PatchMon PATCHMON DOCS

PatchMon API & Integrations Guide

Discord, gethomepage, Ansible, Proxmox auto-enrollment, plus the Auto-Enrollment and Integration REST APIs with an embedded OpenAPI browser.

LAST UPDATED

22 August 2026

CHAPTERS

7

READ ONLINE

https://patchmon.net/docs/patchmon-api-integrations-guide

SOURCE

github.com/PatchMon/PatchMon

Contents

1. Table of Contents
2. Chapter 1: Discord Notifications
3. Chapter 2: gethomepage Dashboard Card
4. Chapter 3: Ansible Dynamic Inventory
5. Chapter 4: Proxmox LXC Auto-Enrollment Guide
6. Chapter 5: Auto-Enrollment API Documentation
7. Chapter 6: Integration API Documentation

This guide covers PatchMon's third-party integrations (Discord, gethomepage, Ansible, Proxmox LXC auto-enrollment) and the REST APIs exposed by the server (Auto-Enrollment API, Integration API). A live, interactive API browser rendered from the server's OpenAPI spec is embedded in the published version of this book on patchmon.net/docs.

Table of Contents

Chapter 1: Discord Notifications

Chapter 2: gethomepage Dashboard Card

Chapter 3: Ansible Dynamic Inventory

Chapter 4: Proxmox LXC Auto-Enrollment Guide

Chapter 5: Auto-Enrollment API Documentation

Chapter 6: Integration API Documentation

Chapter 1: Discord Notifications

PatchMon integrates with Discord in two separate, independent ways:

Discord OAuth2 login: let users sign in to PatchMon with their Discord account, or link an existing PatchMon account to a Discord identity. Configured under **Settings → Discord Auth**.

Discord as a notification / alert destination: fire PatchMon alerts and scheduled reports into a Discord channel via an incoming webhook. Configured under **Settings → Alert Channels** as a `webhook` destination.

You can enable either, both, or neither. They don't depend on each other.

> ***Related pages:***
>
> *Users, Roles and RBAC: manage roles and account linking*
>
> *Setting Up OIDC / Single Sign-On: an alternative way to delegate login to an external IdP*

Part 1: Discord OAuth2 Login

Let users authenticate to PatchMon with their Discord account. PatchMon supports three related flows:

Sign in with Discord (for users who don't yet exist): auto-creates a PatchMon account if self-registration is enabled.

Sign in with Discord (for users who do exist): auto-links by matching the verified Discord email to the user's PatchMon email.

Link Discord to an existing logged-in account: from the Profile page, attach a Discord identity to your PatchMon account without changing your password.

Everything is configured through the Settings UI. No environment variables are required; secrets are stored encrypted in the PatchMon database.

What you'll end up with

An additional **Login with Discord** button on the PatchMon login page.

Optional automatic account creation on first Discord login (driven by the PatchMon signup setting).

A Discord avatar and username visible on each user's profile.

Before you begin

You need:

Item	Notes
A running PatchMon instance	Reachable at a fixed URL, e.g. `https://patchmon.example.com`
HTTPS on the PatchMon URL	Discord requires `https://` redirect URIs in production
A PatchMon admin account with `can_manage_settings`	To reach the Discord Auth settings page
A Discord account	To access the Discord Developer Portal

Step 1: Find your callback URL

The callback is derived from PatchMon's configured server URL and is shown to you on the settings screen, but for reference the canonical path is:

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

If PatchMon is showing the wrong hostname (for example `http://localhost:3000` when you're running in production), fix your **Server URL** in **Settings → Server Config** first. The callback URL is read-only in the Discord settings panel and is rebuilt from the server URL whenever you save.

Step 2: Create a Discord application

Go to the Discord Developer Portal (https://discord.com/developers/applications).

Click **New Application** and give it a name (e.g. `PatchMon`).

In the left menu, open **OAuth2**.

Under **Redirects**, click **Add Redirect** and paste your callback URL:

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

Click **Save Changes** at the bottom.

Copy the **Client ID** (shown at the top). You'll paste it into PatchMon in the next step.

Click **Reset Secret** (or **Copy** if the secret is already visible), and save the value. Discord will only show this once. If you lose it, you'll have to reset it again.

> *You do **not** need to set up an OAuth2 URL / redirect URL generator in Discord. PatchMon builds the authorisation URL itself. The only field that matters in the Discord UI is the **Redirects** list.*

Step 3: Configure PatchMon

Sign in to PatchMon as an admin.

Go to **Settings → Discord Auth**.

Fill in the OAuth2 Configuration panel:

Client ID: the Application ID from Discord's app overview.

Client Secret: paste the secret from Step 2 into the field and click **Save**. The **Not set** badge should flip to **Set** (green tick). PatchMon encrypts the secret at rest using its configured `SECRET_ENCRYPTION_KEY`.

Redirect URI: usually leave blank. PatchMon derives the callback from the server URL automatically. Only override if you're behind a proxy that presents a different public URL.

Button Text: customise the login button label, e.g. `Sign in with Discord`. Defaults to `Login with Discord`.

Click **Apply** to save the text fields.

At the top of the panel, flip **Enable Discord OAuth** to on.

Step 4: Test

Open PatchMon in a private / incognito browser window.

On the login page you should now see a **Login with Discord** (or your custom label) button.

Click it. Discord will ask you to authorise the `PatchMon` application.

Accept. You'll be redirected back to PatchMon.

First-login behaviour

If a PatchMon user with the same email already exists and the Discord email is **verified**, PatchMon automatically links the accounts. You're logged in.

If no PatchMon user exists and self-registration is on (**Settings → Users → User Registration Settings → Enable User Self-Registration**), PatchMon creates a new account with:

Username: derived from the Discord username, stripped of unsafe characters, with a numeric suffix if the base name collides.

Email: the Discord email (or `discord_<id>@discord.local` if Discord doesn't expose an email).

Role: the **Default Role for New Users** setting.

If no PatchMon user exists and self-registration is off, the login flow redirects to `/login?error=User+not+found`. An admin must create the account first; next time, the verified-email auto-link kicks in.

Linking Discord to an existing PatchMon account

This is the safer alternative to "Sign in with Discord" for users who already have a PatchMon account. It lets them keep their username / email / password workflow and just adds a Discord badge.

User signs in to PatchMon as normal.

Clicks their avatar → **Profile**.

Scrolls to the **Linked Accounts** section and clicks **Link Discord**.

PatchMon redirects them to Discord to authorise, then back to the profile page.

On success, the profile shows the Discord username and avatar, and a small "discord_linked=true" success banner.

Unlinking

Same panel → **Unlink Discord**. PatchMon refuses to unlink if Discord is the user's only login method (no password set, no OIDC linked), as this would lock the user out. Set a password in the **Change Password** panel first, then retry the unlink.

Troubleshooting: OAuth login

The "Login with Discord" button doesn't appear on the login page

Toggle is off. Check **Settings → Discord Auth → Enable Discord OAuth**.

Client secret is missing. The badge next to the field should say **Set**. If it says **Not set**, paste the secret and click **Save**.

Client ID is blank. Check the same panel; the Client ID field must be populated.

Redirect error: "The redirect URI isn't registered"

The URL Discord is being asked to redirect to doesn't match anything in the Discord app's **Redirects** list.

In Discord's Developer Portal, open your app → **OAuth2** → **Redirects** and make sure `https://patchmon.example.com/api/v1/auth/discord/callback` is listed **exactly**. The protocol (`https://`), host, port, and path must all match.

Don't include a trailing slash; don't include query strings.

If PatchMon is behind a reverse proxy, make sure PatchMon's **Server URL** reflects the public URL, not the internal one.

Error: "Discord is not fully configured"

One of **Client ID** or **Client Secret** is missing. Fill them both in, then click **Apply** and **Save** respectively.

Error: "Already linked" when linking

Someone else in PatchMon is already linked to that Discord account. Only one PatchMon user can hold a given Discord identity at a time.

First-login auto-create didn't happen

Auto-create only runs when **Settings → Users → User Registration Settings → Enable User Self-Registration** is on. If it's off, pre-create the user (with a matching email) and try again.

Part 2: Discord as a Notification / Alert Destination

PatchMon can push alerts, events and scheduled reports to a Discord channel via an **incoming webhook** (Discord's built-in mechanism for posting into a channel from an external service). This is handled by the generic "webhook" alert channel. PatchMon detects Discord URLs automatically and formats the message as a Discord embed.

What you'll end up with

A Discord channel that receives rich embedded messages for every PatchMon event of the type(s) you've subscribed.

Colour-coded severity (critical = red, error = orange, warning = yellow, informational = blue).

Structured fields based on the event type (container stops, host down, user role changes, etc.).

Scheduled reports (daily / weekly / monthly summaries) also delivered as embeds, with a plain-text excerpt and CSV attached where supported.

Step 1: Create a Discord incoming webhook

In Discord, open the **server** (guild) that owns the target channel.

Server settings → **Integrations** → **Webhooks** → **New Webhook**.

Give the webhook a name (e.g. `PatchMon`), pick the target channel, optionally set an avatar.

Click **Copy Webhook URL**. You should now have a URL shaped like:

```
https://discord.com/api/webhooks/1234567890/abcdefgh-ABCDEFGH1234567890
```

Keep it safe. Anyone who holds this URL can post to your channel.

Step 2: Add the webhook to PatchMon

Sign in to PatchMon with a role that has `can_manage_notifications`.

Go to **Settings → Alert Channels**.

Click **Add Destination**.

Pick **Webhook** as the channel type.

Fill in:

- **Display Name**: e.g. `Ops Discord`. Any label that helps you identify the channel later.
- **Webhook URL**: paste the Discord webhook URL from Step 1.

PatchMon detects it is a Discord URL automatically (the UI shows "Discord and Slack URLs are auto-detected for rich formatting"). Nothing else to configure for Discord.

Click **Save**.

> ***Heads-up:*** *Anything else that starts with* `https://discord.com/api/webhooks/`, `https://discordapp.com/api/webhooks/`, *or* `https://www.discord.com/api/webhooks/` *is treated as Discord and formatted with embeds. Slack URLs are detected similarly. Everything else is sent as a plain JSON* `{"title":..., "message":..., "severity":...}` *POST, which you can consume with your own handler.*

Step 3: Route alerts to the destination

Creating the destination does not automatically route any events to it. You need at least one routing rule.

Still on **Settings → Alert Channels**, scroll to the **Routing Rules** section.

Click **Add Rule**.

Pick the destination you just created from the dropdown.

Choose the events / severities you want to send. The recommended starter set:

- `host_went_down`
- `host_came_up`
- `container_stopped`
- `security_updates_available`
- `user_tfa_disabled`
- `account_locked`

Save the rule.

Your Discord channel should start receiving notifications on the next matching event. To test quickly, simulate a host-down event by stopping the PatchMon agent on any non-critical host and waiting for the next check-in cycle.

Step 4 (optional): Route scheduled reports

Alongside real-time alerts, PatchMon can send a periodic summary report to the same webhook.

On the **Alert Channels** page, scroll to **Scheduled Reports**.

Click **Add Schedule**.

Configure:

Destinations: tick your Discord webhook.

Frequency: daily, weekdays, weekly (pick days), or monthly (pick day or "last day").

Delivery time: hour and minute in your server's timezone.

Sections: which report sections to include (Open alerts, Hosts by outstanding updates, Top outdated security packages).

Save.

For Discord delivery, scheduled reports are rendered as:

A title with the report subject.

A short plain-text excerpt of the HTML body (tags stripped).

A **PatchMon** footer.

If a CSV attachment is configured, it is posted as a separate file via Discord's multipart upload.

Message format

Real-time alerts

Each event becomes a Discord embed:

Title: the event title (e.g. `Host Down: web01.example.com`).

Description: the full event message.

Colour: derived from severity (`critical` red, `error` orange, `warning` yellow, `informational` blue, everything else grey).

Fields: structured fields per event type (e.g. for `container_stopped`: host name, container name, image, old status, new status).

Footer: `PatchMon`.

Scheduled reports

Title: report subject line.

Description: excerpt of the HTML body, with tags (including `<script>` blocks) stripped.

Footer: `PatchMon`.

Troubleshooting: Notifications

Webhook URL shows "Webhook URL is required"

The form rejected an empty URL. Paste the full Discord webhook URL you copied in Step 1.

Destination saved but no Discord messages arrive

Walk through this list in order:

Did any matching event fire? Check **Alerts → Notification Logs**. If the log shows no rows for your destination, no events matched your routing rules. Adjust the rules.

Does the log show a failure? Filter the log by destination. If the delivery attempt failed, hover over the row to see the error Discord returned. Common ones:

`401` or `404`: the webhook has been deleted in Discord. Re-create it and update the URL.

`429 Too Many Requests`: you're hitting Discord's rate limit. Reduce the event volume, or split across multiple webhooks / channels.

Did PatchMon even try? Check the PatchMon server logs:

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

Is the URL actually Discord? PatchMon only formats as an embed when the URL hostname is `discord.com`, `discordapp.com`, or `www.discord.com` **and** the path contains `/api/webhooks/`. A typo in the URL (e.g. `discord.co` or no `/api/` segment) falls back to

the generic JSON POST format, which Discord will reject. Confirm the URL contains `/api/webhooks/`.

Posts are plain text, not embeds

The URL is not being recognised as Discord. See the last point above and verify the exact hostname and path.

Everything works but messages are posted to the wrong channel

The webhook URL encodes the target channel. In Discord, go to server settings → **Integrations** → **Webhooks**, select the webhook, and change **Channel**. Alternatively, create a new webhook for the correct channel and update PatchMon to use it.

I want to remove the webhook cleanly

In PatchMon, **Settings → Alert Channels**, find the destination, click **Delete**.

In Discord, server settings → **Integrations** → **Webhooks**, find the webhook, click **Delete Webhook**. This is the reliable way to revoke. Deleting only in PatchMon leaves the URL live; if anyone else captured the URL they can still post to your channel.

Security notes

OAuth login

The Discord **Client Secret** is stored encrypted in the PatchMon database using the server's `SECRET_ENCRYPTION_KEY`. Make sure that environment variable is set and is not the default value in production.

Account linking by email is only performed when Discord reports the user's email as **verified**, to prevent account takeover via an unverified email address.

PatchMon uses **PKCE (S256)** for Discord OAuth2 code exchange, so the authorisation code can't be replayed even if intercepted.

The Discord OAuth **state** is tied to a short-lived (10-minute) session stored in Redis; it's one-time-use and bound to an HttpOnly `discord_state` cookie.

Webhooks

Discord webhook URLs are **bearer tokens**. Anyone with the URL can post to your channel. Treat the webhook URL like a password.

PatchMon stores webhook URLs encrypted at rest if a `SECRET_ENCRYPTION_KEY` is configured. Without one, URLs are stored in plaintext. Don't skip setting the encryption key.

Do not paste webhook URLs into public GitHub issues, screenshots, or chat channels.

Consider creating a dedicated Discord channel and webhook per PatchMon environment (prod / staging) so you can revoke them independently.

Quick reference

Task	Where
Create / edit Discord OAuth app	Discord Developer Portal (https://discord.com/developers/applications)
Enable Discord login in PatchMon	**Settings → Discord Auth**
Sign in via Discord	Login page → **Login with Discord** button
Link existing account to Discord	**Profile → Linked Accounts → Link Discord**
Create Discord webhook	Server → Settings → Integrations → Webhooks
Add webhook to PatchMon	**Settings → Alert Channels → Add Destination → Webhook**
Route events to Discord	**Settings → Alert Channels → Routing Rules**
Schedule summary reports to Discord	**Settings → Alert Channels → Scheduled Reports**
Check delivery history	**Alerts → Notification Logs**

Chapter 2: gethomepage Dashboard Card

PatchMon exposes a dedicated read-only endpoint designed to be consumed by a GetHomepage (https://gethomepage.dev/) (formerly *Homepage*) `customapi` widget. Drop a PatchMon card into your existing homepage to see total hosts, pending updates, and security updates at a glance.

> ***Related pages:***
>
> *Integration API Documentation: the generic scoped API (a different integration type)*
>
> *Users, Roles and RBAC: permission required to create the API key*

At a glance

Endpoint: `GET /api/v1/gethomepage/stats`

Auth: HTTP Basic, using a PatchMon-issued API key dedicated to the GetHomepage integration

Widget type: `customapi` (https://gethomepage.dev/widgets/services/customapi/) in GetHomepage

Fields available: 8 core metrics + a top-3 OS breakdown + a full `os_distribution` array

Rate limit: shares the standard API rate limit; GetHomepage polls every 60 seconds, well within the limit

Default widget

Out of the box the widget shows three metrics:

Total Hosts

Hosts Needing Updates

Security Updates

Additional metrics can be added by editing the `mappings:` in your GetHomepage `services.yml`. See Configuration options below.

Prerequisites

A running PatchMon 2.x instance reachable from the machine running GetHomepage.

GetHomepage already installed and rendering at least one page.

Network path between GetHomepage and PatchMon on HTTP or HTTPS. HTTPS is strongly recommended.

PatchMon admin access (you need `can_manage_settings` to create API keys).

Setup

Step 1: Create a GetHomepage API key

Sign in to PatchMon as an admin.

Go to **Settings → Integrations**.

Open the **GetHomepage** tab.

Click **New API Key** and fill in:

Token Name: e.g. `GetHomepage dashboard`.

Allowed IP Addresses *(optional)*: restrict to the IP of the machine running GetHomepage.

Expiration Date *(optional)*: set one if this is a temporary key.

Click **Create Token**.

Step 2: Copy the credentials

A success modal is shown with:

Token Key: the API username.

Token Secret: the API password. **Shown only once. Save it immediately.**

Base64-encoded credentials: pre-built `Authorization: Basic` value, ready to paste.

Complete widget configuration: a ready-to-drop-in YAML snippet.

> *Click **Copy Config** to copy the full YAML block. The secret is never retrievable again after you close this modal. If you lose it, you have to delete the key and create a new one.*

Step 3: Configure GetHomepage

Option A: Paste the copied YAML (quickest)

Open your GetHomepage `services.yml`.

Paste the YAML block that PatchMon gave you.

Save the file.

Restart GetHomepage.

The YAML looks like this:

```
- PatchMon:
    href: https://patchmon.example.com
    description: PatchMon Statistics
    icon: https://patchmon.example.com/assets/favicon.svg
    widget:
      type: customapi
      url: https://patchmon.example.com/api/v1/gethomepage/stats
      headers:
        Authorization: Basic <base64_encoded_credentials>
      mappings:
        - field: total_hosts
          label: Total Hosts
        - field: hosts_needing_updates
          label: Needs Updates
        - field: security_updates
          label: Security Updates
```

Option B: Build it by hand

Encode your credentials:

```
echo -n "YOUR_TOKEN_KEY:YOUR_TOKEN_SECRET" | base64
```

Paste the widget into `services.yml`, replacing `<your_base64_credentials>` with the result:

```
- PatchMon:
    href: https://patchmon.example.com
    description: PatchMon Statistics
    icon: https://patchmon.example.com/assets/favicon.svg
    widget:
      type: customapi
      url: https://patchmon.example.com/api/v1/gethomepage/stats
      headers:
        Authorization: Basic <your_base64_credentials>
      mappings:
        - field: total_hosts
          label: Total Hosts
        - field: hosts_needing_updates
          label: Needs Updates
        - field: security_updates
          label: Security Updates
```

Restart GetHomepage:

```
docker restart gethomepage
# or
systemctl restart gethomepage
```

Configuration options

Customising the fields displayed

The default configuration displays **3 metrics**. You can add more. PatchMon returns **8 numeric metrics** and the top-3 OS breakdown, and the widget supports 6–8 comfortably before it becomes cluttered.

