present. Without it, the server may think it is serving over HTTP and emit `Secure` cookies that the browser then refuses to send.

5. 502 Bad Gateway from the Reverse Proxy

Symptoms

Browser shows `502 Bad Gateway` or `503 Service Unavailable`.

Nginx log shows `upstream prematurely closed connection` or `connect() failed (111: Connection refused)`.

Diagnose

```
# 1. Is the server container up?
docker compose ps server

# 2. Is it healthy?
curl http://localhost:3000/health

# 3. Can the reverse proxy reach the container?
#    From the reverse-proxy host:
curl -v http://<server-host>:3000/health
```

Common Fixes

Container not running: `docker compose up -d server`.

Health check failing: follow **1. Container Won't Start** above.

Wrong upstream port: the default is `3000`. If you changed `PORT` in `.env`, update the reverse-proxy config to match.

Firewall blocking 3000: if the reverse proxy is on a different host, open TCP/3000 between them, or bind the container on a UNIX socket / private interface.

Healthcheck grace too short: on slow storage, Postgres may take 20-60 s to become healthy on first boot. Increase `start_period` on the `database` and `redis` healthcheck, or simply wait.

6. Redis Connection Refused / NOAUTH

Symptoms

```
redis: dial tcp redis:6379: connect: connection refused
```

or

```
redis: NOAUTH Authentication required.
```

Cause

Connection refused: the `redis` container is not running or crashed.

NOAUTH: the server is connecting with no password (or the wrong one) while Redis requires one.

Fix

The `docker-compose.yml` starts Redis with:

```
redis:
  command: redis-server --requirepass ${REDIS_PASSWORD}
```

Both the server and Redis read `REDIS_PASSWORD` from the same `.env`. If you change it, **restart both**:

```
docker compose up -d redis server
```

Verify from inside the network:

```
docker compose exec redis redis-cli -a "$REDIS_PASSWORD" ping
# expected: PONG
```

Check the server sees the right password:

```
docker compose exec server env | grep REDIS_
```

> *If you set `REDIS_PASSWORD` with special shell characters (`$`, `!`, space), Docker Compose may mis-quote them. Stick to alphanumerics + `-_` for the password or wrap it in single quotes in `.env`.*

7. Upload / Body-Limit Errors

Symptoms

Agent reports fail with HTTP 413 `Request Entity Too Large`.

Web UI pages showing very large tables (huge Docker inventory, thousands of packages) return 413 on save.

Cause

PatchMon caps JSON request bodies to protect against memory exhaustion. Two separate limits apply:

Env var	Default	What it caps
`JSON_BODY_LIMIT`	`5` (MB)	Every non-agent JSON endpoint (UI API, settings, etc.).
`AGENT_UPDATE_BODY_LIMIT`	`2` (MB)	`POST /api/v1/hosts/update` only: the agent report payload.

Both are integers **in megabytes**. A host with thousands of packages + Docker + compliance data can breach the 2 MB default.

Fix

Raise the limits in `.env`:

```
JSON_BODY_LIMIT=10
AGENT_UPDATE_BODY_LIMIT=8
```

Restart:

```
docker compose restart server
```

If you are fronting PatchMon with Nginx, also raise its limit, otherwise Nginx 413s before the server sees the body:

```
client_max_body_size 16m;
```

8. Slow Queries / Database Pressure

Symptoms

API responses slow down as the number of hosts grows.

Postgres container CPU usage sustained near 100%.

Server logs show `context deadline exceeded` on database calls.

Diagnose

```
# Postgres activity
docker compose exec database psql -U "$POSTGRES_USER" -d "$POSTGRES_DB" \
  -c "SELECT pid, state, wait_event, query_start, substr(query, 1, 120) FROM
pg_stat_activity WHERE state ≠ 'idle';"

# Check pool stats (from the server's perspective, via health+metrics if enabled)
curl -H 'Accept: application/json' http://localhost:3000/health
```

Common Fixes

Tune the pool and related timeouts in `.env`:

Env var	Default	When to tune
`DB_CONNECTION_LIMIT`	`30`	Raise to `60` - `100` for 1000+ hosts. Postgres defaults to 100; do not exceed without raising `max_connections` in Postgres.
`DB_POOL_TIMEOUT`	`20` (s)	Raise if you see "timeout acquiring connection" under burst load.
`DB_IDLE_TIMEOUT`	`300` (s)	How long idle connections stay in the pool.
`DB_MAX_LIFETIME`	`1800` (s)	Hard cap per connection. Useful with load balancers that reset idle TCP.

