Installing PatchMon Server on Docker

Overview

PatchMon is a containerised application that monitors system patches and updates. The application consists of four main services:

- **Database**: PostgreSQL 17
- **Redis**: Redis 7 for BullMQ job queues and caching
- **Server**: Go API server with embedded frontend (serves both API and static files)
- **guacd**: Apache Guacamole daemon for in-browser RDP (Windows hosts)

Images

- **Server**: ghcr.io/patchmon/patchmon-server

Tags

- `latest`: The latest stable release of PatchMon
- `x.y.z`: Full version tags (e.g. `1.2.3`) - Use this for exact version pinning.
- `x.y`: Minor version tags (e.g. `1.2`) - Use this to get the latest patch release in a minor version series.
- `x`: Major version tags (e.g. `1`) - Use this to get the latest minor and patch release in a major version series.
- `edge`: The latest development build in main branch. This tag may often be unstable and is intended only for testing and development purposes.

Quick Start

Production Deployment

Automated (recommended)

Run the setup script from an empty directory. It will download `docker-compose.yml` and `env.example`, generate all required secrets, and walk you through configuring your URL and timezone interactively:

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

Once the script finishes, start PatchMon:

```
docker compose up -d
```

Access the application at the URL you configured (default: `http://localhost:3000`).

Manual

1. Download the Docker Compose file and environment example:

   ```
   mkdir patchmon && cd patchmon
   curl -fsSL -o docker-compose.yml https://raw.githubusercontent.com/PatchMon/PatchMon/refs/heads/main/docker/docker-compose.yml
   curl -fsSL -o env.example https://raw.githubusercontent.com/PatchMon/PatchMon/refs/heads/main/docker/env.example
   ```

2. Create your `.env` file from the example:

   ```
   cp env.example .env
   ```

3. Generate and insert the required secrets:

   ```
   sed -i "s/^POSTGRES_PASSWORD=.*/POSTGRES_PASSWORD=$(openssl rand -hex 32)/" .env
   sed -i "s/^REDIS_PASSWORD=.*/REDIS_PASSWORD=$(openssl rand -hex 32)/" .env
   sed -i "s/^JWT_SECRET=.*/JWT_SECRET=$(openssl rand -hex 64)/" .env
   sed -i "s/^SESSION_SECRET=.*/SESSION_SECRET=$(openssl rand -hex 64)/" .env
   sed -i "s/^AI_ENCRYPTION_KEY=.*/AI_ENCRYPTION_KEY=$(openssl rand -hex 64)/" .env
   ```

4. Edit `.env` and configure the required variables. See `env.example` for the full list and docs.patchmon.net for detailed explanations.
5. Start the application:

   ```
   docker compose up -d
   ```

6. Access the application at `http://localhost:3000`

The `docker-compose.yml` reads all configuration from your `.env` file. You do not need to edit the compose file itself.

Updating

By default, the compose file uses the `latest` tag for the server image.

This means you can update PatchMon to the latest version as easily as:

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

This command will:

- Pull the latest images from the registry
- Recreate containers with updated images
- Maintain your data and configuration

Version-Specific Updates

If you'd like to pin your Docker deployment of PatchMon to a specific version, you can do this in the compose file.

When you do this, updating to a new version requires manually updating the image tags in the compose file yourself:

1. Update the image tag in `docker-compose.yml`. For example:
   ```
   services:
     server:
       image: ghcr.io/patchmon/patchmon-server:1.2.3  # Update version here
      ...
   ```
2. Then run the update command:
   ```
   docker compose pull
   docker compose up -d
   ```

> [!TIP] Check the releases page for version-specific changes and migration notes.

Configuration

All configuration is managed through the `.env` file.

For the full list of available variables, see `env.example` in this directory.

For detailed explanations of each variable (defaults, usage, and examples), see the PatchMon Environment Variables Reference at docs.patchmon.net.

Volumes

The compose file creates two Docker volumes:

- `postgres_data`: PostgreSQL's data directory.
- `redis_data`: Redis's data directory.

Agent binaries are included in the server image at `/app/agents` and served read-only. Deploy or pull a new image to update agents.

Frontend assets (JS, CSS, default logos) are embedded in the server binary. Custom logos are stored in the database and served via the API.

If you wish to bind any of their respective container paths to a host path rather than a Docker volume, you can do so in the Docker Compose file.

Docker Swarm Deployment

> [!NOTE] This section covers deploying PatchMon to a Docker Swarm cluster. For standard Docker Compose deployments on a single host, use the production deployment guide above.

Network Configuration

When deploying to Docker Swarm with a reverse proxy (e.g., Traefik), proper network configuration is critical. The default `docker-compose.yml` uses an internal bridge network that enables service-to-service communication:

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

All services (database, redis, and server) connect to this internal network, allowing them to discover each other by service name.