Each `mappings` entry has two parts:

- `field:` the JSON key returned by the PatchMon API (case-sensitive, exactly as listed below)
- `label:` the human-readable label rendered by GetHomepage

Available fields

Field	Type	Description	Included by default
`total_hosts`	Number	Total hosts in PatchMon, matching the Total Hosts card on the dashboard	Yes
`hosts_needing_updates`	Number	Hosts with at least one outdated package	Yes
`security_updates`	Number	Total security updates available across all hosts	Yes
`up_to_date_hosts`	Number	Hosts with zero outdated packages	No
`total_outdated_packages`	Number	Sum of all outdated packages across hosts	No
`hosts_with_security_updates`	Number	Hosts requiring at least one security patch	No
`total_repos`	Number	Active repositories being monitored	No
`recent_updates_24h`	Number	Successful updates in the last 24 hours	No
`top_os_1_name`	String	Name of the most common OS (e.g. "Ubuntu")	No (use label instead, see below)
`top_os_1_count`	Number	Count of the most common OS	No
`top_os_2_name`	String	Name of the 2nd most common OS	No
`top_os_2_count`	Number	Count of the 2nd most common OS	No
`top_os_3_name`	String	Name of the 3rd most common OS	No
`top_os_3_count`	Number	Count of the 3rd most common OS	No
`os_distribution`	Array	Full OS breakdown (advanced use only; GetHomepage cannot render arrays directly)	No
`last_updated`	String (ISO 8601)	Timestamp the stats were generated	No

> *The `top_os_*_name` string fields render poorly in `customapi` widgets. Use the corresponding `_count` fields and put the OS name in the `label:`. See Displaying OS distribution.*

Quick recipe: add a fourth metric

Before:

```
mappings:
  - field: total_hosts
    label: Total Hosts
  - field: hosts_needing_updates
    label: Needs Updates
  - field: security_updates
    label: Security Updates
```

After:

```
mappings:
  - field: total_hosts
    label: Total Hosts
  - field: hosts_needing_updates
    label: Needs Updates
  - field: security_updates
    label: Security Updates
  - field: recent_updates_24h      # newly added
    label: Updated (24h)
```

Save, restart GetHomepage, and you've gone from 3 to 4 metrics.

Example widget configurations

All examples assume you've already populated the `Authorization` header with your encoded credentials.

Security-focused widget

```
widget:
  type: customapi
  url: https://patchmon.example.com/api/v1/gethomepage/stats
  headers:
    Authorization: Basic <credentials>
  mappings:
    - field: security_updates
      label: Security Patches
    - field: hosts_with_security_updates
      label: Hosts at Risk
    - field: hosts_needing_updates
      label: Total Pending
```

Repository / coverage widget

```
widget:
  type: customapi
  url: https://patchmon.example.com/api/v1/gethomepage/stats
  headers:
    Authorization: Basic <credentials>
  mappings:
    - field: total_repos
      label: Repositories
    - field: total_hosts
      label: Managed Hosts
    - field: up_to_date_hosts
      label: Up-to-Date
```

Activity widget

```
widget:
  type: customapi
  url: https://patchmon.example.com/api/v1/gethomepage/stats
  headers:
    Authorization: Basic <credentials>
  mappings:
    - field: recent_updates_24h
      label: Updated (24h)
    - field: hosts_needing_updates
      label: Pending Updates
    - field: up_to_date_hosts
      label: Fully Patched
```

Maximum-information widget (all 8 numeric metrics)

```
widget:
  type: customapi
  url: https://patchmon.example.com/api/v1/gethomepage/stats
  headers:
    Authorization: Basic <credentials>
  mappings:
    - field: total_hosts
      label: Total Hosts
    - field: hosts_needing_updates
      label: Needs Updates
    - field: up_to_date_hosts
      label: Up-to-Date
    - field: security_updates
      label: Security Updates
    - field: hosts_with_security_updates
      label: Security Hosts
    - field: total_outdated_packages
      label: Outdated Packages
    - field: total_repos
      label: Repositories
    - field: recent_updates_24h
      label: Updated (24h)
```

Note this widget will be quite tall. Keep it to 3–5 metrics for most layouts.

Multiple environments

```
# Production - security-focused
- PatchMon Prod:
    href: https://patchmon-prod.example.com
    description: Production Patches
    icon: https://patchmon-prod.example.com/assets/favicon.svg
    widget:
      type: customapi
      url: https://patchmon-prod.example.com/api/v1/gethomepage/stats
      headers:
        Authorization: Basic <prod_credentials>
      mappings:
        - field: total_hosts
          label: Hosts
        - field: security_updates
          label: Security
        - field: hosts_needing_updates
          label: Pending

# Development - package-focused
- PatchMon Dev:
    href: https://patchmon-dev.example.com
    description: Development Patches
    icon: https://patchmon-dev.example.com/assets/favicon.svg
    widget:
      type: customapi
      url: https://patchmon-dev.example.com/api/v1/gethomepage/stats
      headers:
        Authorization: Basic <dev_credentials>
      mappings:
        - field: total_hosts
          label: Hosts
        - field: total_outdated_packages
          label: Packages
        - field: up_to_date_hosts
          label: Updated
```

Displaying OS distribution

Step 1: Find out your top 3 operating systems

```
curl -s -H "Authorization: Basic YOUR_BASE64_CREDENTIALS" \
  https://patchmon.example.com/api/v1/gethomepage/stats \
  | jq '{top_os_1_name, top_os_1_count, top_os_2_name, top_os_2_count,
top_os_3_name, top_os_3_count}'
```

Sample output:

```
{
  "top_os_1_name": "Ubuntu",
  "top_os_1_count": 35,
  "top_os_2_name": "Debian",
  "top_os_2_count": 18,
  "top_os_3_name": "Rocky Linux",
  "top_os_3_count": 12
}
```

Step 2: Add the counts to the widget, using the names as labels

```
mappings:
  - field: total_hosts
    label: Total Hosts
  - field: top_os_1_count
    label: Ubuntu          # from top_os_1_name
  - field: top_os_2_count
    label: Debian          # from top_os_2_name
  - field: top_os_3_count
    label: Rocky Linux     # from top_os_3_name
```

Step 3: Restart GetHomepage

```
docker restart gethomepage
# or
systemctl restart gethomepage
```

The widget now shows your infrastructure OS breakdown. If your top 3 OSes change over time, update the labels; PatchMon will reorder the counts automatically based on actual host counts.

Custom icon

```
# PatchMon logo
icon: https://patchmon.example.com/assets/favicon.svg
icon: https://patchmon.example.com/assets/logo_dark.png
icon: https://patchmon.example.com/assets/logo_light.png

# GetHomepage built-in icon
icon: server

# Local icon inside your GetHomepage image / volume
icon: /icons/patchmon.png
```

API reference

Endpoints

Method	Path	Description
`GET`	`/api/v1/gethomepage/stats`	Returns the widget payload described above
`GET`	`/api/v1/gethomepage/health`	Simple liveness probe. Returns `status: "ok"`, the current timestamp, and the name of the API key used.

Authentication

Type: HTTP Basic Authentication

Format: `Authorization: Basic <base64(token_key:token_secret)>`

Token type: `gethomepage` (enforced server-side; a credential created under the **API** tab won't work here, and vice versa)

Stats response

```
{
  "total_hosts": 42,
  "total_outdated_packages": 156,
  "total_repos": 12,
  "hosts_needing_updates": 15,
  "up_to_date_hosts": 27,
  "security_updates": 23,
  "hosts_with_security_updates": 8,
  "recent_updates_24h": 34,
  "os_distribution": [
    { "name": "Ubuntu",     "count": 20, "os_type": "linux", "os_version": "22.04"
},
    { "name": "Debian",     "count": 12, "os_type": "linux", "os_version": "12" },
    { "name": "Rocky Linux", "count": 10, "os_type": "linux", "os_version": "9" }
  ],
  "top_os_1_name": "Ubuntu",
  "top_os_1_count": 20,
  "top_os_2_name": "Debian",
  "top_os_2_count": 12,
  "top_os_3_name": "Rocky Linux",
  "top_os_3_count": 10,
  "last_updated": "2026-04-24T12:34:56Z"
}
```

Health response

```
{
  "status": "ok",
  "timestamp": "2026-04-24T12:34:56Z",
  "api_key": "GetHomepage dashboard"
}
```

Managing API keys

Viewing existing keys

Go to **Settings → Integrations → GetHomepage**. For each key you see:

- Token name
- Creation date
- Last-used timestamp
- Active / Inactive status
- Expiration date (if set)

Disable / Enable / Delete

- **Disable / Enable**: toggle the button on the row to temporarily block or restore access without deleting the credential.
- **Delete**: click the trash icon. This is permanent; any widget using that key will start returning 401.

Security features

- **IP restrictions**: per-key allowlist (CIDRs supported).
- **Expiration dates**: automatic sunset.
- **Last-used tracking**: spot keys that have silently stopped working, or suspicious usage.
- **One-time secret display**: the secret is shown once, at creation. Never again.

Troubleshooting

Error: "Missing or invalid authorization header"

GetHomepage is not sending the `Authorization` header correctly.

- Verify the `headers:` section is properly indented in `services.yml`.
- Re-encode the credentials; make sure you used `-n` with `echo` so no trailing newline ends up in the base64.
- Confirm you're using `type: customapi`, as other widget types ignore arbitrary headers.

Error: "Invalid API key"

The key does not exist in PatchMon.

Check **Settings → Integrations → GetHomepage** for the key.

Re-create the key if it's missing, update the GetHomepage config with the new credentials.

Error: "API key is disabled" / "API key has expired"

Enable the key, or create a new one with a later expiration.

Error: "IP address not allowed"

Your GetHomepage instance's outbound IP is not in the credential's allowlist. Either add it, or remove the allowlist if not needed.

Widget shows nothing

Work through this checklist:

Can GetHomepage reach PatchMon at all? Test with `curl` from inside the GetHomepage container: `curl -v https://patchmon.example.com/api/v1/gethomepage/health -H "Authorization: Basic ..."`

Is the API key active and not expired?

Is the base64 credential correct?

Is `services.yml` valid YAML? (run `yamllint services.yml` if unsure)

Has GetHomepage been restarted since the last change?

Check GetHomepage's container logs for error messages.

Testing the endpoint directly

```
# Step 1: encode
echo -n "your_key:your_secret" | base64

# Step 2: test
curl -H "Authorization: Basic YOUR_BASE64" \
     https://patchmon.example.com/api/v1/gethomepage/stats | jq
```

Every numeric field in the response (including `top_os_*_count`) can be used in a widget mapping.

Security best practices

Always use HTTPS. The credentials are sent on every 60-second poll. Don't put them on the wire in the clear.

IP-restrict the key to the GetHomepage instance's IP.

Give the key an expiration and rotate it as part of your normal credential rotation.

Monitor the last-used timestamp to spot suspicious activity.

One key per GetHomepage instance if you're running several, to make rotation and revocation easier.

Store `services.yml` with appropriate file permissions on the GetHomepage host.

Integration architecture

Rate limiting

The `/api/v1/gethomepage/*` endpoints are subject to PatchMon's general API rate limit of 100 requests per 15 minutes per IP by default. GetHomepage's default poll interval of 60 seconds sits well within this limit (15 requests per 15 minutes). If you lower GetHomepage's poll interval aggressively, you may start hitting `429 Too Many Requests`; stay above 10 seconds.

Support and resources

PatchMon documentation: patchmon.net/docs (https://patchmon.net/docs)

GetHomepage documentation: gethomepage.dev (https://gethomepage.dev)

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

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

Chapter 3: Ansible Dynamic Inventory

The **patchmon.dynamic_inventory** Ansible plugin queries PatchMon's scoped integration API and turns it into a live Ansible inventory. Hosts and their group memberships stay in sync with PatchMon automatically, so you stop hand-editing `hosts.ini`.

GitHub repository: github.com/PatchMon/PatchMon-ansible (https://github.com/PatchMon/PatchMon-ansible)

Ansible Galaxy namespace: `patchmon.dynamic_inventory`

License: AGPL-3.0-or-later

> ***Related pages:***
>
> *Integration API Documentation: full reference for the scoped `/api/v1/api/...` endpoints the plugin talks to*
>
> *Auto-Enrolment API Docs: how to create the Basic-Auth credentials this plugin needs*

What the plugin does

For each request, the plugin:

- Calls `GET /api/v1/api/hosts` on your PatchMon instance with HTTP Basic Auth.
- Receives a JSON list of active hosts, their IPs, and their PatchMon host-group memberships.
- Builds an Ansible inventory in memory:
  - Each PatchMon host becomes an Ansible host, keyed by `hostname`.
  - `ansible_host` is set to the host's `ip` field (so Ansible connects directly to the IP even if DNS is iffy).
  - Each PatchMon **host group** becomes an Ansible group, and the host is added as a member.

The result is a fully dynamic `ansible-inventory --list` tree driven entirely by PatchMon's groupings.

Requirements

Component	Minimum version
Ansible	2.19.0
Python	3.6
`requests`	2.25.1

Install the Python dependency on the machine running `ansible`:

```
pip install 'requests>=2.25.1'
```

Installation

From Ansible Galaxy (recommended)

```
ansible-galaxy collection install patchmon.dynamic_inventory
```

From source

```
git clone https://github.com/PatchMon/PatchMon-ansible.git
cd PatchMon-ansible/patchmon/dynamic_inventory

# Build the collection tarball
ansible-galaxy collection build

# Install it locally
ansible-galaxy collection install patchmon-dynamic_inventory-*.tar.gz

# Install Python dependencies
pip install -r requirements.txt
```

Creating an API Credential in PatchMon

The plugin authenticates as an **integration API** credential (one of the scoped Basic-Auth tokens managed by PatchMon's integration API). It is **not** a normal user password.

Sign in to PatchMon as a user with `can_manage_settings`.

Go to **Settings → Integrations** and select the **API** tab.

Click **Create API Key** and fill in:

Name: e.g. `Ansible inventory`

Scopes: at minimum, `host:read`. If you want the plugin to read host stats as well, add the other read scopes. See Integration API Documentation for the full scope list.

Allowed IP addresses (optional): restrict the credential to the public IP of your Ansible controller.

Expiration (optional): set a date if the credential is temporary.

Click **Create**.

Copy the secret immediately. It is displayed only once. Save both the **Token Key** (the username) and **Token Secret** (the password).

> *The plugin's `api_key` config value is PatchMon's **Token Key**. The plugin's `api_secret` is PatchMon's **Token Secret**. The labels differ; the meaning is the same.*

Configuration

Create an inventory file, e.g. `patchmon_inventory.yml`:

```
---
plugin: patchmon.dynamic_inventory
api_url: https://patchmon.example.com/api/v1/api/hosts/
api_key: your_token_key
api_secret: your_token_secret
verify_ssl: true
```

Configuration options

Option	Required	Default	Description
`plugin`	yes	(required)	Must be `patchmon.dynamic_inventory`
`api_url`	yes	(required)	URL of the PatchMon scoped hosts endpoint. For PatchMon 2.x this is `https://<your-patchmon-host>/api/v1/api/hosts/`
`api_key`	yes	(required)	The **Token Key** from the PatchMon API credential
`api_secret`	yes	(required)	The **Token Secret** from the PatchMon API credential
`verify_ssl`	no	`true`	Whether to verify the PatchMon server's TLS certificate. Only disable on internal dev setups with self-signed certs

Using environment variables and Ansible Vault

Hard-coding the secret into `patchmon_inventory.yml` is not recommended. Use Ansible's environment-variable lookup or Ansible Vault instead:

```
---
plugin: patchmon.dynamic_inventory
api_url: https://patchmon.example.com/api/v1/api/hosts/
api_key: "{{ lookup('env', 'PATCHMON_API_KEY') }}"
api_secret: "{{ lookup('env', 'PATCHMON_API_SECRET') }}"
verify_ssl: true
```

Then:

```
export PATCHMON_API_KEY=your_token_key
export PATCHMON_API_SECRET=your_token_secret
ansible-inventory -i patchmon_inventory.yml --list
```

Making it the default inventory

Add to your `ansible.cfg`:

```
[defaults]
inventory = patchmon_inventory.yml

[inventory]
enable_plugins = patchmon.dynamic_inventory.dynamic_inventory
```

Every `ansible` / `ansible-playbook` / `ansible-inventory` invocation from this directory will now use PatchMon as its source of truth.

Usage

List all hosts

```
ansible-inventory -i patchmon_inventory.yml --list
```

Ping every host

```
ansible all -i patchmon_inventory.yml -m ping
```

Run a playbook against a PatchMon host group

If your PatchMon host group is named `web_servers`, the Ansible group name is also `web_servers`:

```
ansible-playbook -i patchmon_inventory.yml playbook.yml --limit web_servers
```

Intersect multiple groups

Standard Ansible group-pattern syntax applies. For example, to target all hosts in both `web_servers` **and** `production`:

```
ansible-playbook -i patchmon_inventory.yml playbook.yml --limit
'web_servers:&production'
```

API Response Format

The plugin expects the PatchMon API endpoint to return JSON shaped like this:

```
{
  "hosts": [
    {
      "hostname": "server1.example.com",
      "ip": "192.168.1.10",
      "host_groups": [
        { "name": "web_servers" },
        { "name": "production" }
      ]
    },
    {
      "hostname": "server2.example.com",
      "ip": "192.168.1.11",
      "host_groups": [
        { "name": "db_servers" },
        { "name": "production" }
      ]
    }
  ],
  "total": 2
}
```

This matches the shape returned by `GET /api/v1/api/hosts` in PatchMon 2.x (the `host_groups` array also contains an `id` field, which the plugin ignores).

Inventory mapping

- **Host name**: `hostname` becomes the Ansible inventory key.
- **Connection IP**: `ip` is set as the `ansible_host` variable on that host.
- **Groups**: every `{ "name": "...", "id": "..." }` in `host_groups` becomes an Ansible group, and the host is added to it.

Hosts with no entries in `host_groups` end up in Ansible's built-in `ungrouped` group.

Examples

List inventory output

```
ansible-inventory -i patchmon_inventory.yml --list
```

Example output:

```
{
  "_meta": {
    "hostvars": {
      "server1.example.com": { "ansible_host": "192.168.1.10" },
      "server2.example.com": { "ansible_host": "192.168.1.11" }
    }
  },
  "all": {
    "children": ["ungrouped", "web_servers", "db_servers", "production"]
  },
  "db_servers":   { "hosts": ["server2.example.com"] },
  "production":   { "hosts": ["server1.example.com", "server2.example.com"] },
  "web_servers":  { "hosts": ["server1.example.com"] }
}
```

Target specific groups

```
ansible-playbook -i patchmon_inventory.yml playbook.yml --limit web_servers
ansible-playbook -i patchmon_inventory.yml playbook.yml --limit production
```

Filtering at the API level

The scoped API `/api/v1/api/hosts` also accepts a `?hostgroup=` query parameter. If you want a plugin invocation that only returns, say, the `production` group, set:

```
api_url: https://patchmon.example.com/api/v1/api/hosts/?hostgroup=production
```

This reduces the payload size and is handy when you have thousands of hosts and only want Ansible to see a subset.