For the full env-var reference, see the Configuration section in the main admin docs.

If the problem persists, enable the request logger (`ENABLE_LOGGING=true` , `LOG_LEVEL=debug`) for a short window to catch which endpoint is hot.

9. Admin User Locked Out

Situation

You are the only superadmin and you:

Forgot your password and disabled "Forgot password" flows, or

Lost access to your TFA device and backup codes, or

Enabled `OIDC_DISABLE_LOCAL_AUTH=true` and your IdP is now broken.

No Built-in CLI Reset

PatchMon does **not** ship a CLI subcommand for resetting admin passwords or clearing TFA. The server binary is a single daemon (`patchmon-server`) with no subcommands, and the separate `migrate` binary only supports `up` , `down` , `force V` , and `version` . The intentional design is that all user management goes through the web UI.

The workaround is direct database modification.

Workaround: Reset the Password via `psql`

Step 1: generate a bcrypt hash of the password you want to set. PatchMon stores passwords with bcrypt cost 10 (matching the legacy Node.js implementation). Any language works:

```
# With htpasswd (from apache2-utils / httpd-tools)
htpasswd -bnBC 10 "" 'new-password-here' | tr -d ':\n'
# prints something like: $2y$10$...

# Or with Python (bcrypt library)
python3 -c 'import bcrypt; print(bcrypt.hashpw(b"new-password-here",
bcrypt.gensalt(10)).decode())'
```

Note: bcrypt produces hashes starting with `$2a$10$` , `$2b$10$` , or `$2y$10$` : all three are compatible.

Step 2: connect to Postgres and update the user row:

```
docker compose exec database psql -U "$POSTGRES_USER" -d "$POSTGRES_DB"
```

```
-- See what accounts exist
SELECT id, username, email, role, is_active FROM users ORDER BY created_at;

-- Reset the password hash. Replace <bcrypt-hash> with the output from step 1.
-- Keep the single quotes and escape the hash's $ signs if your shell interprets
them.
UPDATE users
SET password_hash = '<bcrypt-hash>', updated_at = NOW()
WHERE username = 'admin';

-- Confirm
SELECT username, role, is_active, updated_at FROM users WHERE username = 'admin';

\q
```

Step 3: log in with the new password and immediately rotate the credentials through the UI (Profile → Change password) to confirm the flow works, and re-enrol TFA if needed.

Workaround: Unlock a Locked-Out Account

After too many failed logins, accounts are locked for `LOCKOUT_DURATION_MINUTES` (default 15). To clear the lock immediately:

```
UPDATE users
SET failed_login_attempts = 0, locked_until = NULL
WHERE username = 'admin';
```

Workaround: Disable TFA on an Account

```
UPDATE users
SET tfa_enabled = false, tfa_secret = NULL, tfa_backup_codes = NULL
WHERE username = 'admin';
```

The user can re-enrol TFA after logging in.

Workaround: Re-enable Local Login When OIDC is Broken

If `OIDC_DISABLE_LOCAL_AUTH=true` is blocking you, edit `.env`:

```
OIDC_DISABLE_LOCAL_AUTH=false
```

then:

```
docker compose restart server
```

Log in with local credentials, fix the OIDC config, and flip it back.

> ***Always take a database backup before running*** `UPDATE` ***statements.*** `docker compose exec database pg_dump -U "$POSTGRES_USER" "$POSTGRES_DB" > patchmon-backup-$(date +%F).sql`.

10. Quick-Reference: When to Restart What

Change	What to restart
`DATABASE_URL`, `REDIS_*`, `JWT_SECRET`, `SESSION_SECRET`	`docker compose restart server`
`CORS_ORIGIN`, `TRUST_PROXY`, `ENABLE_HSTS`	`docker compose restart server`
`JSON_BODY_LIMIT`, `AGENT_UPDATE_BODY_LIMIT`	`docker compose restart server`
`OIDC_*`	`docker compose restart server`
Postgres config change (custom `postgresql.conf`)	`docker compose restart database` (then wait for health)
Redis password change	`docker compose up -d redis server` (both)
Reverse-proxy config	Reload your reverse proxy (`nginx -s reload`, Traefik is live, etc.)
Agent binary rebuilt and placed in `AGENT_BINARIES_DIR`	Nothing. Agents pick it up on next `check-version`.