Important: If you're using an external reverse proxy network (like `traefik-net`), ensure that:

1. All PatchMon services remain on the `patchmon-internal` network for internal communication
2. The server service can be configured to also bind to the reverse proxy network if needed
3. Service names resolve correctly within the same network

Service Discovery in Swarm

In Docker Swarm, service discovery works through:

- **Service Name Resolution**: Service names resolve to virtual IPs within the same network
- **Load Balancing**: Requests to a service name are automatically load-balanced across all replicas
- **Network Isolation**: Services on different networks cannot communicate directly

Configuration for Swarm with Traefik

If you're using Traefik as a reverse proxy:

1. Keep the default `patchmon-internal` network for server services
2. Configure Traefik in your Swarm deployment with its own network
3. Ensure the server service is reachable through the internal network

Example modification for Swarm:

```
services:
  server:
    image: ghcr.io/patchmon/patchmon-server:latest
    networks:
      - patchmon-internal
    deploy:
      replicas: 1
      labels:
        - "traefik.enable=true"
```

```
      - "traefik.http.routers.patchmon.rule=Host(`patchmon.my.domain`)"
      # ... other Traefik labels
```

Traefik routes external traffic to the server service, which serves both the API and frontend.

Troubleshooting Network Issues

Error: `host not found in upstream "server"`

This typically occurs when:

1. Services are on different networks
2. Services haven't fully started (check health checks)
3. Service names haven't propagated through DNS

Solution:

- Verify all services are on the same internal network
- Check service health status: `docker ps` (production) or `docker service ps` (Swarm)
- Wait for health checks to pass before accessing the application
- Confirm network connectivity: `docker exec <container> ping server`

Development

This section is for developers who want to contribute to PatchMon or run it in development mode.

Development Setup

For development with live reload and source code mounting:

1. Clone the repository:

   ```
   git clone https://github.com/PatchMon/PatchMon.git
   cd PatchMon
   ```

2. Start development environment:

   ```
   docker compose -f docker/docker-compose.dev.yml up
   ```

   See Development Commands for more options.

3. Access the application:
   - Application (API + frontend): `http://localhost:3000`
   - Database: `localhost:5432`
   - Redis: `localhost:6379`

Development Docker Compose

The development compose file (`docker/docker-compose.dev.yml`):

- Builds images locally from source using development targets
- Enables hot reload with Docker Compose watch functionality
- Exposes database and backend ports for testing and development
- Mounts source code directly into containers for live development
- Supports debugging with enhanced logging

Building Images Locally

Both Dockerfiles use multi-stage builds with separate development and production targets.

Note: When using locally-built images (e.g. `patchmon-server:dev`), do **not** run `docker compose pull` for the full stack—Compose will try to pull that name from Docker Hub and fail (it exists only on your machine). Use `docker compose up -d` so Compose uses your local image, or run `docker compose pull database redis` if you only want to refresh Postgres/Redis.

Server agent binaries: The server image includes agent scripts and prebuilt binaries. To build locally, run `make build-all-for-docker` in `agent-source-code/` first so `agents-prebuilt/` is populated.

```
# Build development image
docker build -f docker/server.Dockerfile --target development --provenance=false --sbom=false
-t patchmon-server:dev .

# Build production image (default target)
docker build -f docker/server.Dockerfile --provenance=false --sbom=false -t patchmon-
server:latest .
```

Development Commands

Hot Reload Development

```
# Attached, live log output, services stopped on Ctrl+C
docker compose -f docker/docker-compose.dev.yml up

# Attached with Docker Compose watch for hot reload
docker compose -f docker/docker-compose.dev.yml up --watch

# Detached
docker compose -f docker/docker-compose.dev.yml up -d

# Quiet, no log output, with Docker Compose watch for hot reload
docker compose -f docker/docker-compose.dev.yml watch
```

Rebuild Services

```
# Rebuild specific service
docker compose -f docker/docker-compose.dev.yml up -d --build server

# Rebuild all services
docker compose -f docker/docker-compose.dev.yml up -d --build
```

Development Ports

The development setup exposes additional ports for debugging:

- **Database**: `5432` - Direct PostgreSQL access
- **Redis**: `6379` - Direct Redis access
- **Backend**: `3001` - API server with development features
- **Frontend**: `3000` - React development server with hot reload

Development Workflow

1. **Initial Setup**: Clone repository and start development environment

   ```
   git clone https://github.com/PatchMon/PatchMon.git
   cd PatchMon
   docker compose -f docker/docker-compose.dev.yml up -d --build
   ```

2. **Hot Reload Development**: Use Docker Compose watch for automatic reload

```
docker compose -f docker/docker-compose.dev.yml up --watch --build
```

3. **Code Changes**:
   - **Frontend/Backend Source**: Files are synced automatically with watch mode
   - **Package.json Changes**: Triggers automatic service rebuild
   - **Prisma Schema Changes**: Backend service restarts automatically
4. **Database Access**: Connect database client directly to `localhost:5432`
5. **Redis Access**: Connect Redis client directly to `localhost:6379`
6. **Debug**: If started with `docker compose [...] up -d` or `docker compose [...] watch`, check logs manually:

   ```
   docker compose -f docker/docker-compose.dev.yml logs -f
   ```

   Otherwise logs are shown automatically in attached modes (`up`, `up --watch`).

Features in Development Mode

- **Hot Reload**: Automatic code synchronization and service restarts
- **Enhanced Logging**: Detailed logs for debugging
- **Direct Access**: Exposed ports for database, Redis, and API debugging
- **Health Checks**: Built-in health monitoring for services
- **Volume Persistence**: Development data persists between restarts

---

Revision #9
Created 2025-10-04 17:39:39 UTC by Iby
Updated 2026-04-23 21:04:44 UTC by Iby