Authentication and SSL

The plugin uses **HTTP Basic Authentication**. The `Authorization` header it sends is `Basic base64(api_key:api_secret)`.

SSL certificate verification is on by default (`verify_ssl: true`). Disable it only when testing against an internal instance with a self-signed certificate, and never in production.

Troubleshooting

Test the API endpoint directly

```
curl -u "TOKEN_KEY:TOKEN_SECRET" https://patchmon.example.com/api/v1/api/hosts
```

You should get a JSON document with a `hosts` array. If not, double-check:

- The URL. PatchMon 2.x exposes the endpoint under `/api/v1/api/hosts` (note the double `/api/`).
- The credential. Ensure you're using the **Token Key** as the username and the **Token Secret** as the password, not a normal PatchMon user login.
- That the credential has the `host:read` scope (or is unscoped).
- That any IP allowlist on the credential includes the IP Ansible is calling from.

Debug the inventory

```
ansible-inventory -i patchmon_inventory.yml --list --debug
ansible-inventory -i patchmon_inventory.yml --list -vvv
```

Look for 401 Unauthorized (wrong credentials) or 403 Forbidden (missing scope / IP restriction) in the verbose output.

Common issues

Symptom	Likely cause	Fix
`401 Unauthorized`	Token key or secret wrong	Regenerate the credential in **Settings → Integrations**
`403 Forbidden` with "IP address not allowed"	Allowlist on the credential blocks the controller	Edit the credential and add the controller's public IP, or remove the allowlist
`403 Forbidden` with "Insufficient scope"	Credential lacks `host:read`	Edit the credential and tick the `host:read` scope
SSL cert error	Self-signed cert, or `verify_ssl: true` against an internal PKI	Install the CA chain on the controller, or temporarily set `verify_ssl: false`
Empty inventory	No hosts in PatchMon, or `?hostgroup=` filter matches nothing	Test with `curl` first; verify the group name spelling
JSON parsing errors	API URL points at the wrong path (e.g. `/api/v1/hosts` instead of `/api/v1/api/hosts`)	Correct the URL. The scoped API is under `/api/v1/api/`

Security best practices

Create a dedicated credential for Ansible. Don't reuse the same API key across multiple tools. If one is compromised, you want to revoke just that one.

Scope it tightly. `host:read` is enough for inventory; grant no more.

IP-restrict the credential to your Ansible controller(s).

Set an expiration on the credential and rotate it as part of your normal key rotation.

Vault the secret. Use `ansible-vault encrypt_string` or an environment variable. Never commit plaintext secrets to git.

Always use HTTPS and `verify_ssl: true` in production.

Contributing

Pull requests are welcome on PatchMon-ansible (https://github.com/PatchMon/PatchMon-ansible). Issues and feature requests can be filed at PatchMon-ansible/issues (https://github.com/PatchMon/PatchMon-ansible/issues).

Chapter 4: Proxmox LXC Auto-Enrollment Guide

Overview

PatchMon's Proxmox Auto-Enrollment feature enables you to automatically discover and enroll LXC containers from your Proxmox hosts into PatchMon for centralized patch management. This eliminates manual host registration and ensures comprehensive coverage of your Proxmox infrastructure.

What It Does

Automatically discovers running LXC containers on Proxmox hosts

Bulk enrolls containers into PatchMon without manual intervention

Installs agents inside each container automatically

Assigns to host groups based on token configuration

Tracks enrollment with full audit logging

Key Benefits

Zero-Touch Enrollment - Run once, enroll all containers

Secure by Design - Token-based authentication with hashed secrets

Rate Limited - Prevents abuse with per-day host limits

IP Restricted - Optional IP whitelisting for enhanced security

Fully Auditable - Tracks who enrolled what and when

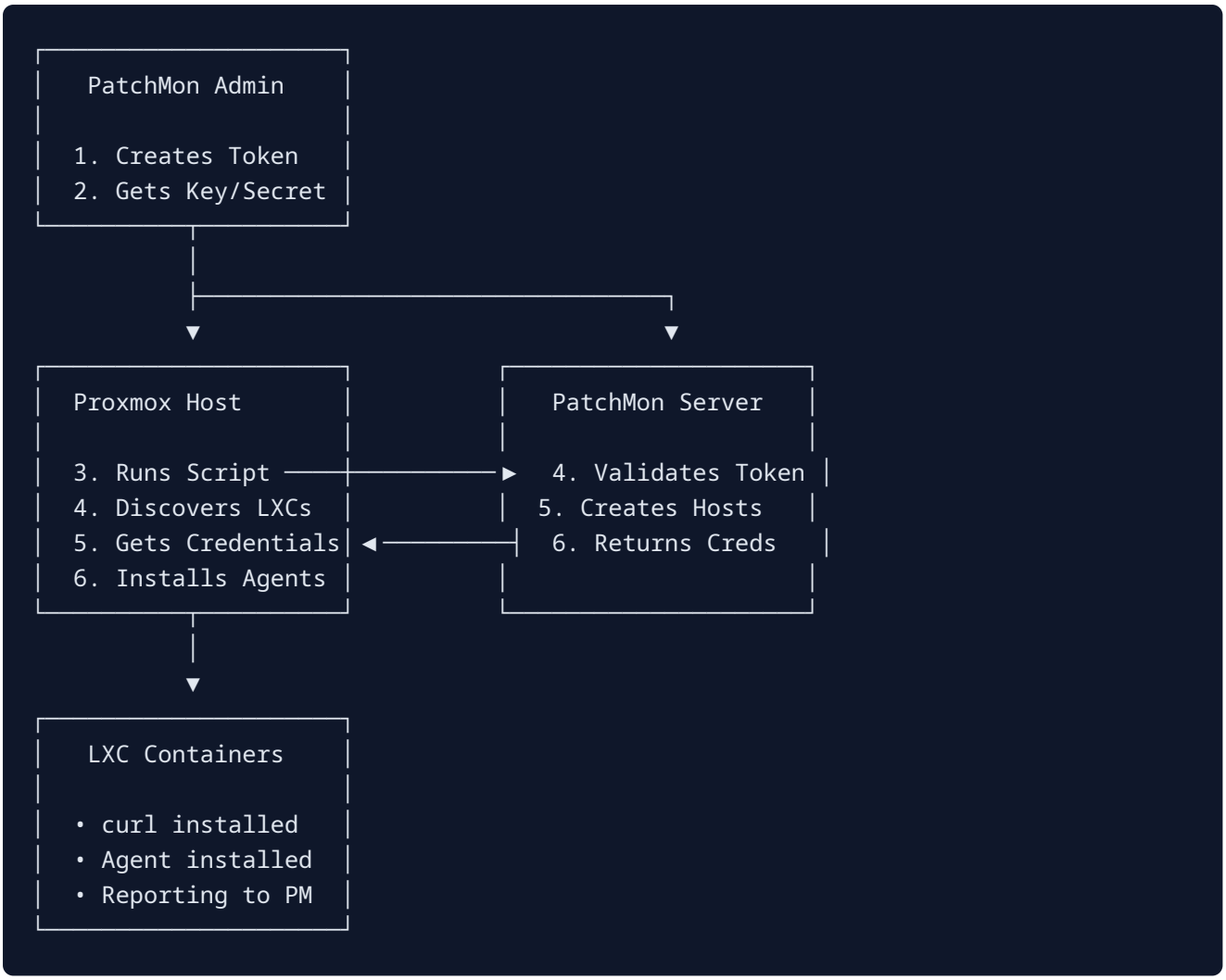
Safe to Rerun - Already-enrolled containers are automatically skipped

Table of Contents

- How It Works
- Prerequisites
- Quick Start
- Step-by-Step Setup
- Usage Examples
- Configuration Options
- Security Best Practices
- Troubleshooting
- Advanced Usage
- API Reference

How It Works

Architecture Overview



Enrollment Process (Step by Step)

Admin creates auto-enrollment token in PatchMon UI

- Configures rate limits, IP restrictions, host group assignment
- Receives `token_key` and `token_secret` (shown only once!)

Admin runs enrollment script on Proxmox host

- Script authenticated with auto-enrollment token
- Discovers all running LXC containers using `pct list`

For each container, the script:

- Gathers hostname, IP address, OS information, machine ID
- Calls PatchMon API to create host entry
- Receives unique `api_id` and `api_key` for that container
- Uses `pct exec` to enter the container
- Installs curl if missing
- Downloads and runs PatchMon agent installer
- Agent authenticates with container-specific credentials

Containers appear in PatchMon with full patch tracking enabled

Two-Tier Security Model

1. Auto-Enrollment Token (Script → PatchMon)

Purpose: Create new host entries
Scope: Limited to enrollment operations only
Storage: Secret is hashed in database
Lifespan: Reusable until revoked/expired
Security: Rate limits + IP restrictions

2. Host API Credentials (Agent → PatchMon)

Purpose: Report patches, send data, receive commands
Scope: Per-host unique credentials
Storage: API key is hashed (bcrypt) in database
Lifespan: Permanent for that host
Security: Host-specific, can be regenerated

Why This Matters:

Compromised enrollment token ≠ compromised hosts
Compromised host credential ≠ compromised enrollment
Revoked enrollment token = no new enrollments (existing hosts unaffected)
Lost credentials = create new token, don't affect existing infrastructure

Prerequisites

PatchMon Server Requirements

PatchMon version with auto-enrollment support
Admin user with "Manage Settings" permission
Network accessible from Proxmox hosts

Proxmox Host Requirements

Proxmox VE installed and running
One or more LXC containers (VMs not supported)
Root access to Proxmox host
Network connectivity to PatchMon server
Required commands: `pct`, `curl`, `jq`, `bash`

Container Requirements

- Running state (stopped containers are skipped)
- Debian-based or RPM-based Linux distribution
- Network connectivity to PatchMon server
- Package manager (apt/yum/dnf) functional

Network Requirements

Source	Destination	Port	Protocol	Purpose
Proxmox Host	PatchMon Server	443 (HTTPS)	TCP	Enrollment API calls
LXC Containers	PatchMon Server	443 (HTTPS)	TCP	Agent installation & reporting

Firewall Notes:

- Outbound only connections (no inbound ports needed)
- HTTPS recommended (HTTP supported for internal networks)
- Self-signed certificates supported with `-k` flag

Quick Start

1. Create Token (In PatchMon UI)

- Go to **Settings → Integrations → Auto-Enrollment & API** tab
- Click **"New Token"**
- Configure:
  - **Name**: "Production Proxmox"
  - **Max Hosts/Day**: 100
  - **Host Group**: Select target group
  - **IP Restriction**: Your Proxmox host IP
- **Save credentials immediately** (shown only once!)

2. One-Line Enrollment (On Proxmox Host)

```
curl -s "https://patchmon.example.com/api/v1/auto-enrollment/script?type=proxmox-lxc&token_key=YOUR_KEY&token_secret=YOUR_SECRET" | bash
```

That's it! All running LXC containers will be enrolled and the PatchMon agent installed.

3. Verify in PatchMon

- Go to **Hosts** page
- See your containers listed with "pending" status

Agent connects automatically after installation (usually within seconds)

Status changes to "active" with package data

Step-by-Step Setup

Step 1: Create Auto-Enrollment Token

Via PatchMon Web UI

Log in to PatchMon as an administrator

Navigate to Settings

```
Dashboard → Settings → Integrations → Auto-Enrollment & API tab
```

Click "New Token" button

Fill in token details:

Field	Value	Required	Description
Token Name	`Proxmox Production`	Yes	Descriptive name for this token
Max Hosts Per Day	`100`	Yes	Rate limit (1-1000)
Default Host Group	`Proxmox LXC`	No	Auto-assign enrolled hosts
Allowed IP Addresses	`192.168.1.10`	No	Comma-separated IPs
Expiration Date	`2027-01-01`	No	Auto-disable after date

Click "Create Token"

CRITICAL: Save Credentials Now!

You'll see a success modal with:

```
Token Key:    patchmon_ae_a1b2c3d4e5f6...
Token Secret: 8f7e6d5c4b3a2f1e0d9c8b7a...
```

Copy both values immediately! They cannot be retrieved later.

Pro Tip: Copy the one-line installation command shown in the modal - it has credentials pre-filled.

Step 2: Prepare Proxmox Host

Install Required Dependencies

```
# SSH to your Proxmox host
ssh root@proxmox-host

# Install jq (JSON processor)
apt-get update && apt-get install -y jq curl

# Verify installations
which pct jq curl
# Should show paths for all three commands
```

Download Enrollment Script

Method A: Direct Download from PatchMon (Recommended)

```
# Download with credentials embedded (copy from PatchMon UI)
curl -s "https://patchmon.example.com/api/v1/auto-enrollment/script?type=proxmox-
lxc&token_key=YOUR_KEY&token_secret=YOUR_SECRET" \
    -o /root/proxmox_auto_enroll.sh

chmod +x /root/proxmox_auto_enroll.sh
```

Method B: Manual Configuration

```
# Download script template
cd /root
wget
https://raw.githubusercontent.com/PatchMon/PatchMon/main/agents/proxmox_auto_enrol
l.sh
chmod +x proxmox_auto_enroll.sh

# Edit configuration
nano proxmox_auto_enroll.sh

# Update these lines:
PATCHMON_URL="https://patchmon.example.com"
AUTO_ENROLLMENT_KEY="patchmon_ae_your_key_here"
AUTO_ENROLLMENT_SECRET="your_secret_here"
```

Step 3: Test with Dry Run

Always test first!

```
# Dry run shows what would happen without making changes
DRY_RUN=true ./proxmox_auto_enroll.sh
```

Expected output:

```
[INFO] Found 5 LXC container(s)
[INFO] Processing LXC 100: webserver (status: running)
[INFO]   [DRY RUN] Would enroll: proxmox-webserver
[INFO] Processing LXC 101: database (status: running)
[INFO]   [DRY RUN] Would enroll: proxmox-database
...
[INFO] Successfully Enrolled:  5 (dry run)
```

Step 4: Run Actual Enrollment

```
# Enroll all containers
./proxmox_auto_enroll.sh
```

Monitor the output:

- Green `[SUCCESS]` = Container enrolled and agent installed
- Yellow `[WARN]` = Container skipped (already enrolled or stopped)
- Red `[ERROR]` = Failure (check troubleshooting section)

Step 5: Verify in PatchMon

- **Go to Hosts page** in PatchMon UI
- **Look for newly enrolled containers** (names prefixed with "proxmox-")
- **Initial status is "pending"** (normal!)
- **Agent connects automatically** after installation (usually within seconds)
- **Status changes to "active"** with package data populated

Troubleshooting: If status stays "pending" after a couple of minutes, see Agent Not Reporting section.

Usage Examples

Basic Enrollment

```
# Enroll all running LXC containers
./proxmox_auto_enroll.sh
```

Dry Run Mode

```
# Preview what would be enrolled (no changes made)
DRY_RUN=true ./proxmox_auto_enroll.sh
```

Debug Mode

```
# Show detailed logging for troubleshooting
DEBUG=true ./proxmox_auto_enroll.sh
```

Custom Host Prefix

```
# Prefix container names (e.g., "prod-webserver" instead of "webserver")
HOST_PREFIX="prod-" ./proxmox_auto_enroll.sh
```

Force Install Mode (Broken Packages)

If containers have broken packages (CloudPanel, WHM, cPanel, etc.) that block `apt-get`:

```
# Bypass broken packages during agent installation
FORCE_INSTALL=true ./proxmox_auto_enroll.sh
```

Or use the force parameter when downloading:

```
curl -s "https://patchmon.example.com/api/v1/auto-enrollment/script?type=proxmox-lxc&token_key=KEY&token_secret=SECRET&force=true" | bash
```

What force mode does:

- Skips `apt-get update` if broken packages detected
- Only installs missing critical tools (jq, curl, bc)
- Uses `--fix-broken --yes` flags safely
- Validates installations before proceeding

Scheduled Enrollment (Cron)

Automatically enroll new containers on a schedule. Since cron runs with a minimal environment (limited `PATH`, no user variables), you need to ensure the crontab has the correct environment set up for the script to find required commands like `pct`, `curl`, and `jq`.

Setting Up the Crontab

Edit the root crontab:

```
crontab -e
```

Add the following. The `PATH` and environment variables at the top are essential - without them the script will fail because cron does not inherit your shell's environment:

```
# === PatchMon Auto-Enrollment Environment ===
# Cron uses a minimal PATH by default (/usr/bin:/bin). The enrollment script
# requires pct, curl, and jq which may live in /usr/sbin or other paths.
# Set a full PATH so all commands are found.
SHELL=/bin/bash
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin

# Enrollment credentials (required by the script)
PATCHMON_URL=https://patchmon.example.com
AUTO_ENROLLMENT_KEY=patchmon_ae_your_key_here
AUTO_ENROLLMENT_SECRET=your_secret_here

# Optional overrides
# HOST_PREFIX=proxmox-
# FORCE_INSTALL=false
# CURL_FLAGS=-sk

# === Schedule ===
# Run daily at 2 AM
0 2 * * * /root/proxmox_auto_enroll.sh >> /var/log/patchmon-enroll.log 2>&1

# Or hourly for dynamic environments where containers are created frequently
# 0 * * * * /root/proxmox_auto_enroll.sh >> /var/log/patchmon-enroll.log 2>&1
```

Why This Matters

Cron does not load your interactive shell profile (`~/.bashrc` , `~/.profile` , etc.). This means:

What cron is missing	Impact	Fix
`PATH` only includes `/usr/bin:/bin`	`pct` not found (lives in `/usr/sbin`)	Set `PATH` at top of crontab
No exported variables	`PATCHMON_URL` , credentials are empty	Define them in crontab or use a wrapper
No TTY	Colour output codes may cause log clutter	Redirect to log file with `2>&1`

Alternative: Wrapper Script

If you prefer not to put credentials in the crontab, create a wrapper script instead:

```
cat > /root/patchmon_enroll_cron.sh << 'EOF'
#!/bin/bash
# Wrapper that sets the environment for cron execution

export PATH="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"
export PATCHMON_URL="https://patchmon.example.com"
export AUTO_ENROLLMENT_KEY="patchmon_ae_your_key_here"
export AUTO_ENROLLMENT_SECRET="your_secret_here"
# export HOST_PREFIX="proxmox-"
# export CURL_FLAGS="-sk"

/root/proxmox_auto_enroll.sh
EOF

chmod 700 /root/patchmon_enroll_cron.sh
```

Then reference the wrapper in crontab:

```
0 2 * * * /root/patchmon_enroll_cron.sh >> /var/log/patchmon-enroll.log 2>&1
```

Make sure the wrapper script is only readable by root (`chmod 700`) since it contains secrets.