See Also

- Agent Troubleshooting: decision tree for agent-side issues.
- Managing the PatchMon Agent: CLI, service, logs, update, and removal.
- Installing the PatchMon Agent: enrolment walkthrough.
- Agent Configuration Reference (config.yml): every agent config parameter.

Chapter 13: Agent Troubleshooting

Overview

This page is a **quick-reference decision tree** for the most common PatchMon agent symptoms. Each entry points to detailed steps in Managing the PatchMon Agent: Common Troubleshooting or another relevant section. Use this page to triage, then follow the linked section for the full fix.

If the problem is on the **server** side (container crashes, database migrations, CORS, reverse proxy), go to Server Troubleshooting instead.

The 30-Second Triage

Before anything else, run these three commands on the affected host:

```
sudo patchmon-agent diagnostics    # full picture in one shot
sudo systemctl status patchmon-agent    # (or: rc-service patchmon-agent status)
sudo journalctl -u patchmon-agent -n 50 --no-pager    # (or: tail -n 50
/etc/patchmon/logs/patchmon-agent.log)
```

On Windows (elevated PowerShell):

```
patchmon-agent diagnostics
Get-Service PatchMonAgent
Get-Content 'C:\ProgramData\PatchMon\patchmon-agent.log' -Tail 50
```

The `diagnostics` output includes system info, configuration status, network reachability, credential validity, and the last 10 log lines. In most cases it tells you immediately which part is broken. Full details: Managing the PatchMon Agent: `diagnostics`.

Decision Tree

Symptom → Likely Cause → Where to Look

Symptom	Likely cause	Jump to
Host shows **Pending** in the UI, never flips to Active	Agent not running, or first report never delivered	Host shows Pending
Host shows **Offline** in the UI	WebSocket is down (service crashed or network dropped)	Host shows Offline
Agent **won't start**	Bad `config.yml`, bad credentials, port/permission issue	Agent won't start
Agent **can't reach server**: DNS failure	DNS resolution broken on host	Cannot reach server: DNS
Agent **can't reach server**: TLS / cert	CA not trusted or certificate invalid	Cannot reach server: TLS
Agent **can't reach server**: firewall	Outbound 443 blocked	Cannot reach server: firewall
Report interval **not updating**	Agent is using cached interval; server sync not yet completed	Report interval not updating
Auto-update **failing**	Hash mismatch, SSL error on download, or update loop cooldown	Auto-update failing
`credentials.yml` **missing or corrupt**	File deleted, permissions wrong, or YAML syntax broken	Credentials file missing or corrupt
Agent runs but **logs not being written**	Wrong path, permission problem, or disk full	Logs not being written
Service **keeps restarting** in a loop	Config error or unreachable server with `Restart=always`	Service keeps restarting
`Permission denied` on every command	Running without `sudo` / Administrator	Permission denied

Host Shows Pending

The host record was created but the agent has not sent its first report.

Quick checks:

```
sudo systemctl status patchmon-agent
sudo patchmon-agent ping
sudo patchmon-agent report      # force an immediate report
```

If `ping` fails, see Cannot reach server: DNS, TLS, or firewall.

If `ping` succeeds but the host stays Pending, the agent has not pushed a report yet. Run `sudo patchmon-agent report` and watch `journalctl -u patchmon-agent -f` for errors during report upload. A common cause is `413 Request Entity Too Large`. See Server Troubleshooting: Upload/Body-Limit Errors.

Full details: Managing the PatchMon Agent: Agent Shows "Pending" in PatchMon.

Host Shows Offline

The host sent at least one report in the past, but its WebSocket is currently disconnected.

Quick checks:

```
sudo systemctl is-active patchmon-agent    # should print "active"
sudo journalctl -u patchmon-agent -n 50 --no-pager
```

Common causes:

- **Service stopped or crashed**: `sudo systemctl restart patchmon-agent` and watch the logs for the underlying error.
- **Reverse proxy not forwarding WebSocket upgrade headers**: see Server Troubleshooting: Agent Can't Connect Over WebSocket.
- **NAT / load balancer timing out idle connections**: raise the proxy's idle timeout to at least 65 s. The agent sends WebSocket pings every 30 s.
- **Temporary network blip**: the agent auto-reconnects with exponential backoff. Wait 60 s and re-check.

Full details: Managing the PatchMon Agent: Agent Shows "Offline" in PatchMon.

Agent Won't Start

The service exits immediately or never stays running.

