Integration API documentation (scoped credentials)

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

1. Log in to your PatchMon instance as an administrator
2. Go to **Settings** → **Integrations**
3. 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.

Format

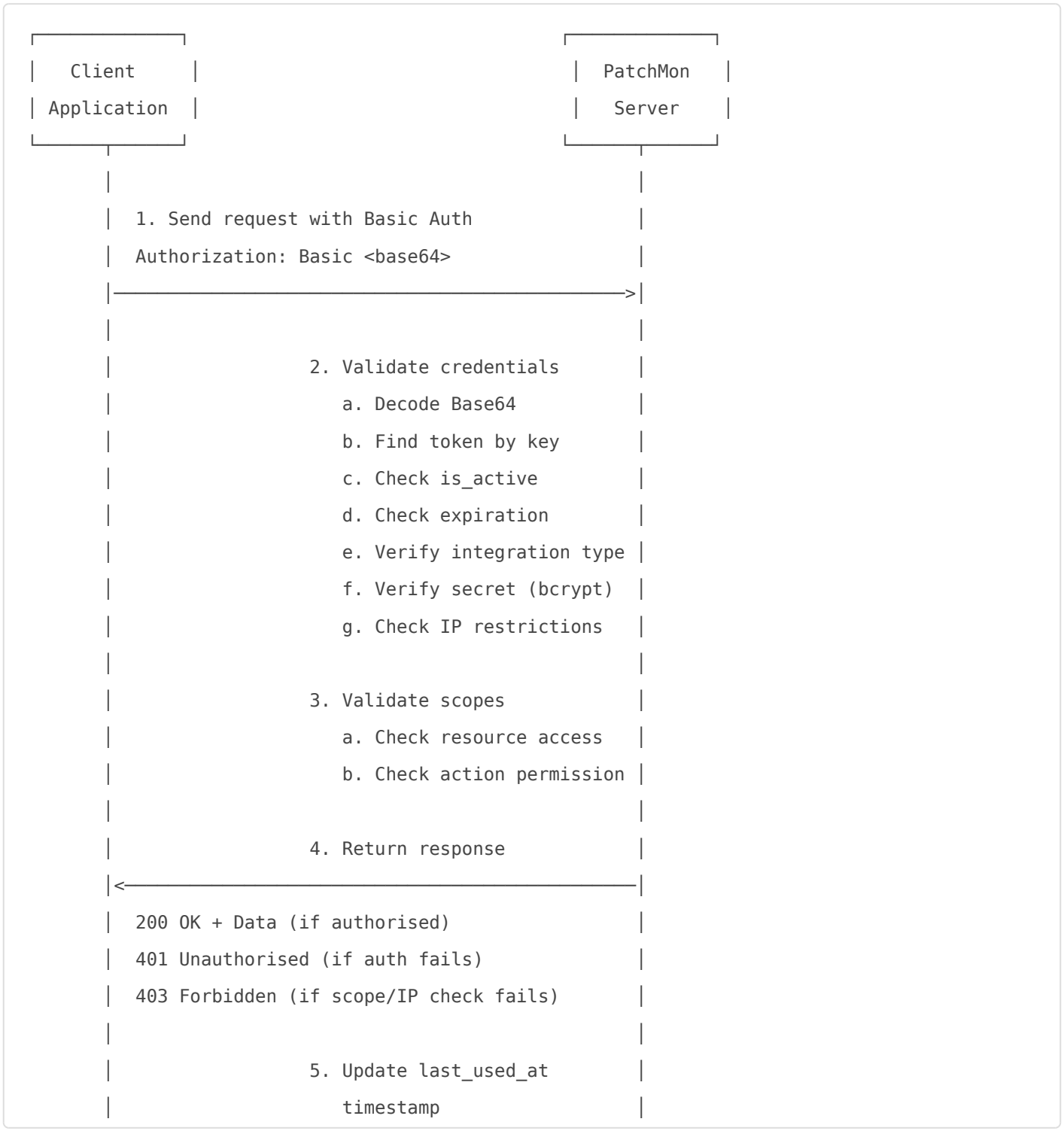
```
Authorization: Basic <base64(token_key:token_secret)>
```

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
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)
`put`	Replace host data
`patch`	Partially update host data
`update`	General update operations
`delete`	Delete hosts

Example scope configurations:

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

// Read and update
```

```
{ "host": ["get", "patch"] }

// Full access
{ "host": ["get", "put", "patch", "update", "delete"] }
```

Important Notes

- Scopes are **explicit** — no inheritance or wildcards. Each action must be explicitly granted.
- `get` does **not** automatically include `patch` 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

Endpoint	Method	Scope	Description
`/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 `?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",
      "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`)

Field	Type	Description
`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	Host status, e.g. `active`, `pending` (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
`filtered_by_groups`	array	Groups used for filtering (only present when filtering)

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": "1.4.0",
  "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 MB",
  "swap_size": "2048 MB",
  "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	—	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
400	`Cannot delete host due to foreign key constraints`	The host has related data that prevents deletion
404	`Host not found`	No host exists with the given ID
403	`Access denied`	Credential does not have `host:delete` permission

> ⚠ **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
```

Delete a Host

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

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"
```

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

1. **Immediately disable** the credential in PatchMon UI (Settings → Integrations → toggle off)
2. **Review the "Last Used" timestamp** to understand the window of exposure
3. **Check server logs** for any unauthorised access
4. **Create new credentials** with a different scope if needed
5. **Delete the compromised credential** after verification
6. **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` — 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

Error Message	HTTP Code	Cause	Solution
`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
`Cannot delete host due to foreign key constraints`	400	Host has related data preventing deletion	Check PatchMon server logs for details
`Failed to delete host`	500	Unexpected error during host deletion	Check PatchMon server logs for details
`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:

1. Check PatchMon server logs for detailed error information
2. Use the built-in Swagger UI to test endpoints interactively
3. Search or create an issue at github.com/PatchMon/PatchMon
4. Join the PatchMon community on Discord

Revision #7
Created 2025-11-10 21:12:40 UTC by Iby
Updated 2026-02-12 14:31:02 UTC by Iby