Log Rotation

For long-running cron schedules, consider adding log rotation to prevent unbounded log growth:

```
cat > /etc/logrotate.d/patchmon-enroll << 'EOF'
/var/log/patchmon-enroll.log {
    weekly
    rotate 4
    compress
    missingok
    notifempty
}
EOF
```

Verifying Cron is Working

```
# Check the cron job is registered
crontab -l | grep patchmon

# Check recent cron execution logs
grep patchmon /var/log/syslog | tail -n 20

# Check enrollment log output
tail -f /var/log/patchmon-enroll.log
```

Already-enrolled containers are automatically skipped on each run, so there is no risk of duplicates or errors from repeated execution.

Multi-Environment Setup

```
# Production environment (uses prod token)
export PATCHMON_URL="https://patchmon.example.com"
export AUTO_ENROLLMENT_KEY="patchmon_ae_prod_..."
export AUTO_ENROLLMENT_SECRET="prod_secret..."
export HOST_PREFIX="prod-"
./proxmox_auto_enroll.sh

# Development environment (uses dev token with different host group)
export AUTO_ENROLLMENT_KEY="patchmon_ae_dev_..."
export AUTO_ENROLLMENT_SECRET="dev_secret..."
export HOST_PREFIX="dev-"
./proxmox_auto_enroll.sh
```

Configuration Options

Environment Variables

All configuration can be set via environment variables:

Variable	Default	Description	Example
`PATCHMON_URL`	Required	PatchMon server URL	`https://patchmon.example.com`
`AUTO_ENROLLMENT_KEY`	Required	Token key from PatchMon	`patchmon_ae_abc123...`
`AUTO_ENROLLMENT_SECRET`	Required	Token secret from PatchMon	`def456ghi789...`
`CURL_FLAGS`	`-s`	Curl options	`-sk` (for self-signed SSL)
`DRY_RUN`	`false`	Preview mode (no changes)	`true` / `false`
`HOST_PREFIX`	`""`	Prefix for host names	`proxmox-`, `prod-`, etc.
`FORCE_INSTALL`	`false`	Bypass broken packages	`true` / `false`
`DEBUG`	`false`	Enable debug logging	`true` / `false`

Script Configuration Section

Or edit the script directly:

```
# ===== CONFIGURATION =====
PATCHMON_URL="${PATCHMON_URL:-https://patchmon.example.com}"
AUTO_ENROLLMENT_KEY="${AUTO_ENROLLMENT_KEY:-your_key_here}"
AUTO_ENROLLMENT_SECRET="${AUTO_ENROLLMENT_SECRET:-your_secret_here}"
CURL_FLAGS="${CURL_FLAGS:--s}"
DRY_RUN="${DRY_RUN:-false}"
HOST_PREFIX="${HOST_PREFIX:-}"
FORCE_INSTALL="${FORCE_INSTALL:-false}"
```

Token Configuration (PatchMon UI)

Configure tokens in **Settings → Integrations → Auto-Enrollment & API**:

General Settings:

- **Token Name**: Descriptive identifier
- **Active Status**: Enable/disable without deleting
- **Expiration Date**: Auto-disable after date

Security Settings:

- **Max Hosts Per Day**: Rate limit (resets daily at midnight)
- **Allowed IP Addresses**: Comma-separated IP whitelist
- **Default Host Group**: Auto-assign enrolled hosts

Usage Statistics:

- **Hosts Created Today**: Current daily count
- **Last Used**: Timestamp of most recent enrollment
- **Created By**: Admin user who created token
- **Created At**: Token creation timestamp

Security Best Practices

Token Management

- **Store Securely**
  - Save credentials in password manager (1Password, LastPass, etc.)
  - Never commit to version control
  - Use environment variables or secure config management (Vault)
- **Principle of Least Privilege**

Create separate tokens for prod/dev/staging
Use different tokens for different Proxmox clusters
Set appropriate rate limits per environment

Regular Rotation

Rotate tokens every 90 days
Disable unused tokens immediately
Monitor token usage for anomalies

IP Restrictions

Always set `allowed_ip_ranges` in production
Update if Proxmox host IPs change
Use VPN/private network IPs when possible

Expiration Dates

Set expiration for temporary/testing tokens
Review and extend before expiration
Delete expired tokens to reduce attack surface

Network Security

Use HTTPS

Always use encrypted connections in production
Use valid SSL certificates (avoid `-k` flag)
Self-signed OK for internal/testing environments

Network Segmentation

Run enrollment over private network if possible
Use proper firewall rules
Restrict PatchMon server access to known IPs

Access Control

Admin Permissions

Only admins with "Manage Settings" can create tokens
Regular users cannot see token secrets
Use role-based access control (RBAC)

Audit Logging

Monitor token creation/deletion in PatchMon logs

- Track enrollment activity per token
- Review host notes for enrollment source

- **Container Security**
  - Ensure containers have minimal privileges
  - Don't run enrollment as unprivileged user
  - Use unprivileged containers where possible (enrollment still works)

Incident Response

If a token is compromised:

1. **Immediately disable** the token in PatchMon UI
   - Settings → Integrations → Auto-Enrollment & API → Toggle "Disable"
2. **Review recently enrolled hosts**
   - Check host notes for token name and enrollment date
   - Verify all recent enrollments are legitimate
   - Delete any suspicious hosts
3. **Create new token**
   - Generate new credentials
   - Update Proxmox script with new credentials
   - Test enrollment with dry run
4. **Investigate root cause**
   - How were credentials exposed?
   - Update procedures to prevent recurrence
   - Consider additional security measures
5. **Delete old token**
   - After verifying new token works
   - Document incident in change log

Troubleshooting

Common Errors and Solutions

Error: "pct command not found"
Symptom:

```
[ERROR] This script must run on a Proxmox host (pct command not found)
```

Cause: Script is running on a non-Proxmox machine

Solution:

```
# SSH to Proxmox host first
ssh root@proxmox-host
cd /root
./proxmox_auto_enroll.sh
```

Error: "Auto-enrollment credentials required"
Symptom:

```
[ERROR] Failed to enroll hostname - HTTP 401
Response: {"error":"Auto-enrollment credentials required"}
```

Cause: The `X-Auto-Enrollment-Key` and/or `X-Auto-Enrollment-Secret` headers are missing from the request

Solution:

- Verify the script has `AUTO_ENROLLMENT_KEY` and `AUTO_ENROLLMENT_SECRET` set
- Check for extra spaces/newlines in credentials
- Ensure token_key starts with `patchmon_ae_`
- Regenerate token if credentials lost

```
# Test credentials manually
curl -X POST \
  -H "X-Auto-Enrollment-Key: YOUR_KEY" \
  -H "X-Auto-Enrollment-Secret: YOUR_SECRET" \
  -H "Content-Type: application/json" \
  -d '{"friendly_name":"test","machine_id":"test"}' \
  https://patchmon.example.com/api/v1/auto-enrollment/enroll
```

Error: "Invalid or inactive token" / "Invalid token secret"
Symptom:

```
[ERROR] Failed to enroll hostname - HTTP 401
Response: {"error":"Invalid or inactive token"}
```

or

```
[ERROR] Failed to enroll hostname - HTTP 401
Response: {"error":"Invalid token secret"}
```

Cause: Token key not found or disabled (`Invalid or inactive token`), or secret doesn't match (`Invalid token secret`), or token has expired (`Token expired`)

Solution:

- Check token status in PatchMon UI (Settings → Integrations)
- Enable if disabled
- Extend expiration if expired
- Verify the secret matches the one shown when the token was created
- Create new token if credentials are lost (secrets cannot be retrieved)

Error: "Rate limit exceeded"
Symptom:

```
[ERROR] Rate limit exceeded - maximum hosts per day reached
```

Cause: Token's `max_hosts_per_day` limit reached

Solution:

```
# Option 1: Wait until tomorrow (limit resets at midnight)
date
# Check current time, wait until 00:00

# Option 2: Increase limit in PatchMon UI
# Settings → Integrations → Edit Token → Max Hosts Per Day: 200

# Option 3: Create additional token for large enrollments
```

Error: "IP address not authorized"
Symptom:

```
[ERROR] Failed to enroll hostname - HTTP 403
Response: {"error":"IP address not authorized for this token"}
```

Cause: Proxmox host IP not in token's `allowed_ip_ranges`

Solution:

- Find your Proxmox host IP:

```
ip addr show | grep 'inet ' | grep -v 127.0.0.1
```

Update token in PatchMon UI:

Settings → Integrations → Edit Token

Allowed IP Addresses: Add your IP

Or remove IP restriction entirely (not recommended for production)

Error: "jq: command not found"
Symptom:

```
[ERROR] Required command 'jq' not found. Please install it first.
```

Cause: Missing dependency

Solution:

```
# Debian/Ubuntu
apt-get update && apt-get install -y jq

# CentOS/RHEL
yum install -y jq

# Alpine
apk add --no-cache jq
```

Error: "Failed to install agent in container"
Symptom:

```
[WARN] Failed to install agent in container-name (exit: 1)
Install output: E: Unable to locate package curl
```

Cause: Agent installation failed inside LXC container

Solutions:

A. Network connectivity issue:

```
# Test from Proxmox host
pct exec 100 -- ping -c 3 patchmon.example.com

# Test from inside container
pct enter 100
curl -I https://patchmon.example.com
exit
```

B. Package manager issue:

```
# Enter container
pct enter 100

# Update package lists
apt-get update
# or
yum makecache

# Try manual agent install
curl https://patchmon.example.com/api/v1/hosts/install \
  -H "X-API-ID: patchmon_xxx" \
  -H "X-API-KEY: xxx" | bash
```

C. Unsupported OS:

- Agent supports: Ubuntu, Debian, CentOS, RHEL, Rocky Linux, AlmaLinux, Alpine
- Check `/etc/os-release` in container
- Manually install on other distributions

D. Broken packages (use force mode):

```
FORCE_INSTALL=true ./proxmox_auto_enroll.sh
```

Error: SSL Certificate Problems

Symptom:

```
curl: (60) SSL certificate problem: self signed certificate
```

Cause: Self-signed certificate on PatchMon server

Solution:

```
# Use -k flag to skip certificate verification
export CURL_FLAGS="-sk"
./proxmox_auto_enroll.sh
```

Better solution: Install valid SSL certificate on PatchMon server using Let's Encrypt or corporate CA

Warning: Container Already Enrolled
Symptom:

```
[INFO] ✓ Host already enrolled and agent ping successful - skipping enrollment
```

Cause: The script detected an existing agent configuration (`/etc/patchmon/config.yml` and `/etc/patchmon/credentials.yml`) inside the container and the agent successfully pinged the PatchMon server.

This is normal! The script safely skips already-enrolled hosts. No action needed.

If you need to re-enroll:

1. Delete host from PatchMon UI (Hosts page)
2. Remove agent config inside the container: `pct exec <vmid> -- rm -rf /etc/patchmon/`
3. Rerun enrollment script

Agent Not Reporting

If containers show "pending" status after enrollment:

1. Check agent service is running:

```
pct enter 100

# For systemd-based containers
systemctl status patchmon-agent.service

# For OpenRC-based containers (Alpine)
rc-service patchmon-agent status

# For containers without init systems (crontab fallback)
ps aux | grep patchmon-agent
```

2. Check agent files exist:

```
ls -la /etc/patchmon/
# Should show: config.yml and credentials.yml

ls -la /usr/local/bin/patchmon-agent
# Should show the agent binary
```

3. Check agent logs:

```
# Systemd journal logs
journalctl -u patchmon-agent.service --no-pager -n 50

# Or check the agent log file
cat /etc/patchmon/logs/patchmon-agent.log
```

4. Test agent connectivity:

```
/usr/local/bin/patchmon-agent ping
# Should show success if credentials and connectivity are valid
```

5. Verify credentials:

```
cat /etc/patchmon/credentials.yml
# Should show api_id and api_key

cat /etc/patchmon/config.yml
# Should show patchmon_server URL
```

6. Restart the agent service:

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

# OpenRC
rc-service patchmon-agent restart
```

Debug Mode

Enable detailed logging:

```
DEBUG=true ./proxmox_auto_enroll.sh
```

Debug output includes:

- API request/response bodies

- Container command execution details
- Detailed error messages
- curl verbose output

Getting Help

If issues persist:

- **Check PatchMon server logs:**

```
tail -f /path/to/patchmon/backend/logs/error.log
```

- **Create GitHub issue** with:
  - PatchMon version
  - Proxmox version
  - Script output (redact credentials!)
  - Debug mode output
  - Server logs (if accessible)
- **Join Discord community** for real-time support

Advanced Usage

Selective Enrollment

Enroll only specific containers:

```
# Only enroll containers 100-199
nano proxmox_auto_enroll.sh

# Add after line "while IFS= read -r line; do"
vmid=$(echo "$line" | awk '{print $1}')
if [[ $vmid -lt 100 ]] || [[ $vmid -gt 199 ]]; then
    continue
fi
```

Or use container name filtering:

```
# Only enroll containers with "prod" in name
if [[ ! "$name" =~ prod ]]; then
    continue
fi
```

Custom Host Naming

Advanced naming strategies:

```
# Include Proxmox node name
HOST_PREFIX="$(hostname)-"
# Result: proxmox01-webserver, proxmox02-database

# Include datacenter/location
HOST_PREFIX="dc1-"
# Result: dc1-webserver, dc1-database

# Include environment and node
HOST_PREFIX="prod-$(hostname | cut -d. -f1)-"
# Result: prod-px01-webserver
```

Multi-Node Proxmox Cluster

For Proxmox clusters with multiple nodes:

Option 1: Same token, different prefix per node

```
# On node 1
HOST_PREFIX="node1-" ./proxmox_auto_enroll.sh

# On node 2
HOST_PREFIX="node2-" ./proxmox_auto_enroll.sh
```

Option 2: Different tokens per node

Create token for each node with different default host groups

Node 1 → "Proxmox Node 1" group

Node 2 → "Proxmox Node 2" group

Option 3: Centralized automation

```
#!/bin/bash
# central_enroll.sh

NODES=(
  "root@proxmox01.example.com"
  "root@proxmox02.example.com"
  "root@proxmox03.example.com"
)

for node in "${NODES[@]}"; do
  echo "Enrolling containers from $node..."
  ssh "$node" "bash /root/proxmox_auto_enroll.sh"
done
```

Integration with Infrastructure as Code

Ansible Playbook:

```
---
- name: Enroll Proxmox LXC containers in PatchMon
  hosts: proxmox_hosts
  become: yes
  tasks:
    - name: Install dependencies
      apt:
        name:
          - curl
          - jq
        state: present

    - name: Download enrollment script
      get_url:
        url: "{{ patchmon_url }}/api/v1/auto-enrollment/script?type=proxmox-
lxc&token_key={{ token_key }}&token_secret={{ token_secret }}"
        dest: /root/proxmox_auto_enroll.sh
        mode: '0700'

    - name: Run enrollment
      command: /root/proxmox_auto_enroll.sh
      register: enrollment_output

    - name: Show enrollment results
      debug:
        var: enrollment_output.stdout_lines
```

Terraform (with null_resource):

```
resource "null_resource" "patchmon_enrollment" {
  triggers = {
    cluster_instance_ids = join(",", proxmox_lxc.containers.*.vmid)
  }

  provisioner "remote-exec" {
    connection {
      host = var.proxmox_host
      user = "root"
      private_key = file(var.ssh_key_path)
    }

    inline = [
      "apt-get install -y jq",
      "curl -s '${var.patchmon_url}/api/v1/auto-enrollment/script?type=proxmox-
lxc&token_key=${var.token_key}&token_secret=${var.token_secret}' | bash"
    ]
  }
}
```

Bulk API Enrollment

For very large deployments (100+ containers), use the bulk API endpoint directly:

```
#!/bin/bash
# bulk_enroll.sh

# Gather all container info
containers_json=$(pct list | tail -n +2 | while read -r line; do
  vmid=$(echo "$line" | awk '{print $1}')
  name=$(echo "$line" | awk '{print $3}')

  echo "{\"friendly_name\":\"$name\",\"machine_id\":\"proxmox-lxc-$vmid\"}"
done | jq -s '.')

# Send bulk enrollment request
curl -X POST \
  -H "X-Auto-Enrollment-Key: $AUTO_ENROLLMENT_KEY" \
  -H "X-Auto-Enrollment-Secret: $AUTO_ENROLLMENT_SECRET" \
  -H "Content-Type: application/json" \
  -d "{\"hosts\":$containers_json}" \
  "$PATCHMON_URL/api/v1/auto-enrollment/enroll/bulk"
```

Benefits:

- Single API call for all containers
- Faster for 50+ containers
- Partial success supported (individual failures don't block others)

Limitations:

- Max 50 hosts per request
- Does not install agents (must be done separately)
- Less detailed error reporting per host

Webhook-Triggered Enrollment

Trigger enrollment from PatchMon webhook (requires custom setup):

```
#!/bin/bash
# webhook_listener.sh

# Simple webhook listener
while true; do
  # Listen for webhook on port 9000
  nc -l -p 9000 -c 'echo -e "HTTP/1.1 200 OK\n\n"; /root/proxmox_auto_enroll.sh'
done
```

Then configure PatchMon (or monitoring system) to call webhook when conditions are met.

API Reference

Admin Endpoints (Authentication Required)

All admin endpoints require JWT authentication:

```
Authorization: Bearer <jwt_token>
```

Create Token
Endpoint: `POST /api/v1/auto-enrollment/tokens`

Request:

```
{
  "token_name": "Proxmox Production",
  "max_hosts_per_day": 100,
  "default_host_group_id": "uuid",
  "allowed_ip_ranges": ["192.168.1.10", "10.0.0.5"],
  "expires_at": "2026-12-31T23:59:59Z",
  "metadata": {
    "integration_type": "proxmox-lxc",
    "environment": "production"
  }
}
```

Response: `201 Created`

```
{
  "message": "Auto-enrollment token created successfully",
  "token": {
    "id": "uuid",
    "token_name": "Proxmox Production",
    "token_key": "patchmon_ae_abc123...",
    "token_secret": "def456...",  // Only shown here!
    "max_hosts_per_day": 100,
    "default_host_group": {
      "id": "uuid",
      "name": "Proxmox LXC",
      "color": "#3B82F6"
    },
    "created_by": {
      "id": "uuid",
      "username": "admin",
      "first_name": "John",
      "last_name": "Doe"
    },
    "expires_at": "2026-12-31T23:59:59Z"
  },
  "warning": "Save the token_secret now - it cannot be retrieved later!"
}
```

List Tokens

Endpoint: `GET /api/v1/auto-enrollment/tokens`

Response: `200 OK`