Diagnose by running the agent in the foreground with debug logging:

```
sudo systemctl stop patchmon-agent
sudo patchmon-agent serve --log-level debug
# (Ctrl+C to stop; then: sudo systemctl start patchmon-agent)
```

Common causes:

- **Broken `config.yml`**: YAML syntax error. Validate with `yq e . /etc/patchmon/config.yml` or reinstall to regenerate defaults.
- **Missing `credentials.yml`**: see Credentials file missing or corrupt.
- **Invalid `patchmon_server` URL**: must start with `http://` or `https://`, no trailing slash.

Port conflict (rare): the agent itself doesn't listen on any port, but if you've enabled the SSH-proxy integration with a static local port, check with `ss -lntp | grep patchmon`.

Wrong architecture binary: `file /usr/local/bin/patchmon-agent` and compare against `uname -m`. If they disagree, reinstall from the UI so the correct binary is fetched.

Full details: Managing the PatchMon Agent: Testing and Diagnostics and Debugging a Problem.

Cannot Reach Server: DNS

Log or `ping` output contains `no such host`, `dial tcp: lookup <hostname>`, or `server misbehaving`.

Diagnose:

```
nslookup patchmon.example.com
# or
dig +short patchmon.example.com
```

Fix:

If DNS resolution fails, fix `/etc/resolv.conf` or your internal DNS records.

Inside containers (Docker/LXC/Kubernetes): make sure the container's DNS resolver can reach your internal DNS.

If you use a `/etc/hosts` override, check the entry matches the URL in `/etc/patchmon/config.yml`.

Full details: Managing the PatchMon Agent: "Connectivity Test Failed".

Cannot Reach Server: TLS

Log contains `x509: certificate signed by unknown authority`, `tls: failed to verify certificate`, or (on Windows) `Could not establish trust relationship for the SSL/TLS secure channel`.

Preferred fix: install the CA into the system trust store:

Debian/Ubuntu: copy CA to `/usr/local/share/ca-certificates/` (must have `.crt` extension) and run `sudo update-ca-certificates`.

RHEL/Rocky/Fedora: copy CA to `/etc/pki/ca-trust/source/anchors/` and run `sudo update-ca-trust`.

Alpine: `sudo apk add ca-certificates`, then same as Debian.

Windows: import via `certlm.msc` into **Local Computer → Trusted Root Certification Authorities**.

Quick bypass (lab only): set `skip_ssl_verify: true` in `/etc/patchmon/config.yml` and restart the service, or set env var `PATCHMON_SKIP_SSL_VERIFY=true`.

> ***Do not*** *enable* `skip_ssl_verify` *in production. It disables TLS verification entirely. See the warning in Managing the PatchMon Agent: SSL Certificate Errors.*

Full details: Managing the PatchMon Agent: SSL Certificate Errors.

Cannot Reach Server: Firewall

`ping` reports "connectivity test failed" but DNS and TLS are fine. Typically `i/o timeout` or `connection refused`.

Diagnose:

```
# TCP reachability on 443
timeout 5 bash -c "</dev/tcp/patchmon.example.com/443" && echo open || echo
blocked
# or:
nc -vz patchmon.example.com 443
```

Fix:

Open outbound TCP/443 (or whatever port your reverse proxy uses) to the PatchMon server on the host's / network's egress firewall.

If the host is behind an outbound HTTP proxy, set `HTTPS_PROXY` and `HTTP_PROXY` in the agent's environment. On systemd, add them as `Environment=` lines to a drop-in at `/etc/systemd/system/patchmon-agent.service.d/proxy.conf`:

```
[Service]
Environment="HTTPS_PROXY=http://proxy.corp:3128"
Environment="HTTP_PROXY=http://proxy.corp:3128"
Environment="NO_PROXY=localhost,127.0.0.1"
```

then `sudo systemctl daemon-reload && sudo systemctl restart patchmon-agent`.

Full details: Managing the PatchMon Agent: "Connectivity Test Failed".

Report Interval Not Updating

You changed the report interval in **Settings → Agent Updates**, but the affected host still reports on the old schedule.

Background: The server pushes `update_interval` changes to connected agents via the WebSocket. If the WebSocket is down, or the change was made just before a restart, the agent may be running on its cached value in `/etc/patchmon/config.yml`.

