Agent Side Management

patchmon-agent Management

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

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)

Table of Contents

CLI Command Reference

All commands must be run as **root** (or with `sudo`). The agent will refuse to run if it does not have root privileges.

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:

1. Loads configuration and credentials from `/etc/patchmon/`
2. Sends a startup ping to the PatchMon server
3. Establishes a persistent WebSocket connection (real-time commands)
4. Sends an initial system report in the background
5. Starts periodic reporting on the configured interval (default: 60 minutes)
6. Syncs integration status and update interval from the server
7. 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. This is extremely 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:

1. **Network connectivity** — can the agent reach the PatchMon server?
2. **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 v1.4.0
```

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

Agent Information:
  Version: 1.4.0
  Config File: /etc/patchmon/config.yml
  Credentials File: /etc/patchmon/credentials.yml
  Log File: /etc/patchmon/logs/patchmon-agent.log
  Log Level: info

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: 1.4.0
  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_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:

1. Validates the inputs (non-empty, valid URL format)
2. Saves the server URL to `/etc/patchmon/config.yml`
3. Saves the credentials to `/etc/patchmon/credentials.yml` (with `600` permissions)
4. 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 1.4.0)
```

Output when update is available:

```
Agent update available!
  Current version: 1.3.2
  Latest version: 1.4.0

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

Output when auto-update is disabled on the server:

```
Current version: 1.3.2
Latest version: 1.4.0
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:

1. Checks for recent updates (prevents update loops within 5 minutes)
2. Queries the server for the latest version
3. Downloads the new binary
4. **Verifies binary integrity** via SHA-256 hash comparison (mandatory)
5. Creates a timestamped backup of the current binary (e.g., `patchmon-agent.backup.20260212_143000`)
6. Writes the new binary to a temporary file and validates it
7. Atomically replaces the current binary
8. Cleans up old backups (keeps the last 3)
9. 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 v1.4.0
```

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 &
```

Viewing Logs

The agent writes logs to two locations depending on the init system:

Init System	Journal	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`

Log File Details

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)

The agent uses the lumberjack 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
```

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=1.4.0
```

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:

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

When an update is detected:

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

Manual Update

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

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

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:

```
/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)
```

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)

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

This 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

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:

```
# 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"
```

> **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` is blocked when the `PATCHMON_ENV` environment variable is set to `production`. This is a security measure to prevent disabling TLS verification in production.

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

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 Operating Systems

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+

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

config.yml Mangement and parameters

Overview

The PatchMon agent is configured through a YAML configuration file located at `/etc/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 (`/etc/patchmon/credentials.yml`) stores the host's API authentication details.

Both 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	`/etc/patchmon/config.yml`	Agent settings, server URL, integrations
Credentials	`/etc/patchmon/credentials.yml`	API ID and API Key for host authentication
Log File	`/etc/patchmon/logs/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 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.
  # Three modes:
  #   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.
  # Can be toggled from the PatchMon UI.
  # Default: "on-demand"
  compliance: "on-demand"

  # 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
```

Parameters In Detail

`patchmon_server`

Type	String (URL)
Required	Yes
Default	None — must be provided
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 API request/response bodies and detailed execution flow. 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/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 that agents across your fleet do not all report at the exact 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 → Integrations tab → Toggle Docker on/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 that are 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 effectively 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

1. SSH into the host where the PatchMon agent is installed
2. Open the config file:

```
sudo nano /etc/patchmon/config.yml
```

3. Find the `integrations` section and change `ssh-proxy-enabled` to `true`:

```
integrations:
  docker: false
  compliance: "on-demand"
  ssh-proxy-enabled: true    # ← Change from false to true
```

4. Save the file and restart the agent:

```
# Systemd
sudo systemctl restart patchmon-agent.service

# OpenRC (Alpine)
sudo rc-service patchmon-agent restart
```

5. 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.

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) or `"disabled"` (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: "disabled"
  ssh-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:

1. The installer **checks if the existing configuration is valid** by running `patchmon-agent ping`
2. If the ping succeeds, the installer **exits without overwriting** — the existing configuration is preserved
3. 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`

This means **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 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
`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: true` setting is **preserved** across server-driven updates
- New integrations added in agent updates are **automatically added** to the file with their defaults (you'll 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>
```

Example:

```
sudo patchmon-agent config set-api patchmon_1a2b3c4d abcdef123456 https://patchmon.example.com
```

This command:

1. Validates the server URL format
2. Saves the server URL to `config.yml`
3. Saves the credentials to `credentials.yml`
4. Tests connectivity with a ping to the server
5. 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:

1. **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.
2. **Agent updates**: After an agent update, new integration keys may appear in the file with default values.
3. **Reinstallation**: A reinstall only overwrites config if the existing ping test fails.

Your `ssh-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
```

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 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: "on-demand"
  ssh-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: false
ssh-proxy-enabled: false
```