```
[
  {
    "id": "uuid",
    "token_name": "Proxmox Production",
    "token_key": "patchmon_ae_abc123...",
    "is_active": true,
    "allowed_ip_ranges": ["192.168.1.10"],
    "max_hosts_per_day": 100,
    "hosts_created_today": 15,
    "last_used_at": "2025-10-11T14:30:00Z",
    "expires_at": "2026-12-31T23:59:59Z",
    "created_at": "2025-10-01T10:00:00Z",
    "default_host_group_id": "uuid",
    "metadata": {"integration_type": "proxmox-lxc"},
    "host_groups": {
      "id": "uuid",
      "name": "Proxmox LXC",
      "color": "#3B82F6"
    },
    "users": {
      "id": "uuid",
      "username": "admin",
      "first_name": "John",
      "last_name": "Doe"
    }
  }
]
```

Get Token Details

Endpoint: `GET /api/v1/auto-enrollment/tokens/:tokenId`

Response: `200 OK` (same structure as single token in list)

Update Token

Endpoint: `PATCH /api/v1/auto-enrollment/tokens/:tokenId`

Request:

```
{
  "is_active": false,
  "max_hosts_per_day": 200,
  "allowed_ip_ranges": ["192.168.1.0/24"],
  "expires_at": "2027-01-01T00:00:00Z"
}
```

Response: `200 OK`

```
{
  "message": "Token updated successfully",
  "token": { /* updated token object */ }
}
```

Delete Token
Endpoint: `DELETE /api/v1/auto-enrollment/tokens/:tokenId`

Response: `200 OK`

```
{
  "message": "Auto-enrollment token deleted successfully",
  "deleted_token": {
    "id": "uuid",
    "token_name": "Proxmox Production"
  }
}
```

Enrollment Endpoints (Token Authentication)

Authentication via headers:

```
X-Auto-Enrollment-Key: patchmon_ae_abc123...
X-Auto-Enrollment-Secret: def456...
```

Download Enrollment Script
Endpoint: `GET /api/v1/auto-enrollment/script`

Query Parameters:

- `type` (required): Script type (`proxmox-lxc` or `direct-host`)
- `token_key` (required): Auto-enrollment token key
- `token_secret` (required): Auto-enrollment token secret
- `force` (optional): `true` to enable force install mode

Example:

```
curl "https://patchmon.example.com/api/v1/auto-enrollment/script?type=proxmox-
lxc&token_key=KEY&token_secret=SECRET&force=true"
```

Response: `200 OK` (bash script with credentials injected)

Enroll Single Host
Endpoint: `POST /api/v1/auto-enrollment/enroll`

Request:

```
{
  "friendly_name": "webserver",
  "machine_id": "proxmox-lxc-100-abc123",
  "metadata": {
    "vmid": "100",
    "proxmox_node": "proxmox01",
    "ip_address": "10.0.0.10",
    "os_info": "Ubuntu 22.04 LTS"
  }
}
```

Response: `201 Created`

```
{
  "message": "Host enrolled successfully",
  "host": {
    "id": "uuid",
    "friendly_name": "webserver",
    "api_id": "patchmon_abc123",
    "api_key": "def456ghi789",
    "host_group": {
      "id": "uuid",
      "name": "Proxmox LXC",
      "color": "#3B82F6"
    },
    "status": "pending"
  }
}
```

Error Responses:

> ***Note:*** *The API does not perform duplicate host checks. Duplicate prevention is handled client-side by the enrollment script, which checks for an existing agent configuration inside each container before calling the API.*

`429 Too Many Requests` - Rate limit exceeded:

```
{
  "error": "Rate limit exceeded",
  "message": "Maximum 100 hosts per day allowed for this token"
}
```

Bulk Enroll Hosts

Endpoint: `POST /api/v1/auto-enrollment/enroll/bulk`

Request:

```
{
  "hosts": [
    {
      "friendly_name": "webserver",
      "machine_id": "proxmox-lxc-100-abc123"
    },
    {
      "friendly_name": "database",
      "machine_id": "proxmox-lxc-101-def456"
    }
  ]
}
```

Limits:

- Minimum: 1 host
- Maximum: 50 hosts per request

Response: `201 Created`

```
{
  "message": "Bulk enrollment completed: 2 succeeded, 0 failed, 0 skipped",
  "results": {
    "success": [
      {
        "id": "uuid",
        "friendly_name": "webserver",
        "api_id": "patchmon_abc123",
        "api_key": "def456"
      },
      {
        "id": "uuid",
        "friendly_name": "database",
        "api_id": "patchmon_ghi789",
        "api_key": "jkl012"
      }
    ],
    "failed": [],
    "skipped": []
  }
}
```

FAQ

General Questions

Q: Can I use the same token for multiple Proxmox hosts?
A: Yes, as long as the combined enrollment count stays within `max_hosts_per_day` limit. Rate limits are per-token, not per-host.

Q: What happens if I run the script multiple times?
A: Already-enrolled containers are automatically skipped. The script checks for existing agent configuration inside each container and skips those where the agent is already installed and responsive. Safe to rerun!

Q: Can I enroll stopped LXC containers?
A: No, containers must be running. The script needs to execute commands inside the container to install the agent. Start containers before enrolling.

Q: Does this work with Proxmox VMs (QEMU)?
A: No, this script is LXC-specific and uses `pct exec` to enter containers. VMs require manual enrollment or a different automation approach (SSH-based).

Q: How do I unenroll a host?
A: Go to PatchMon UI → Hosts → Select host → Delete. The agent will stop reporting and the host record is removed from the database.

Q: Can I change the host group after enrollment?
A: Yes! In PatchMon UI → Hosts → Select host → Edit → Change host group.

Q: Can I see which hosts were enrolled by which token?
A: Yes, check the host "Notes" field in PatchMon. It includes the token name and enrollment timestamp.

Q: What if my Proxmox host IP address changes?
A: Update the token's `allowed_ip_ranges` in PatchMon UI (Settings → Integrations → Edit Token).

Q: Can I have multiple tokens with different host groups?
A: Yes! Create separate tokens for prod/dev/staging with different default host groups. Great for environment segregation.

Q: Is there a way to trigger enrollment from PatchMon GUI?
A: Not currently (would require inbound network access). The script must run on the Proxmox host. Future versions may support webhooks or agent-initiated enrollment.

Security Questions

Q: Are token secrets stored securely?
A: Yes, token secrets are hashed using bcrypt before storage. Only the hash is stored in the database, never the plain text.

Q: What happens if someone steals my auto-enrollment token?
A: They can create new hosts up to the rate limit, but cannot control existing hosts or access host data. Immediately disable the token in PatchMon UI if compromised.

Q: Can I audit who created which tokens?
A: Yes, each token stores the `created_by_user_id`. View in PatchMon UI or query the database.

Q: How does IP whitelisting work?
A: PatchMon checks the client IP from the HTTP request. If `allowed_ip_ranges` is configured, the IP must match one of the allowed ranges using CIDR notation (e.g., `192.168.1.0/24`). Single IP addresses are also supported (e.g., `192.168.1.10`).

Q: Can I use the same credentials for enrollment and agent communication?
A: No, they're separate. Auto-enrollment credentials create hosts. Each host gets unique API credentials for agent communication. This separation limits the blast radius of credential compromise.

Technical Questions

Q: Why does the agent require curl inside the container?
A: The agent script uses curl to communicate with PatchMon. The enrollment script automatically installs curl if missing.

Q: What Linux distributions are supported in containers?
A: Ubuntu, Debian, CentOS, RHEL, Rocky Linux, AlmaLinux, Alpine Linux. Any distribution with apt/yum/dnf/apk package managers.

Q: How much bandwidth does enrollment use?
A: Minimal. The script download is ~15KB, agent installation is ~50-100KB per container. Total: ~1-2MB for 10 containers.

Q: Can I run enrollment in parallel for faster processing?
A: Not recommended. The script processes containers sequentially to avoid overwhelming the PatchMon server. For 100+ containers, consider the bulk API endpoint.

Q: Does enrollment restart containers?
A: No, containers remain running. The agent is installed without reboots or service disruptions.

Q: What if the container doesn't have a hostname?
A: The script uses the container name from Proxmox as a fallback.

Q: Can I customize the agent installation?
A: Yes, modify the `install_url` in the enrollment script or use the PatchMon agent installation API parameters.

Troubleshooting Questions

Q: Why does enrollment fail with "dpkg was interrupted"?
A: Your container has broken packages. Use `FORCE_INSTALL=true` to bypass, or manually fix dpkg:

```
pct enter 100
dpkg --configure -a
apt-get install -f
```

Q: Why does the agent show "pending" status forever?
A: Agent likely can't reach PatchMon server. Check:

- Container network connectivity: `pct exec 100 -- ping patchmon.example.com`
- Agent service running: `pct exec 100 -- systemctl status patchmon-agent.service`
- Agent logs: `pct exec 100 -- journalctl -u patchmon-agent.service`

Q: Can I test enrollment without actually creating hosts?
A: Yes, use dry run mode: `DRY_RUN=true ./proxmox_auto_enroll.sh`

Q: How do I get more verbose output?
A: Use debug mode: `DEBUG=true ./proxmox_auto_enroll.sh`

Support and Resources

Documentation

- **PatchMon Documentation**: https://patchmon.net/docs (https://patchmon.net/docs)
- **API Reference**: https://patchmon.net/docs/api-reference (https://patchmon.net/docs/api-reference)
- **Agent Documentation**: https://patchmon.net/docs/patchmon-operator-guide#managing-the-patchmon-agent (https://patchmon.net/docs/patchmon-operator-guide#managing-the-patchmon-agent)

Community

- **Discord**: https://patchmon.net/discord (https://patchmon.net/discord)
- **GitHub Issues**: https://github.com/PatchMon/PatchMon/issues (https://github.com/PatchMon/PatchMon/issues)
- **GitHub Discussions**: https://github.com/PatchMon/PatchMon/discussions (https://github.com/PatchMon/PatchMon/discussions)

Professional Support

For enterprise support, training, or custom integrations:

- **Email**: support@patchmon.net
- **Website**: https://patchmon.net/support (https://patchmon.net/support)

Chapter 5: Auto-Enrollment API Documentation

Overview

PatchMon's auto-enrollment API enables automated device onboarding using tools like Ansible, Terraform, or custom scripts. It covers token management, host enrollment, and agent installation endpoints.

Table of Contents

- API Architecture
- Authentication
- Admin Endpoints
- Enrollment Endpoints
- Host Management Endpoints
- Ansible Integration Examples
- Error Handling
- Rate Limiting
- Security Considerations

API Architecture

Base URL Structure

```
https://your-patchmon-server.com/api/v1/
```

The API version is `v1` and is fixed in the server.

Endpoint Categories

Category	Path Prefix	Authentication	Purpose
Admin	`/auto-enrollment/tokens/*`	JWT (Bearer token)	Token management (CRUD)
Enrollment	`/auto-enrollment/*`	Token key + secret (headers)	Host enrollment & script download
Host	`/hosts/*`	API ID + key (headers)	Agent installation & data reporting

Two-Tier Security Model

Tier 1: Auto-Enrollment Token

- **Purpose**: Create new host entries via enrollment
- **Scope**: Limited to enrollment operations only
- **Authentication**: `X-Auto-Enrollment-Key` + `X-Auto-Enrollment-Secret` headers
- **Rate Limited**: Yes (configurable hosts per day per token)
- **Storage**: Secret is hashed (bcrypt) in the database

Tier 2: Host API Credentials

- **Purpose**: Agent communication (data reporting, updates, commands)
- **Scope**: Per-host unique credentials
- **Authentication**: `X-API-ID` + `X-API-KEY` headers
- **Rate Limited**: No (per-host)
- **Storage**: API key is hashed (bcrypt) in the database

Why two tiers?

- Compromised enrollment token does not compromise existing hosts
- Compromised host credential does not compromise enrollment
- Revoking an enrollment token stops new enrollments without affecting existing hosts

Authentication

Admin Endpoints (JWT)

All admin endpoints require a valid JWT Bearer token from an authenticated user with "Manage Settings" permission:

```
curl -H "Authorization: Bearer <jwt_token>" \
     -H "Content-Type: application/json" \
     https://your-patchmon-server.com/api/v1/auto-enrollment/tokens
```

Enrollment Endpoints (Token Key + Secret)

Enrollment endpoints authenticate via custom headers:

```
curl -H "X-Auto-Enrollment-Key: patchmon_ae_abc123..." \
     -H "X-Auto-Enrollment-Secret: def456ghi789..." \
     -H "Content-Type: application/json" \
     https://your-patchmon-server.com/api/v1/auto-enrollment/enroll
```

Host Endpoints (API ID + Key)

Host endpoints authenticate via API credential headers:

```
curl -H "X-API-ID: patchmon_abc123" \
     -H "X-API-KEY: def456ghi789" \
     https://your-patchmon-server.com/api/v1/hosts/install
```

Admin Endpoints

All admin endpoints require JWT authentication and "Manage Settings" permission.

Create Auto-Enrollment Token

Endpoint: `POST /api/v1/auto-enrollment/tokens`

Request Body:

Field	Type	Required	Default	Description
`token_name`	string	Yes	(required)	Descriptive name (max 255 chars)
`max_hosts_per_day`	integer	No	`100`	Rate limit (1–1000)
`default_host_group_id`	string	No	`null`	UUID of host group to auto-assign
`allowed_ip_ranges`	string[]	No	`[]`	IP whitelist (exact IPs or CIDR notation)
`expires_at`	string	No	`null`	ISO 8601 expiration date
`metadata`	object	No	`{}`	Custom metadata (e.g. `integration_type`, `environment`)
`scopes`	object	No	`null`	Permission scopes (only for API integration type tokens)

Example Request:

```
{
  "token_name": "Proxmox Production",
  "max_hosts_per_day": 100,
  "default_host_group_id": "uuid-of-host-group",
  "allowed_ip_ranges": ["192.168.1.10", "10.0.0.0/24"],
  "expires_at": "2026-12-31T23:59:59Z",
  "metadata": {
    "integration_type": "proxmox-lxc",
    "environment": "production"
  }
}
```

Response: `201 Created`

```
{
  "message": "Auto-enrollment token created successfully",
  "token": {
    "id": "uuid",
    "token_name": "Proxmox Production",
    "token_key": "patchmon_ae_abc123...",
    "token_secret": "def456ghi789...",
    "max_hosts_per_day": 100,
    "default_host_group": {
      "id": "uuid",
      "name": "Proxmox LXC",
      "color": "#3B82F6"
    },
    "created_by": {
      "id": "uuid",
      "username": "admin",
      "first_name": "John",
      "last_name": "Doe"
    },
    "expires_at": "2026-12-31T23:59:59Z",
    "scopes": null
  },
  "warning": "Save the token_secret now - it cannot be retrieved later!"
}
```

> ***Important:*** *The `token_secret` is only returned in this response. It is hashed before storage and cannot be retrieved again.*

List Auto-Enrollment Tokens

Endpoint: `GET /api/v1/auto-enrollment/tokens`

Response: `200 OK`

```
[
  {
    "id": "uuid",
    "token_name": "Proxmox Production",
    "token_key": "patchmon_ae_abc123...",
    "is_active": true,
    "allowed_ip_ranges": ["192.168.1.10"],
    "max_hosts_per_day": 100,
    "hosts_created_today": 15,
    "last_used_at": "2025-10-11T14:30:00Z",
    "expires_at": "2026-12-31T23:59:59Z",
    "created_at": "2025-10-01T10:00:00Z",
    "default_host_group_id": "uuid",
    "metadata": { "integration_type": "proxmox-lxc" },
    "scopes": null,
    "host_groups": {
      "id": "uuid",
      "name": "Proxmox LXC",
      "color": "#3B82F6"
    },
    "users": {
      "id": "uuid",
      "username": "admin",
      "first_name": "John",
      "last_name": "Doe"
    }
  }
]
```


Tokens are returned in descending order by creation date. The `token_secret` is never included in list responses.

Get Token Details

Endpoint: `GET /api/v1/auto-enrollment/tokens/{tokenId}`

Response: `200 OK`. Same structure as a single token in the list response (without `token_secret`).

Error: `404 Not Found` if `tokenId` does not exist.

Update Token

Endpoint: `PATCH /api/v1/auto-enrollment/tokens/{tokenId}`

All fields are optional. Only include fields you want to change.

Request Body:

Field	Type	Description
`token_name`	string	Updated name (1–255 chars)
`is_active`	boolean	Enable or disable the token
`max_hosts_per_day`	integer	Updated rate limit (1–1000)
`allowed_ip_ranges`	string[]	Updated IP whitelist
`default_host_group_id`	string	Updated host group (set to empty string to clear)
`expires_at`	string	Updated expiration date (ISO 8601)
`scopes`	object	Updated scopes (API integration type tokens only)

Example Request:

```
{
  "is_active": false,
  "max_hosts_per_day": 200,
  "allowed_ip_ranges": ["192.168.1.0/24"]
}
```

Response: `200 OK`

```
{
  "message": "Token updated successfully",
  "token": {
    "id": "uuid",
    "token_name": "Proxmox Production",
    "token_key": "patchmon_ae_abc123...",
    "is_active": false,
    "max_hosts_per_day": 200,
    "allowed_ip_ranges": ["192.168.1.0/24"],
    "host_groups": { "id": "uuid", "name": "Proxmox LXC", "color": "#3B82F6" },
    "users": { "id": "uuid", "username": "admin", "first_name": "John",
"last_name": "Doe" }
  }
}
```

Errors:

- `404 Not Found`: Token does not exist
- `400 Bad Request`: Host group not found, or scopes update attempted on a non-API token

Delete Token

Endpoint: `DELETE /api/v1/auto-enrollment/tokens/{tokenId}`

Response: `200 OK`

```
{
  "message": "Auto-enrollment token deleted successfully",
  "deleted_token": {
    "id": "uuid",
    "token_name": "Proxmox Production"
  }
}
```

Error: `404 Not Found` if `tokenId` does not exist.

Enrollment Endpoints

Download Enrollment Script

Endpoint: `GET /api/v1/auto-enrollment/script`

This endpoint validates the token credentials, then serves a bash script with the PatchMon server URL, token credentials, and configuration injected automatically.

Query Parameters:

Parameter	Required	Description
`type`	Yes	Script type: `proxmox-lxc` or `direct-host`
`token_key`	Yes	Auto-enrollment token key
`token_secret`	Yes	Auto-enrollment token secret
`force`	No	Set to `true` to enable force install mode (for broken packages)

Example:

```
curl "https://patchmon.example.com/api/v1/auto-enrollment/script?type=proxmox-lxc&token_key=KEY&token_secret=SECRET"
```

Response: `200 OK`. Plain text bash script with credentials injected.

Errors:

`400 Bad Request`: Missing or invalid `type` parameter

`401 Unauthorized`: Missing credentials, invalid/inactive token, invalid secret, or expired token

`404 Not Found`: Script file not found on server

Enroll Single Host

Endpoint: `POST /api/v1/auto-enrollment/enroll`

Headers:

```
X-Auto-Enrollment-Key: patchmon_ae_abc123...
X-Auto-Enrollment-Secret: def456ghi789...
Content-Type: application/json
```

Request Body:

Field	Type	Required	Description
`friendly_name`	string	Yes	Display name for the host (max 255 chars)
`machine_id`	string	No	Unique machine identifier (max 255 chars)
`metadata`	object	No	Additional metadata (vmid, proxmox_node, ip_address, os_info, etc.)

Example Request:

```
{
  "friendly_name": "webserver",
  "machine_id": "proxmox-lxc-100-abc123",
  "metadata": {
    "vmid": "100",
    "proxmox_node": "proxmox01",
    "ip_address": "10.0.0.10",
    "os_info": "Ubuntu 22.04 LTS"
  }
}
```

Response: `201 Created`

```
{
  "message": "Host enrolled successfully",
  "host": {
    "id": "uuid",
    "friendly_name": "webserver",
    "api_id": "patchmon_abc123def456",
    "api_key": "raw-api-key-value",
    "host_group": {
      "id": "uuid",
      "name": "Proxmox LXC",
      "color": "#3B82F6"
    },
    "status": "pending"
  }
}
```

> ***Note:*** *The `api_key` is only returned in this response (plain text). It is hashed before storage. The `host_group` is `null` if no default host group is configured on the token.*

Error Responses:

Status	Error	Cause
`400`	Validation errors	Missing or invalid `friendly_name`
`401`	`Auto-enrollment credentials required`	Missing `X-Auto-Enrollment-Key` or `X-Auto-Enrollment-Secret` headers
`401`	`Invalid or inactive token`	Token key not found or token is disabled
`401`	`Invalid token secret`	Secret does not match
`401`	`Token expired`	Token has passed its expiration date
`403`	`IP address not authorized for this token`	Client IP not in `allowed_ip_ranges`
`429`	`Rate limit exceeded`	Token's `max_hosts_per_day` limit reached

> ***Duplicate handling:*** *The API does not perform server-side duplicate host checks. Duplicate prevention is handled client-side by the enrollment script, which checks for an existing agent configuration (`/etc/patchmon/config.yml`) inside each container before calling the API.*

Host Management Endpoints

These endpoints are used by the PatchMon agent (not the enrollment script). They authenticate using the per-host `X-API-ID` and `X-API-KEY` credentials returned during enrollment.

Download Agent Installation Script

Endpoint: `GET /api/v1/hosts/install`

Serves a shell script that bootstraps the PatchMon agent on a host. The script uses a secure bootstrap token mechanism; actual API credentials are not embedded directly in the script.

Headers:

```
X-API-ID: patchmon_abc123
X-API-KEY: def456ghi789
```

Query Parameters:

Parameter	Required	Description
`force`	No	Set to `true` to enable force install mode
`arch`	No	Architecture override (e.g. `amd64`, `arm64`); auto-detected if omitted

Response: `200 OK`. Plain text shell script with bootstrap token injected.

Download Agent Binary/Script

Endpoint: `GET /api/v1/hosts/agent/download`

Downloads the PatchMon agent binary (Go binary for modern agents) or migration script (for legacy bash agents).

Headers:

```
X-API-ID: patchmon_abc123
X-API-KEY: def456ghi789
```

Query Parameters:

Parameter	Required	Description
`arch`	No	Architecture (e.g. `amd64`, `arm64`)
`force`	No	Set to `binary` to force binary download

Response: `200 OK`. Binary file or shell script.

Host Data Update

Endpoint: `POST /api/v1/hosts/update`

Used by the agent to report package data, system information, and hardware details.

Headers:

```
X-API-ID: patchmon_abc123
X-API-KEY: def456ghi789
Content-Type: application/json
```

Request Body Fields:

Field	Type	Required	Description
`packages`	array	Yes	Array of package objects (max 10,000)
`packages[].name`	string	Yes	Package name
`packages[].currentVersion`	string	Yes	Currently installed version
`packages[].availableVersion`	string	No	Available update version
`packages[].needsUpdate`	boolean	Yes	Whether an update is available
`packages[].isSecurityUpdate`	boolean	No	Whether the update is security-related
`agentVersion`	string	No	Reporting agent version
`osType`	string	No	Operating system type
`osVersion`	string	No	Operating system version
`hostname`	string	No	System hostname
`ip`	string	No	System IP address
`architecture`	string	No	CPU architecture
`cpuModel`	string	No	CPU model name
`cpuCores`	integer	No	Number of CPU cores
`ramInstalled`	float	No	Installed RAM in GB
`swapSize`	float	No	Swap size in GB
`diskDetails`	array	No	Array of disk objects
`gatewayIp`	string	No	Default gateway IP
`dnsServers`	array	No	Array of DNS server IPs
`networkInterfaces`	array	No	Array of network interface objects
`kernelVersion`	string	No	Running kernel version
`installedKernelVersion`	string	No	Installed (on-disk) kernel version
`selinuxStatus`	string	No	SELinux status (`enabled` , `disabled` , or `permissive`)
`systemUptime`	string	No	System uptime. Inside a container (Proxmox LXC, Docker) this is the container's own uptime, not the host's
`bootTime`	string	No	Boot instant as a UTC ISO 8601 timestamp. Lets the UI tick uptime live

Field	Type	Required	Description
			between reports
`loadAverage`	array	No	Load average values
`machineId`	string	No	Machine ID
`needsReboot`	boolean	No	Whether a reboot is required
`rebootReason`	string	No	Reason a reboot is required
`repositories`	array	No	Configured package repositories
`executionTime`	string	No	Time taken to gather data

Example Request:

```
{
  "packages": [
    {
      "name": "nginx",
      "currentVersion": "1.18.0",
      "availableVersion": "1.20.0",
      "needsUpdate": true,
      "isSecurityUpdate": false
    }
  ],
  "agentVersion": "1.5.0",
  "cpuModel": "Intel Xeon E5-2680 v4",
  "cpuCores": 8,
  "ramInstalled": 16.0,
  "swapSize": 2.0,
  "diskDetails": [
    {
      "device": "/dev/sda1",
      "mountPoint": "/",
      "size": "50GB",
      "used": "25GB",
      "available": "25GB"
    }
  ],
  "gatewayIp": "192.168.1.1",
  "dnsServers": ["8.8.8.8", "8.8.4.4"],
  "networkInterfaces": [
    {
      "name": "eth0",
      "ip": "192.168.1.10",
      "mac": "00:11:22:33:44:55"
    }
  ],
  "kernelVersion": "5.4.0-74-generic",
  "selinuxStatus": "disabled"
}
```


Response: `200 OK`


```
{
  "message": "Host updated successfully",
  "packagesProcessed": 1,
  "updatesAvailable": 1,
  "securityUpdates": 0
}
```

Ansible Integration Examples

Basic Playbook for Proxmox Enrollment

```
---
- name: Enroll Proxmox LXC containers in PatchMon
  hosts: proxmox_hosts
  become: yes
  vars:
    patchmon_url: "https://patchmon.example.com"
    token_key: "{{ vault_patchmon_token_key }}"
    token_secret: "{{ vault_patchmon_token_secret }}"
    host_prefix: "prod-"

  tasks:
    - name: Install dependencies
      apt:
        name:
          - curl
          - jq
        state: present

    - name: Download enrollment script
      get_url:
        url: "{{ patchmon_url }}/api/v1/auto-enrollment/script?type=proxmox-
lxc&token_key={{ token_key }}&token_secret={{ token_secret }}"
        dest: /root/proxmox_auto_enroll.sh
        mode: '0700'

    - name: Run enrollment
      command: /root/proxmox_auto_enroll.sh
      environment:
        HOST_PREFIX: "{{ host_prefix }}"
        DEBUG: "true"
      register: enrollment_output

    - name: Show enrollment results
      debug:
        var: enrollment_output.stdout_lines
```

Advanced Playbook with Token Management

```
---
- name: Manage PatchMon Proxmox Integration
  hosts: localhost
  vars:
    patchmon_url: "https://patchmon.example.com"
    admin_token: "{{ vault_patchmon_admin_token }}"

  tasks:
    - name: Create Proxmox enrollment token
      uri:
        url: "{{ patchmon_url }}/api/v1/auto-enrollment/tokens"
        method: POST
        headers:
          Authorization: "Bearer {{ admin_token }}"
          Content-Type: "application/json"
        body_format: json
        body:
          token_name: "{{ inventory_hostname }}-proxmox"
          max_hosts_per_day: 200
          default_host_group_id: "{{ proxmox_host_group_id }}"
          allowed_ip_ranges: ["{{ proxmox_host_ip }}"]
          expires_at: "2026-12-31T23:59:59Z"
          metadata:
            integration_type: "proxmox-lxc"
            environment: "{{ environment }}"
        status_code: 201
      register: token_response

    - name: Store token credentials
      set_fact:
        enrollment_token_key: "{{ token_response.json.token.token_key }}"
        enrollment_token_secret: "{{ token_response.json.token.token_secret }}"

    - name: Deploy enrollment script to Proxmox hosts
      include_tasks: deploy_enrollment.yml
      vars:
        enrollment_token_key: "{{ enrollment_token_key }}"
        enrollment_token_secret: "{{ enrollment_token_secret }}"
```

Ansible Role

```
# roles/patchmon_proxmox/tasks/main.yml
---
- name: Install PatchMon dependencies
  package:
    name:
      - curl
      - jq
    state: present

- name: Create PatchMon directory
  file:
    path: /opt/patchmon
    state: directory
    mode: '0755'

- name: Download enrollment script
  get_url:
    url: "{{ patchmon_url }}/api/v1/auto-enrollment/script?type=proxmox-
lxc&token_key={{ token_key }}&token_secret={{ token_secret }}&force={{
force_install | default('false') }}"
    dest: /opt/patchmon/proxmox_auto_enroll.sh
    mode: '0700'

- name: Run enrollment script
  command: /opt/patchmon/proxmox_auto_enroll.sh
  environment:
    PATCHMON_URL: "{{ patchmon_url }}"
    AUTO_ENROLLMENT_KEY: "{{ token_key }}"
    AUTO_ENROLLMENT_SECRET: "{{ token_secret }}"
    HOST_PREFIX: "{{ host_prefix | default('') }}"
    DRY_RUN: "{{ dry_run | default('false') }}"
    DEBUG: "{{ debug | default('false') }}"
    FORCE_INSTALL: "{{ force_install | default('false') }}"
  register: enrollment_output

- name: Display enrollment results
  debug:
    var: enrollment_output.stdout_lines
  when: enrollment_output.stdout_lines is defined

- name: Fail if enrollment had errors
  fail:
    msg: "Enrollment failed with errors"
  when: enrollment_output.rc != 0
```

Ansible Vault for Credentials

```
# group_vars/all/vault.yml (encrypted with ansible-vault)
---
vault_patchmon_admin_token: "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."
vault_patchmon_token_key: "patchmon_ae_abc123..."
vault_patchmon_token_secret: "def456ghi789..."
```

Playbook with Error Handling and Retries

```
---
- name: Robust Proxmox enrollment with error handling
  hosts: proxmox_hosts
  become: yes
  vars:
    patchmon_url: "https://patchmon.example.com"
    token_key: "{{ vault_patchmon_token_key }}"
    token_secret: "{{ vault_patchmon_token_secret }}"
    max_retries: 3
    retry_delay: 30

  tasks:
    - name: Test PatchMon connectivity
      uri:
        url: "{{ patchmon_url }}/api/v1/auto-enrollment/tokens"
        method: GET
        headers:
          Authorization: "Bearer {{ vault_patchmon_admin_token }}"
        status_code: 200
      retries: "{{ max_retries }}"
      delay: "{{ retry_delay }}"

    - name: Download enrollment script
      get_url:
        url: "{{ patchmon_url }}/api/v1/auto-enrollment/script?type=proxmox-
lxc&token_key={{ token_key }}&token_secret={{ token_secret }}"
        dest: /root/proxmox_auto_enroll.sh
        mode: '0700'
      retries: "{{ max_retries }}"
      delay: "{{ retry_delay }}"

    - name: Run enrollment with retry logic
      shell: |
        for i in {1..{{ max_retries }}}; do
          echo "Attempt $i of {{ max_retries }}"
          if /root/proxmox_auto_enroll.sh; then
            echo "Enrollment successful"
            exit 0
          else
            echo "Enrollment failed, retrying in {{ retry_delay }} seconds..."
            sleep {{ retry_delay }}
          fi
        done
        echo "All enrollment attempts failed"
        exit 1
      register: enrollment_result

    - name: Handle enrollment failure
      fail:
        msg: "Proxmox enrollment failed after {{ max_retries }} attempts"
      when: enrollment_result.rc != 0
```

```
    - name: Parse enrollment results
      set_fact:
        enrolled_count: "{{ enrollment_result.stdout | regex_search('Successfully
Enrolled:\\s+(\\d+)', '\\1') | default('0') }}"
        failed_count: "{{ enrollment_result.stdout | regex_search('Failed:\\s+
(\\d+)', '\\1') | default('0') }}"

    - name: Report enrollment statistics
      debug:
        msg: |
          Enrollment completed:
          - Successfully enrolled: {{ enrolled_count }} containers
          - Failed: {{ failed_count }} containers
```

Error Handling

HTTP Status Codes

Code	Meaning	When It Occurs
`200`	OK	Successful read/update operations
`201`	Created	Token or host created successfully
`400`	Bad Request	Validation errors, invalid host group, invalid script type
`401`	Unauthorized	Missing, invalid, or expired credentials
`403`	Forbidden	IP address not in token's whitelist
`404`	Not Found	Token or resource not found
`429`	Too Many Requests	Token's daily host creation limit exceeded
`500`	Internal Server Error	Unexpected server error

Error Response Formats

Simple error:

```
{
  "error": "Error message describing what went wrong"
}
```

Error with detail:

```
{
  "error": "Rate limit exceeded",
  "message": "Maximum 100 hosts per day allowed for this token"
}
```

Validation errors (400):

```
{
  "errors": [
    {
      "msg": "Token name is required (max 255 characters)",
      "param": "token_name",
      "location": "body"
    }
  ]
}
```

Rate Limiting

Token-Based Rate Limits

Each auto-enrollment token has a configurable `max_hosts_per_day` limit:

- **Default**: 100 hosts per day per token
- **Range**: 1–1000 hosts per day
- **Reset**: Daily (when the first request of a new day is received)
- **Scope**: Per-token, not per-IP

When the limit is exceeded, the API returns `429 Too Many Requests`:

```
{
  "error": "Rate limit exceeded",
  "message": "Maximum 100 hosts per day allowed for this token"
}
```

Global Rate Limiting

The auto-enrollment endpoints are also subject to the server's global authentication rate limiter, which applies to all authentication-related endpoints.

Security Considerations

Token Security

- **Secret hashing**: Token secrets are hashed with bcrypt (cost factor 10) before storage

One-time display: Secrets are only returned during token creation

Rotation: Recommended every 90 days

Scope limitation: Tokens can only create hosts. They cannot read, modify, or delete existing host data.

IP Restrictions

Tokens support IP whitelisting with both exact IPs and CIDR notation:

```
{
  "allowed_ip_ranges": ["192.168.1.10", "10.0.0.0/24"]
}
```

IPv4-mapped IPv6 addresses (e.g. `::ffff:192.168.1.10`) are automatically handled.

Host API Key Security

Host API keys (`api_key`) are hashed with bcrypt before storage

The installation script uses a bootstrap token mechanism; the actual API credentials are not embedded in the script

Bootstrap tokens are single-use and expire after 5 minutes

Network Security

Always use HTTPS in production

The `ignore_ssl_self_signed` server setting automatically configures curl flags in served scripts

Implement firewall rules to restrict PatchMon server access to known IPs

Audit Trail

All enrollment activity is logged:

Token name included in host notes (e.g. "Auto-enrolled via Production Proxmox on 2025-10-11T14:30:00Z")

Token creation tracks `created_by_user_id`

`last_used_at` timestamp updated on each enrollment

Complete Endpoint Summary

Admin Endpoints (JWT Authentication)

Method	Path	Description
`POST`	`/api/v1/auto-enrollment/tokens`	Create token
`GET`	`/api/v1/auto-enrollment/tokens`	List all tokens
`GET`	`/api/v1/auto-enrollment/tokens/{tokenId}`	Get single token
`PATCH`	`/api/v1/auto-enrollment/tokens/{tokenId}`	Update token
`DELETE`	`/api/v1/auto-enrollment/tokens/{tokenId}`	Delete token

Enrollment Endpoints (Token Authentication)

Method	Path	Description
`GET`	`/api/v1/auto-enrollment/script?type=...`	Download enrollment script
`POST`	`/api/v1/auto-enrollment/enroll`	Enroll a host

Host Endpoints (API Credentials)

Method	Path	Description
`GET`	`/api/v1/hosts/install`	Download installation script
`GET`	`/api/v1/hosts/agent/download`	Download agent binary/script
`POST`	`/api/v1/hosts/update`	Report host data

Quick Reference: curl Examples

Create a token:

```
curl -X POST \
  -H "Authorization: Bearer <jwt_token>" \
  -H "Content-Type: application/json" \
  -d '{
    "token_name": "Production Proxmox",
    "max_hosts_per_day": 100,
    "default_host_group_id": "uuid",
    "allowed_ip_ranges": ["192.168.1.10"]
  }' \
  https://patchmon.example.com/api/v1/auto-enrollment/tokens
```

Download and run enrollment script:

```
curl -s "https://patchmon.example.com/api/v1/auto-enrollment/script?type=proxmox-lxc&token_key=KEY&token_secret=SECRET" | bash
```

Enroll a host directly:

```
curl -X POST \
  -H "X-Auto-Enrollment-Key: patchmon_ae_abc123..." \
  -H "X-Auto-Enrollment-Secret: def456ghi789..." \
  -H "Content-Type: application/json" \
  -d '{
    "friendly_name": "webserver",
    "machine_id": "proxmox-lxc-100-abc123"
  }' \
  https://patchmon.example.com/api/v1/auto-enrollment/enroll
```

Download agent installation script:

```
curl -H "X-API-ID: patchmon_abc123" \
     -H "X-API-KEY: def456ghi789" \
     https://patchmon.example.com/api/v1/hosts/install | bash
```

Integration Patterns

Pattern 1: Script-Based (Simplest)

```
# Download and execute in one command (credentials are injected into the script)
curl -s "https://patchmon.example.com/api/v1/auto-enrollment/script?type=proxmox-lxc&token_key=KEY&token_secret=SECRET" | bash
```

Pattern 2: API-First (Most Control)

```
# 1. Create token via admin API
# 2. Enroll hosts via enrollment API
# 3. Download agent scripts using per-host API credentials
# 4. Install agents with host-specific credentials
```

Pattern 3: Hybrid (Recommended for Automation)