Fix:

```
# 1. Verify the WebSocket is up (host shows as Online in the UI)
# 2. Restart the agent so it fetches fresh values on startup
sudo systemctl restart patchmon-agent

# 3. Check the value the agent sees
sudo patchmon-agent config show | grep -i interval
```

The agent syncs `update_interval`, `docker_enabled`, and `compliance_enabled` from the server on startup and on every WebSocket-pushed setting change, then writes the result back to `config.yml`. No restart is normally required for runtime sync. A restart is only needed for the first startup sync.

Full details: Managing the PatchMon Agent: Configuration Management (see "When do changes require a restart?" table).

Auto-Update Failing

The agent log contains `update failed`, `hash mismatch`, `binary verification failed`, or `update was performed X ago, skipping`.

Diagnose:

```
sudo patchmon-agent version
sudo patchmon-agent check-version
ls -la /etc/patchmon/.last_update_timestamp
ls -la /usr/local/bin/patchmon-agent.backup.*
```

Common causes:

- **Update loop cooldown**: the agent refuses to re-update within 5 minutes of the last attempt. Wait 5 minutes.
- **Server has not shipped a new binary**: verify on the server that the file in `AGENT_BINARIES_DIR` (or `AGENTS_DIR`) matches the expected architecture.
- **Hash mismatch**: the server must send a SHA-256 hash of the binary. If it doesn't, the agent refuses to install (this is mandatory for security, not a bug). Update your server to a release that provides hashes.
- **SSL error during download**: treat as Cannot reach server: TLS. `skip_ssl_verify` is **explicitly blocked** for binary downloads in production, so fix the certificate trust instead.

Auto-update disabled: `patchmon-agent check-version` will say `Auto-update disabled by server administrator`. Enable it at **Settings → Agent Updates → Master auto-update** and the per-host toggle on the host detail page.

Manual force:

```
sudo patchmon-agent update-agent
```

Full details: Managing the PatchMon Agent: Agent Updates and Agent Not Auto-Updating.

Credentials File Missing or Corrupt

Log contains `credentials file not found`, `failed to load credentials`, or `API credentials are missing`.

Diagnose:

```
ls -la /etc/patchmon/credentials.yml
# expected: -rw------- 1 root root  ~120  <date>  /etc/patchmon/credentials.yml

sudo cat /etc/patchmon/credentials.yml
# expected:
#   api_id: "patchmon_abc123"
#   api_key: "<64 hex chars>"
```

If the file is missing:

```
# Reconfigure using the credentials shown at enrolment time.
# If you don't have them, regenerate from the UI:
#   Hosts → <host> → Show Credentials → Regenerate
sudo patchmon-agent config set-api <API_ID> <API_KEY> <SERVER_URL>
```

If the file is present but permissions are wrong:

```
sudo chmod 600 /etc/patchmon/credentials.yml
sudo chown root:root /etc/patchmon/credentials.yml
```

If the YAML is malformed (e.g. manually edited and broken), either restore the latest backup (`ls -la /etc/patchmon/credentials.yml.backup.*`) or re-run `config set-api`.

Full details: Managing the PatchMon Agent: "Credentials File Not Found" and `config set-api`.

Logs Not Being Written

The agent is running but `/etc/patchmon/logs/patchmon-agent.log` is empty or missing.

Diagnose:

```
# Does the directory exist and is it writable by root?
ls -la /etc/patchmon/logs/
# expected: drwx------ 2 root root ...

# Check disk space
df -h /etc/patchmon
```

Common causes:

- **Log directory missing**: `sudo mkdir -p /etc/patchmon/logs && sudo chmod 700 /etc/patchmon/logs`.
- `log_file` **points elsewhere**: run `sudo patchmon-agent config show` and check the path.
- **Disk full**: `df -h`. Free space or change `log_file` to a different partition.
- **Running as the wrong user** (non-standard install): the agent must run as root. See Permission denied.
- **systemd-only logging**: on systemd, the agent logs to both the journal and the file. If the file is empty but `journalctl -u patchmon-agent` has entries, the file may be rotating correctly. Check for `.log.gz` backups.

The agent uses built-in log rotation (10 MB per file, 5 backups, 14-day retention). You do **not** need logrotate.

Full details: Managing the PatchMon Agent: Viewing Logs.

Service Keeps Restarting

`systemctl status patchmon-agent` shows `activating (auto-restart)` repeatedly, or the process count climbs.

Diagnose:

```
sudo systemctl status patchmon-agent
sudo journalctl -u patchmon-agent --since "15 minutes ago" --no-pager
```

Common causes:

- **Invalid** `config.yml` (YAML syntax error): the agent refuses to start; systemd restarts it every 10 s.
- **Invalid credentials**: the agent logs "invalid API credentials" and exits.
- **Server unreachable for long**: the agent will keep trying (it does **not** exit on network errors in normal operation), so constant restarts point at one of the two above.
- **Binary crash**: check `dmesg` for OOM kills or segfaults: `sudo dmesg -T | grep -i patchmon`.

The systemd unit uses `Restart=always` with `RestartSec=10`, which is correct for production but masks crash loops. Disable auto-restart temporarily to see the real error:

```
sudo systemctl edit patchmon-agent --force --full
# change: Restart=always  →  Restart=no
sudo systemctl daemon-reload
sudo systemctl start patchmon-agent
# read the single run's output, then revert:
sudo systemctl edit patchmon-agent --force --full    # restore Restart=always
sudo systemctl daemon-reload
```

Full details: Managing the PatchMon Agent: Service Keeps Restarting.

Permission Denied

Any agent command exits immediately with `permission denied` or `this script must be run as root`.

Fix:

> **Linux / FreeBSD:** use `sudo`. The agent reads package databases, writes to `/etc/patchmon/`, and manages the system service; none of which work without root.
>
> **Windows:** open PowerShell with **Run as Administrator**. A non-elevated PowerShell cannot read installed-package inventory or manage the `PatchMonAgent` service.

If you are root and still see permission errors, check file ownership:

```
ls -la /etc/patchmon/config.yml /etc/patchmon/credentials.yml
/usr/local/bin/patchmon-agent
# All three should be owned by root.
```

Full details: Managing the PatchMon Agent: "Permission Denied" Errors.

Escalation

If the decision tree above does not match your symptom, collect the following before asking for help:

```
# Everything in one command
sudo patchmon-agent diagnostics > /tmp/patchmon-diag.txt 2>&1
sudo journalctl -u patchmon-agent --since "1 hour ago" --no-pager \
  > /tmp/patchmon-journal.txt 2>&1
sudo tail -n 200 /etc/patchmon/logs/patchmon-agent.log \
  > /tmp/patchmon-log.txt 2>&1
sudo patchmon-agent config show > /tmp/patchmon-config.txt 2>&1
sudo patchmon-agent version >> /tmp/patchmon-config.txt
uname -a >> /tmp/patchmon-config.txt
```

Redact the `api_id` before sharing the bundle publicly. The full `api_id` is sensitive even though the key hash is not shown by `config show`.

See Also

- Managing the PatchMon Agent: full CLI, service, log, and removal reference.
- Managing the PatchMon Agent: Common Troubleshooting: the detailed counterpart to this decision tree.
- Agent Configuration Reference (config.yml): every config parameter.
- Installing the PatchMon Agent: enrolment walkthrough.
- Uninstalling the PatchMon Agent: removal walkthrough.
- Server Troubleshooting: for server-side issues.

Chapter 14: Errors on Dashboard After Proxmox Community Update

> ***Note:*** *This only applies to **1.x versions before 1.4.2**. It is not applicable to 2.0.*

Symptom

After upgrading via Proxmox community scripts, the PatchMon dashboard shows errors like "network error" or similar. The UI loads but cannot talk to the backend.

Cause

This is caused by a stale `VITE_API_URL` variable in the frontend environment file (`frontend/.env`). The built frontend bakes in that URL and can no longer reach the backend at runtime.

Fix

Open the frontend env file:

```
sudo nano /opt/<your-domain>/frontend/.env
```

Remove or comment out the line:

```
VITE_API_URL=...
```

Change directory to the PatchMon install (where both `frontend/` and `backend/` live) and rebuild the frontend:

```
cd /opt/<your-domain>
npm run build
```

Reload the page. The dashboard should work again.

Rule of thumb

If `VITE_API_URL` is set, it **must** match `CORS_ORIGIN` in the backend `.env`. If you build and deploy the frontend with several different `VITE_API_URL` values (e.g. internal and external bundles served from the same backend), each of those origins must appear in `CORS_ORIGIN`, comma-separated with no spaces (e.g. `CORS_ORIGIN=https://patchmon.example.com,https://patchmon.internal.lan`).

Ideally, leave `VITE_API_URL` unset and let the frontend use the current origin. The default build is correct for most deployments.

Not seeing the fix?

This issue is resolved in versions ≥ 1.4.2. If you're still on an older version, upgrading to the latest is the preferred fix.

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