```
# 1. Create token via admin API (or UI)
# 2. Download enrollment script with token embedded
# 3. Distribute and run script on Proxmox hosts
# 4. Script handles both enrollment and agent installation
```

Chapter 6: Integration API Documentation

Table of Contents

- Overview
- Interactive API Reference (Swagger)
- Creating API Credentials
- Authentication
- Available Scopes & Permissions
- API Endpoints
  - List Hosts
  - Get Host Statistics
  - Get Host Information
  - Get Host Network Information
  - Get Host System Information
  - Get Host Packages
  - Get Host Package Reports
  - Get Host Agent Queue
  - Get Host Notes
  - Get Host Integrations
  - Delete Host
- Usage Examples
- Security Best Practices
- Troubleshooting

Overview

PatchMon's Integration API provides programmatic access to your PatchMon instance, enabling automation, integration with third-party tools, and custom workflows. API credentials use **HTTP Basic Authentication** with scoped permissions to control access to specific resources and actions.

Key Features

- **Scoped Permissions**: Fine-grained control over what each credential can access
- **IP Restrictions**: Optional IP allowlisting for enhanced security
- **Expiration Dates**: Set automatic expiration for temporary access
- **Basic Authentication**: Industry-standard authentication method (RFC 7617)
- **Rate Limiting**: Built-in protection against abuse

Audit Trail: Track credential usage with last-used timestamps

Use Cases

Automation: Integrate PatchMon data into CI/CD pipelines
Inventory Management: Use with Ansible, Terraform, or other IaC tools
Monitoring: Feed PatchMon data into monitoring dashboards
Custom Scripts: Build custom tools that interact with PatchMon
Third-Party Integrations: Connect PatchMon to other systems

Interactive API Reference (Swagger)

PatchMon includes a built-in interactive API reference powered by Swagger UI. You can explore all available endpoints, view request/response schemas, and test API calls directly from your browser.

To access the Swagger UI:

```
https://<your-patchmon-url>/api/v1/api-docs
```

> ***Note:*** *The Swagger UI requires you to be logged in to PatchMon (JWT authentication). Log in to your PatchMon dashboard first, then navigate to the URL above in the same browser session.*

The Swagger reference covers all internal and scoped API endpoints. This documentation page focuses specifically on the **scoped Integration API** that uses Basic Authentication with API credentials.

Creating API Credentials

Step-by-Step Guide

1. Navigate to Settings
   Log in to your PatchMon instance as an administrator
   Go to **Settings** → **Integrations**
   You will see the **Auto-Enrollment & API** tab

2. Click "New Token"
Click the **"New Token"** button. A modal will appear where you can select the credential type.

3. Select "API" as the Usage Type

In the creation modal, select **"API"** as the usage type. This configures the credential for programmatic access via Basic Authentication.

4. Configure the Credential
Fill in the following fields:

Required Fields:

Field	Description	Example
Token Name	A descriptive name for identification and audit purposes	`Ansible Inventory`, `Monitoring Dashboard`
Scopes	The permissions this credential should have (at least one required)	`host: get`

Optional Fields:

Field	Description	Example
Allowed IP Addresses	Comma-separated list of IPs or CIDR ranges that can use this credential. Leave empty for unrestricted access.	`192.168.1.100, 10.0.0.0/24`
Expiration Date	Automatic expiration date for the credential. Leave empty for no expiration.	`2026-12-31T23:59:59`
Default Host Group	Optionally assign a default host group	`Production`

5. Save Your Credentials
CRITICAL: Save these credentials immediately. The secret cannot be retrieved later.

After creation, a success modal displays:

- **Token Key**: The API key (used as the username in Basic Auth), prefixed with `patchmon_ae_`
- **Token Secret**: The API secret (used as the password). **Shown only once.**
- **Granted Scopes**: The permissions assigned
- **Usage Examples**: Pre-filled cURL commands ready to copy

Copy both the Token Key and Token Secret and store them securely before closing the modal.

Authentication

Basic Authentication

PatchMon API credentials use HTTP Basic Authentication as defined in RFC 7617 (https://tools.ietf.org/html/rfc7617).

Format

```
Authorization: Basic <base64(token_key:token_secret)>
```

How It Works

1. Combine your token key and secret with a colon: `token_key:token_secret`
2. Encode the combined string in Base64
3. Prepend `Basic` to the encoded string
4. Send it in the `Authorization` header

Most HTTP clients handle this automatically (for example, cURL's `-u` flag or Python's `HTTPBasicAuth`).

Authentication Flow

Validation Steps (In Order)

The server performs these checks sequentially. If any step fails, the request is rejected immediately:

1. **Authorization Header**: checks for `Authorization: Basic` header
2. **Credential Format**: validates `key:secret` format after Base64 decoding
3. **Token Existence**: looks up the token key in the database
4. **Active Status**: verifies `is_active` flag is `true`
5. **Expiration**: checks token has not expired (`expires_at`)
6. **Integration Type**: confirms `metadata.integration_type` is `"api"`
7. **Secret Verification**: compares provided secret against the bcrypt hash
8. **IP Restriction**: validates client IP against `allowed_ip_ranges` (if configured)
9. **Last Used Update**: updates the `last_used_at` timestamp (occurs during authentication, before the handler runs)
10. **Scope Validation**: verifies the credential has the required scope for the endpoint (handled by separate middleware)

Available Scopes & Permissions

API credentials use a **resource–action** scope model:

```
{
  "resource": ["action1", "action2"]
}
```

Host Resource

Resource name: `host`

Action	Description
`get`	Read host data (list hosts, view details, stats, packages, network, system, reports, notes, integrations)
`delete`	Delete hosts

Example scope configurations:

```
// Read-only access
{ "host": ["get"] }

// Read and delete
{ "host": ["get", "delete"] }
```

Important Notes

Scopes are **explicit**: no inheritance or wildcards. Each action must be explicitly granted.

`get` does **not** automatically include `delete` or any other action.

At least one action must be granted for at least one resource. Credentials with no scopes will be rejected during creation.

API Endpoints

All endpoints are prefixed with `/api/v1/api` and require Basic Authentication with a credential that has the appropriate scope.

Endpoints Summary

Endpoint	Method	Scope	Description
`/api/v1/api/hosts`	GET	`host:get`	List all hosts with IP, groups, and optional stats
`/api/v1/api/hosts/:id/stats`	GET	`host:get`	Get host package/repo statistics
`/api/v1/api/hosts/:id/info`	GET	`host:get`	Get detailed host information
`/api/v1/api/hosts/:id/network`	GET	`host:get`	Get host network configuration
`/api/v1/api/hosts/:id/system`	GET	`host:get`	Get host system details
`/api/v1/api/hosts/:id/packages`	GET	`host:get`	Get host packages (with optional update filter)
`/api/v1/api/hosts/:id/package_reports`	GET	`host:get`	Get package update history
`/api/v1/api/hosts/:id/agent_queue`	GET	`host:get`	Get agent queue status and jobs
`/api/v1/api/hosts/:id/notes`	GET	`host:get`	Get host notes
`/api/v1/api/hosts/:id/integrations`	GET	`host:get`	Get host integration status
`/api/v1/api/hosts/:id`	DELETE	`host:delete`	Delete a host and all related data

List Hosts

Retrieve a list of all hosts with their IP addresses and host group memberships. Optionally include package update statistics inline with each host.

Endpoint:

```
GET /api/v1/api/hosts
```

Required Scope: `host:get`

Query Parameters:

Parameter	Type	Required	Description
`hostgroup`	string	No	Filter by host group name(s) or UUID(s). Comma-separated for multiple groups (OR logic).
`include`	string	No	Comma-separated list of additional data to include. Supported values: `stats`.

Filtering by Host Groups:

```
# Filter by group name
GET /api/v1/api/hosts?hostgroup=Production

# Filter by multiple groups (hosts in ANY of the listed groups)
GET /api/v1/api/hosts?hostgroup=Production,Development

# Filter by group UUID
GET /api/v1/api/hosts?hostgroup=550e8400-e29b-41d4-a716-446655440000

# Mix names and UUIDs
GET /api/v1/api/hosts?hostgroup=Production,550e8400-e29b-41d4-a716-446655440000
```

Including Stats:

Use `?include=stats` to add package update counts and additional host metadata to each host in a single request. This is more efficient than making separate `/stats` calls for every host.

```
# List all hosts with stats
GET /api/v1/api/hosts?include=stats

# Combine with host group filter
GET /api/v1/api/hosts?hostgroup=Production&include=stats
```

> ***Note:*** *If your host group names contain spaces, URL-encode them with* `%20` *(e.g.* `Web%20Servers`*). Most HTTP clients handle this automatically.*

Response (200 OK) without stats:

```
{
  "hosts": [
    {
      "id": "550e8400-e29b-41d4-a716-446655440000",
      "friendly_name": "web-server-01",
      "hostname": "web01.example.com",
      "ip": "192.168.1.100",
      "host_groups": [
        {
          "id": "660e8400-e29b-41d4-a716-446655440001",
          "name": "Production"
        }
      ]
    }
  ],
  "total": 1,
  "filtered_by_groups": ["Production"]
}
```


Response (200 OK) with stats (`?include=stats`):

```
{
  "hosts": [
    {
      "id": "550e8400-e29b-41d4-a716-446655440000",
      "friendly_name": "web-server-01",
      "hostname": "web01.example.com",
      "ip": "192.168.1.100",
      "host_groups": [
        {
          "id": "660e8400-e29b-41d4-a716-446655440001",
          "name": "Production"
        }
      ],
      "os_type": "Ubuntu",
      "os_version": "24.04 LTS",
      "last_update": "2026-02-12T10:30:00.000Z",
      "status": "active",
      "effective_status": "active",
      "reporting_state": "reporting",
      "update_state": "security_required",
      "needs_reboot": false,
      "updates_count": 15,
      "security_updates_count": 3,
      "total_packages": 342
    }
  ],
  "total": 1,
  "filtered_by_groups": ["Production"]
}
```


> *The `filtered_by_groups` field is only present when a `hostgroup` filter is applied.*

Response Fields:

Field	Type	Description
`hosts`	array	Array of host objects
`hosts[].id`	string (UUID)	Unique host identifier
`hosts[].friendly_name`	string	Human-readable host name
`hosts[].hostname`	string	System hostname
`hosts[].ip`	string	Primary IP address
`hosts[].host_groups`	array	Groups this host belongs to
`hosts[].os_type`	string	Operating system type (only with `include=stats`)
`hosts[].os_version`	string	Operating system version (only with `include=stats`)
`hosts[].last_update`	string (ISO 8601)	Timestamp of last agent update (only with `include=stats`)
`hosts[].status`	string	Enrolment lifecycle state, `pending` or `active` (only with `include=stats`). See "Which status field should I use?" below
`hosts[].effective_status`	string	The status the web interface displays: `pending`, `active` or `inactive` (only with `include=stats`)
`hosts[].reporting_state`	string	Report freshness: `reporting`, `overdue` or `stale` (only with `include=stats`)
`hosts[].update_state`	string	Patch position: `up_to_date`, `updates_pending` or `security_required` (only with `include=stats`)
`hosts[].needs_reboot`	boolean	Whether a reboot is pending (only with `include=stats`)
`hosts[].updates_count`	integer	Number of packages needing updates (only with `include=stats`)
`hosts[].security_updates_count`	integer	Number of security updates available (only with `include=stats`)
`hosts[].total_packages`	integer	Total installed packages (only with `include=stats`)
`total`	integer	Total number of hosts returned

Field	Type	Description
`filtered_by_groups`	array	Groups used for filtering (only present when filtering)

Which status field should I use?
The response carries four status-like fields because they answer four different questions. Picking the wrong one is the most common source of confusion when an integration disagrees with what the web interface shows.

Field	Question it answers	Values
`status`	Has this host finished enrolling?	`pending` until the first check-in, then `active` for ever
`effective_status`	What does the web interface show for this host?	`pending`, `active`, `inactive`
`reporting_state`	How fresh is this host's data?	`reporting`, `overdue`, `stale`
`update_state`	Does this host need patching?	`up_to_date`, `updates_pending`, `security_required`

Two points worth knowing:

> **`status` never becomes `inactive`.** It records how far a host got through enrolment, not whether it is alive. Once a host has checked in once, it stays `active` until you delete it, even if it never reports again. If you are looking for "is this host still talking to PatchMon", use `effective_status` or `reporting_state`.
>
> **`effective_status` is calculated when you make the request**, from `last_update` and the Update Interval configured in Settings. A host reads as `inactive` once it has been silent for more than twice that interval, which is exactly the rule the web interface applies. `reporting_state` uses the same clock but splits the middle out: `overdue` covers one to two intervals of silence, and `stale` is past two.

The `pending` case is the one place the two disagree on purpose. A host that was created but never checked in stays `pending` in `effective_status`, because "never finished enrolling" is more useful than "went quiet", whilst `reporting_state` reports it as `stale`.

> *`effective_status` was added in PatchMon 2.0.3. On earlier versions the endpoint returns only `status`, and you can reproduce the web interface value yourself by comparing `last_update` against twice your configured Update Interval.*

Get Host Statistics

Retrieve package and repository statistics for a specific host.

Endpoint:

```
GET /api/v1/api/hosts/:id/stats
```

Required Scope: `host:get`

Response (200 OK):

```
{
  "host_id": "550e8400-e29b-41d4-a716-446655440000",
  "total_installed_packages": 342,
  "outdated_packages": 15,
  "security_updates": 3,
  "total_repos": 8
}
```

Response Fields:

Field	Type	Description
`host_id`	string (UUID)	The host identifier
`total_installed_packages`	integer	Total packages installed on this host
`outdated_packages`	integer	Packages that need updates
`security_updates`	integer	Packages with security updates available
`total_repos`	integer	Total repositories associated with the host

Get Host Information

Retrieve detailed information about a specific host including OS details and host groups.

Endpoint:

```
GET /api/v1/api/hosts/:id/info
```

Required Scope: `host:get`

Response (200 OK):

```
{
  "id": "550e8400-e29b-41d4-a716-446655440000",
  "machine_id": "abc123def456",
  "friendly_name": "web-server-01",
  "hostname": "web01.example.com",
  "ip": "192.168.1.100",
  "os_type": "Ubuntu",
  "os_version": "24.04 LTS",
  "agent_version": "1.5.0",
  "host_groups": [
    {
      "id": "660e8400-e29b-41d4-a716-446655440001",
      "name": "Production"
    }
  ]
}
```

Get Host Network Information

Retrieve network configuration details for a specific host.

Endpoint:

```
GET /api/v1/api/hosts/:id/network
```

Required Scope: `host:get`

Response (200 OK):

```
{
  "id": "550e8400-e29b-41d4-a716-446655440000",
  "ip": "192.168.1.100",
  "gateway_ip": "192.168.1.1",
  "dns_servers": ["8.8.8.8", "8.8.4.4"],
  "network_interfaces": [
    {
      "name": "eth0",
      "ip": "192.168.1.100",
      "mac": "00:11:22:33:44:55"
    }
  ]
}
```

Get Host System Information

Retrieve system-level information for a specific host including hardware, kernel, and reboot status.

Endpoint:

```
GET /api/v1/api/hosts/:id/system
```

Required Scope: `host:get`

Response (200 OK):


```
{
  "id": "550e8400-e29b-41d4-a716-446655440000",
  "architecture": "x86_64",
  "kernel_version": "6.8.0-45-generic",
  "installed_kernel_version": "6.8.0-50-generic",
  "selinux_status": "disabled",
  "system_uptime": "15 days, 3:22:10",
  "cpu_model": "Intel Xeon E5-2680 v4",
  "cpu_cores": 4,
  "ram_installed": 8192,
  "swap_size": 2048,
  "load_average": {
    "1min": 0.5,
    "5min": 0.3,
    "15min": 0.2
  },
  "disk_details": [
    {
      "filesystem": "/dev/sda1",
      "size": "50G",
      "used": "22G",
      "available": "28G",
      "use_percent": "44%",
      "mounted_on": "/"
    }
  ],
  "needs_reboot": true,
  "reboot_reason": "Kernel update pending"
}
```

Get Host Packages

Retrieve the list of packages installed on a specific host. Use the optional `updates_only` parameter to return only packages with available updates.

Endpoint:

```
GET /api/v1/api/hosts/:id/packages
```

Required Scope: `host:get`

Query Parameters:

Parameter	Type	Required	Default	Description
`updates_only`	string	No	(none)	Set to `true` to return only packages that need updates

Examples:

```
# Get all packages for a host
curl -u "patchmon_ae_abc123:your_secret_here" \
  https://patchmon.example.com/api/v1/api/hosts/HOST_UUID/packages

# Get only packages with available updates
curl -u "patchmon_ae_abc123:your_secret_here" \
  "https://patchmon.example.com/api/v1/api/hosts/HOST_UUID/packages?
updates_only=true"
```

Response (200 OK):

```
{
  "host": {
    "id": "550e8400-e29b-41d4-a716-446655440000",
    "hostname": "web01.example.com",
    "friendly_name": "web-server-01"
  },
  "packages": [
    {
      "id": "package-host-uuid",
      "name": "nginx",
      "description": "High performance web server",
      "category": "web",
      "current_version": "1.18.0-0ubuntu1.5",
      "available_version": "1.24.0-2ubuntu1",
      "needs_update": true,
      "is_security_update": false,
      "last_checked": "2026-02-12T10:30:00.000Z"
    },
    {
      "id": "package-host-uuid-2",
      "name": "openssl",
      "description": "Secure Sockets Layer toolkit",
      "category": "security",
      "current_version": "3.0.2-0ubuntu1.14",
      "available_version": "3.0.2-0ubuntu1.18",
      "needs_update": true,
      "is_security_update": true,
      "last_checked": "2026-02-12T10:30:00.000Z"
    }
  ],
  "total": 2
}
```


Response Fields:

Field	Type	Description
`host`	object	Basic host identification
`host.id`	string (UUID)	Host identifier
`host.hostname`	string	System hostname
`host.friendly_name`	string	Human-readable host name
`packages`	array	Array of package objects
`packages[].id`	string (UUID)	Host-package record identifier
`packages[].name`	string	Package name
`packages[].description`	string	Package description
`packages[].category`	string	Package category
`packages[].current_version`	string	Currently installed version
`packages[].available_version`	string | null	Available update version (null if up to date)
`packages[].needs_update`	boolean	Whether an update is available
`packages[].is_security_update`	boolean	Whether the available update is security-related
`packages[].last_checked`	string (ISO 8601)	When this package was last checked
`total`	integer	Total number of packages returned

> ***Tip:*** *Packages are returned sorted by security updates first, then by update availability. This puts the most critical packages at the top.*

Get Host Package Reports

Retrieve package update history reports for a specific host.

Endpoint:

```
GET /api/v1/api/hosts/:id/package_reports
```

Required Scope: `host:get`

Query Parameters:

Parameter	Type	Required	Default	Description
`limit`	integer	No	10	Maximum number of reports to return

Response (200 OK):

```
{
  "host_id": "550e8400-e29b-41d4-a716-446655440000",
  "reports": [
    {
      "id": "report-uuid",
      "status": "success",
      "date": "2026-02-12T10:30:00.000Z",
      "total_packages": 342,
      "outdated_packages": 15,
      "security_updates": 3,
      "payload_kb": 12.5,
      "execution_time_seconds": 4.2,
      "error_message": null
    }
  ],
  "total": 1
}
```

Get Host Agent Queue

Retrieve agent queue status and job history for a specific host.

Endpoint:

```
GET /api/v1/api/hosts/:id/agent_queue
```

Required Scope: `host:get`

Query Parameters:

Parameter	Type	Required	Default	Description
`limit`	integer	No	10	Maximum number of jobs to return

Response (200 OK):

```
{
  "host_id": "550e8400-e29b-41d4-a716-446655440000",
  "queue_status": {
    "waiting": 0,
    "active": 1,
    "delayed": 0,
    "failed": 0
  },
  "job_history": [
    {
      "id": "job-history-uuid",
      "job_id": "bull-job-id",
      "job_name": "package_update",
      "status": "completed",
      "attempt": 1,
      "created_at": "2026-02-12T10:00:00.000Z",
      "completed_at": "2026-02-12T10:05:00.000Z",
      "error_message": null,
      "output": null
    }
  ],
  "total_jobs": 1
}
```

Get Host Notes

Retrieve notes associated with a specific host.

Endpoint:

```
GET /api/v1/api/hosts/:id/notes
```

Required Scope: `host:get`

Response (200 OK):


```
{
  "host_id": "550e8400-e29b-41d4-a716-446655440000",
  "notes": "Production web server. Enrolled via Proxmox auto-enrollment on 2026-
01-15."
}
```

Get Host Integrations

Retrieve integration status and details for a specific host (e.g. Docker).

Endpoint:

```
GET /api/v1/api/hosts/:id/integrations
```

Required Scope: `host:get`

Response (200 OK, Docker enabled):

```
{
  "host_id": "550e8400-e29b-41d4-a716-446655440000",
  "integrations": {
    "docker": {
      "enabled": true,
      "containers_count": 12,
      "volumes_count": 5,
      "networks_count": 3,
      "description": "Monitor Docker containers, images, volumes, and networks.
Collects real-time container status events."
    }
  }
}
```

Response (200 OK, Docker not enabled):

```
{
  "host_id": "550e8400-e29b-41d4-a716-446655440000",
  "integrations": {
    "docker": {
      "enabled": false,
      "description": "Monitor Docker containers, images, volumes, and networks.
Collects real-time container status events."
    }
  }
}
```

Delete Host

Delete a specific host and all related data (cascade). This permanently removes the host and its associated packages, repositories, update history, Docker data, job history, and group memberships.

Endpoint:

```
DELETE /api/v1/api/hosts/:id
```

Required Scope: `host:delete`

Path Parameters:

Parameter	Type	Required	Description
`id`	string (UUID)	Yes	The unique identifier of the host to delete

Response (200 OK):

```
{
  "message": "Host deleted successfully",
  "deleted": {
    "id": "550e8400-e29b-41d4-a716-446655440000",
    "friendly_name": "web-server-01",
    "hostname": "web01.example.com"
  }
}
```

Response Fields:

Field	Type	Description
`message`	string	Confirmation message
`deleted.id`	string (UUID)	The ID of the deleted host
`deleted.friendly_name`	string	The friendly name of the deleted host
`deleted.hostname`	string	The hostname of the deleted host

Error Responses:

HTTP Code	Error	Description
400	`Invalid host ID format`	The provided ID is not a valid UUID
403	`Access denied`	Credential does not have `host:delete` permission
404	`Host not found`	No host exists with the given ID
500	`Failed to delete host`	Unexpected error during host deletion

> ***Warning:*** *This action is **irreversible**. All data associated with the host (packages, repositories, update history, Docker containers, job history, group memberships, etc.) will be permanently deleted.*

Common Error Responses (All Endpoints)

404 Not Found: Host does not exist (for single-host endpoints):

```
{
  "error": "Host not found"
}
```

500 Internal Server Error: Unexpected server error:

```
{
  "error": "Failed to fetch hosts"
}
```

See the Troubleshooting section for authentication and permission errors.

Usage Examples

cURL Examples

List All Hosts

```
curl -u "patchmon_ae_abc123:your_secret_here" \
  https://patchmon.example.com/api/v1/api/hosts
```

List Hosts with Stats

```
curl -u "patchmon_ae_abc123:your_secret_here" \
  "https://patchmon.example.com/api/v1/api/hosts?include=stats"
```

Filter by Host Group

```
curl -u "patchmon_ae_abc123:your_secret_here" \
  "https://patchmon.example.com/api/v1/api/hosts?hostgroup=Production"
```

Filter by Host Group with Stats

```
curl -u "patchmon_ae_abc123:your_secret_here" \
  "https://patchmon.example.com/api/v1/api/hosts?
hostgroup=Production&include=stats"
```

Filter by Multiple Groups

```
curl -u "patchmon_ae_abc123:your_secret_here" \
  "https://patchmon.example.com/api/v1/api/hosts?hostgroup=Production,Development"
```

Get Host Statistics

```
curl -u "patchmon_ae_abc123:your_secret_here" \
  https://patchmon.example.com/api/v1/api/hosts/HOST_UUID/stats
```

Get Host System Information

```
curl -u "patchmon_ae_abc123:your_secret_here" \
  https://patchmon.example.com/api/v1/api/hosts/HOST_UUID/system
```

Get All Packages for a Host

```
curl -u "patchmon_ae_abc123:your_secret_here" \
  https://patchmon.example.com/api/v1/api/hosts/HOST_UUID/packages
```

Get Only Packages with Available Updates

```
curl -u "patchmon_ae_abc123:your_secret_here" \
  "https://patchmon.example.com/api/v1/api/hosts/HOST_UUID/packages?
updates_only=true"
```

Delete a Host

```
curl -X DELETE -u "patchmon_ae_abc123:your_secret_here" \
  https://patchmon.example.com/api/v1/api/hosts/HOST_UUID
```

Pretty Print JSON Output

```
curl -u "patchmon_ae_abc123:your_secret_here" \
  https://patchmon.example.com/api/v1/api/hosts | jq .
```

Python Examples

Using `requests` Library

```
import requests
from requests.auth import HTTPBasicAuth

# API credentials
API_KEY = "patchmon_ae_abc123"
API_SECRET = "your_secret_here"
BASE_URL = "https://patchmon.example.com"

# Create session with authentication
session = requests.Session()
session.auth = HTTPBasicAuth(API_KEY, API_SECRET)

# List all hosts
response = session.get(f"{BASE_URL}/api/v1/api/hosts")

if response.status_code == 200:
    data = response.json()
    print(f"Total hosts: {data['total']}")

    for host in data['hosts']:
        groups = ', '.join([g['name'] for g in host['host_groups']])
        print(f"  {host['friendly_name']} ({host['ip']}) - Groups: {groups}")
else:
    print(f"Error: {response.status_code} - {response.json()}")
```

Filter by Host Group

```
# Filter by group name (requests handles URL encoding automatically)
response = session.get(
    f"{BASE_URL}/api/v1/api/hosts",
    params={"hostgroup": "Production"}
)
```

List Hosts with Inline Stats

```
# Get hosts with stats in a single request (more efficient than per-host /stats
calls)
response = session.get(
    f"{BASE_URL}/api/v1/api/hosts",
    params={"include": "stats"}
)

if response.status_code == 200:
    data = response.json()
    for host in data['hosts']:
        print(f"{host['friendly_name']}: {host['updates_count']} updates, "
              f"{host['security_updates_count']} security, "
              f"{host['total_packages']} total packages")
```

Get Host Packages (Updates Only)

```
# Get only packages that need updates for a specific host
response = session.get(
    f"{BASE_URL}/api/v1/api/hosts/{host_id}/packages",
    params={"updates_only": "true"}
)

if response.status_code == 200:
    data = response.json()
    print(f"Host: {data['host']['friendly_name']}")
    print(f"Packages needing updates: {data['total']}")
    for pkg in data['packages']:
        security = " [SECURITY]" if pkg['is_security_update'] else ""
        print(f"  {pkg['name']}: {pkg['current_version']} →
{pkg['available_version']}{security}")
```

Get Host Details and Stats

```
# First, get list of hosts
hosts_response = session.get(f"{BASE_URL}/api/v1/api/hosts")
hosts = hosts_response.json()['hosts']

# Then get stats for the first host
if hosts:
    host_id = hosts[0]['id']

    stats = session.get(f"{BASE_URL}/api/v1/api/hosts/{host_id}/stats").json()
    print(f"Installed: {stats['total_installed_packages']}")
    print(f"Outdated: {stats['outdated_packages']}")
    print(f"Security: {stats['security_updates']}")

    info = session.get(f"{BASE_URL}/api/v1/api/hosts/{host_id}/info").json()
    print(f"OS: {info['os_type']} {info['os_version']}")
    print(f"Agent: {info['agent_version']}")
```

Delete a Host

```
# Delete a host by UUID (requires host:delete scope)
host_id = "550e8400-e29b-41d4-a716-446655440000"
response = session.delete(f"{BASE_URL}/api/v1/api/hosts/{host_id}")

if response.status_code == 200:
    data = response.json()
    print(f"Deleted: {data['deleted']['friendly_name']} ({data['deleted']
['hostname']})")
else:
    print(f"Error: {response.status_code} - {response.json()}")
```

Error Handling

```
def get_hosts(hostgroup=None):
    """Get hosts with error handling."""
    try:
        params = {"hostgroup": hostgroup} if hostgroup else {}
        response = session.get(
            f"{BASE_URL}/api/v1/api/hosts",
            params=params,
            timeout=30
        )
        response.raise_for_status()
        return response.json()

    except requests.exceptions.HTTPError as e:
        if e.response.status_code == 401:
            print("Authentication failed - check credentials")
        elif e.response.status_code == 403:
            print("Access denied - insufficient permissions")
        else:
            print(f"HTTP error: {e}")
        return None

    except requests.exceptions.Timeout:
        print("Request timed out")
        return None

    except requests.exceptions.RequestException as e:
        print(f"Request failed: {e}")
        return None
```

Generate Ansible Inventory

```
import json
import requests
from requests.auth import HTTPBasicAuth

API_KEY = "patchmon_ae_abc123"
API_SECRET = "your_secret_here"
BASE_URL = "https://patchmon.example.com"

def generate_ansible_inventory():
    """Generate Ansible inventory from PatchMon hosts."""
    auth = HTTPBasicAuth(API_KEY, API_SECRET)
    response = requests.get(f"{BASE_URL}/api/v1/api/hosts", auth=auth, timeout=30)

    if response.status_code != 200:
        print(f"Error fetching hosts: {response.status_code}")
        return

    data = response.json()

    inventory = {
        "_meta": {"hostvars": {}},
        "all": {"hosts": [], "children": []}
    }

    for host in data['hosts']:
        hostname = host['friendly_name']
        inventory["all"]["hosts"].append(hostname)

        inventory["_meta"]["hostvars"][hostname] = {
            "ansible_host": host['ip'],
            "patchmon_id": host['id'],
            "patchmon_hostname": host['hostname']
        }

        for group in host['host_groups']:
            group_name = group['name'].lower().replace(' ', '_')

            if group_name not in inventory:
                inventory[group_name] = {"hosts": [], "vars": {}}
                inventory["all"]["children"].append(group_name)

            inventory[group_name]["hosts"].append(hostname)

    print(json.dumps(inventory, indent=2))

if __name__ == "__main__":
    generate_ansible_inventory()
```

JavaScript/Node.js Examples

Using Native `fetch` (Node.js 18+)

```
const API_KEY = 'patchmon_ae_abc123';
const API_SECRET = 'your_secret_here';
const BASE_URL = 'https://patchmon.example.com';

const authHeader = 'Basic ' +
Buffer.from(`${API_KEY}:${API_SECRET}`).toString('base64');

async function getHosts(hostgroup = null) {
  const url = new URL('/api/v1/api/hosts', BASE_URL);
  if (hostgroup) {
    url.searchParams.append('hostgroup', hostgroup);
  }

  const response = await fetch(url, {
    headers: {
      'Authorization': authHeader,
      'Content-Type': 'application/json'
    }
  });

  if (!response.ok) {
    const error = await response.json();
    throw new Error(`HTTP ${response.status}: ${error.error}`);
  }

  return await response.json();
}

// List all hosts
getHosts()
  .then(data => {
    console.log(`Total: ${data.total}`);
    data.hosts.forEach(host => {
      console.log(`${host.friendly_name}: ${host.ip}`);
    });
  })
  .catch(error => console.error('Error:', error.message));
```

Ansible Dynamic Inventory

Save this as `patchmon_inventory.py` and make it executable (`chmod +x`):

```
#!/usr/bin/env python3
"""
PatchMon Dynamic Inventory Script for Ansible.
Usage: ansible-playbook -i patchmon_inventory.py playbook.yml
"""

import json
import os
import sys
import requests
from requests.auth import HTTPBasicAuth

API_KEY = os.environ.get('PATCHMON_API_KEY')
API_SECRET = os.environ.get('PATCHMON_API_SECRET')
BASE_URL = os.environ.get('PATCHMON_URL', 'https://patchmon.example.com')

if not API_KEY or not API_SECRET:
    print("Error: PATCHMON_API_KEY and PATCHMON_API_SECRET must be set",
file=sys.stderr)
    sys.exit(1)

def get_inventory():
    auth = HTTPBasicAuth(API_KEY, API_SECRET)
    try:
        response = requests.get(f"{BASE_URL}/api/v1/api/hosts", auth=auth,
timeout=30)
        response.raise_for_status()
        return response.json()
    except requests.exceptions.RequestException as e:
        print(f"Error fetching inventory: {e}", file=sys.stderr)
        sys.exit(1)

def build_ansible_inventory(patchmon_data):
    inventory = {
        "_meta": {"hostvars": {}},
        "all": {"hosts": []}
    }
    groups = {}

    for host in patchmon_data['hosts']:
        hostname = host['friendly_name']
        inventory["all"]["hosts"].append(hostname)

        inventory["_meta"]["hostvars"][hostname] = {
            "ansible_host": host['ip'],
            "patchmon_id": host['id'],
            "patchmon_hostname": host['hostname']
        }

        for group in host['host_groups']:
            group_name = group['name'].lower().replace(' ', '_').replace('-', '_')
```

```
            if group_name not in groups:
                groups[group_name] = {
                    "hosts": [],
                    "vars": {"patchmon_group_id": group['id']}
                }
            groups[group_name]["hosts"].append(hostname)

    inventory.update(groups)
    return inventory

def main():
    if len(sys.argv) == 2 and sys.argv[1] == '--list':
        patchmon_data = get_inventory()
        inventory = build_ansible_inventory(patchmon_data)
        print(json.dumps(inventory, indent=2))
    elif len(sys.argv) == 3 and sys.argv[1] == '--host':
        print(json.dumps({}))
    else:
        print("Usage: patchmon_inventory.py --list", file=sys.stderr)
        sys.exit(1)

if __name__ == '__main__':
    main()
```

Usage:

```
export PATCHMON_API_KEY="patchmon_ae_abc123"
export PATCHMON_API_SECRET="your_secret_here"
export PATCHMON_URL="https://patchmon.example.com"

# Test inventory
./patchmon_inventory.py --list

# Use with ansible
ansible-playbook -i patchmon_inventory.py playbook.yml
ansible -i patchmon_inventory.py all -m ping
```

Security Best Practices

Credential Management

Do:

- Store credentials in a password manager or secrets vault (e.g. HashiCorp Vault, AWS Secrets Manager)
- Use environment variables for automation scripts
- Set expiration dates (recommended: 90 days)

- Grant only the minimum permissions needed (principle of least privilege)
- Rotate credentials regularly and delete old ones after migration

Don't:

- Hard-code credentials in source code
- Commit credentials to version control
- Share credentials via email or chat
- Store credentials in plain-text files

IP Restrictions

Restrict credentials to known IP addresses whenever possible:

```
Allowed IPs: 192.168.1.100, 10.0.0.0/24
```

For dynamic IPs, consider using a VPN with a static exit IP, a cloud NAT gateway, or a proxy server.

Network Security

- **Always use HTTPS** in production environments
- **Verify SSL certificates**: only disable verification (`-k`) for development/testing
- **Use firewall rules** to restrict PatchMon API access at the network level

Monitoring & Auditing

- Check "Last Used" timestamps regularly in the Integrations settings page
- Investigate credentials that have not been used in 30+ days
- Review all active credentials monthly
- Remove credentials for decommissioned systems

If Credentials Are Compromised

- **Immediately disable** the credential in PatchMon UI (Settings → Integrations → toggle off)
- **Review the "Last Used" timestamp** to understand the window of exposure
- **Check server logs** for any unauthorised access
- **Create new credentials** with a different scope if needed
- **Delete the compromised credential** after verification
- **Notify your security team** if sensitive data may have been accessed

---

Troubleshooting

Error Reference

Error Message	HTTP Code	Cause	Solution
`Missing or invalid authorization header`	401	No `Authorization` header, or it doesn't start with `Basic`	Use `-u key:secret` with cURL, or set `Authorization: Basic <base64>` header
`Invalid credentials format`	401	Base64-decoded value doesn't contain a colon separator	Check format is `key:secret` and ensure no extra characters
`Invalid API key`	401	Token key not found in the database	Verify the credential exists in Settings → Integrations
`API key is disabled`	401	Credential has been manually deactivated	Re-enable in Settings → Integrations, or create a new credential
`API key has expired`	401	The expiration date has passed	Create a new credential to replace the expired one
`Invalid API key type`	401	The credential's `integration_type` is not `"api"`	Ensure you created the credential with the "API" usage type
`Invalid API secret`	401	Secret doesn't match the stored bcrypt hash	Create a new credential (secrets cannot be retrieved)
`IP address not allowed`	403	Client IP is not in the credential's `allowed_ip_ranges`	Add your IP: `curl https://ifconfig.me` to find it
`Access denied: does not have permission to {action} {resource}`	403	Credential is missing the required scope	Edit the credential and add the required permission
`Access denied: does not have access to {resource}`	403	The resource is not included in the credential's scopes at all	Edit the credential's scopes to include the resource
`Host not found`	404	The host UUID does not exist	Verify the UUID from the list hosts endpoint
`Invalid host ID format`	400	The host ID is not a valid UUID (DELETE endpoint)	Ensure the ID is a valid UUID format
`Failed to delete host`	500	Unexpected error during host deletion	Check PatchMon server logs for details

Error Message	HTTP Code	Cause	Solution
`Failed to fetch hosts`	500	Unexpected server error	Check PatchMon server logs for details
`Authentication failed`	500	Unexpected error during authentication processing	Check PatchMon server logs; may indicate a database issue

Debug Tips

cURL verbose mode:

```
curl -v -u "patchmon_ae_abc123:your_secret_here" \
  https://patchmon.example.com/api/v1/api/hosts
```

Python debug logging:

```
import logging
logging.basicConfig(level=logging.DEBUG)
requests_log = logging.getLogger("requests.packages.urllib3")
requests_log.setLevel(logging.DEBUG)
requests_log.propagate = True
```

Common Issues

Empty hosts array

- Verify hosts exist in PatchMon UI → Hosts page
- Check the `hostgroup` filter spelling matches exactly (case-sensitive)
- Try listing all hosts without filters first to confirm API access works

Connection timeouts

```
# Test basic connectivity
ping patchmon.example.com
curl -I https://patchmon.example.com/health
```

SSL certificate errors

For development/testing with self-signed certificates:

```
curl -k -u "patchmon_ae_abc123:your_secret_here" \
  https://patchmon.example.com/api/v1/api/hosts
```

For production, install a valid SSL certificate (e.g. Let's Encrypt).

Getting Help

If issues persist:

- Check PatchMon server logs for detailed error information
- Use the built-in Swagger UI to test endpoints interactively
- Search or create an issue at github.com/PatchMon/PatchMon (https://github.com/PatchMon/PatchMon/issues)
- Join the PatchMon community on Discord (https://patchmon.net/discord)

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