Installing PatchMon Server on K8S with Helm

PatchMon Helm Chart Documentation

Helm chart for deploying PatchMon on Kubernetes.

- Chart repository: github.com/RuTHlessBEat200/PatchMon-helm
- Application repository: github.com/PatchMon/PatchMon

Overview

PatchMon v2.0.0 runs as a containerised application made up of four services:

- **Database** -- PostgreSQL 18 (StatefulSet with persistent storage)
- **Redis** -- Redis 8 (used for BullMQ job queues and caching, StatefulSet with persistent storage)
- **Server** -- Single Go binary serving both the backend API and the React frontend static files (StatefulSet with optional HPA)
- **Guacd** -- Apache Guacamole proxy daemon for remote desktop / SSH terminal access (Deployment)

> **Note:** In v2.0.0 the separate backend and frontend containers have been merged into a single `server` binary. If you are upgrading from v1.x, update your values files accordingly — `backend.*` and `frontend.*` keys no longer exist.

The chart deploys all four components into a single namespace and wires them together automatically using init containers, internal ClusterIP services, and a shared ConfigMap.

Container Images

Component	Image	Default Tag
Server	ghcr.io/patchmon/patchmon-server	`2.0.0`
Database	docker.io/postgres	`18-alpine`
Redis	docker.io/redis	`8-alpine`
Guacd	docker.io/guacamole/guacd	`latest`

Available Tags (Server)

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

---

Prerequisites

- Kubernetes 1.19+
- Helm 3.0+
- A PersistentVolume provisioner in the cluster (for database and Redis storage)
- (Optional) An Ingress controller (e.g. NGINX Ingress) for external access
- (Optional) cert-manager for automatic TLS certificate management
- (Optional) Metrics Server for HPA functionality

Quick Start

The quickest way to get PatchMon running is to use the provided `values-quick-start.yaml` file. It contains all required secrets inline and sensible defaults so you can install with a single command.

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

1. Install the chart

```
wget https://github.com/RuTHlessBEat200/PatchMon-helm/blob/main/values-quick-start.yaml
helm install patchmon oci://ghcr.io/ruthlessbeat200/charts/patchmon \
  --namespace patchmon \
  --create-namespace \
  --values values-quick-start.yaml
```

2. Wait for pods to become ready

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

3. Access PatchMon

If ingress is enabled, open the host you configured (e.g. `https://patchmon-dev.example.com`).

Without ingress, use port-forwarding:

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

Then navigate to `http://localhost:3000` and complete the first-time setup to create your admin account.

Production Deployment

For production use, refer to the provided `values-prod.yaml` file as a starting point. It demonstrates how to:

- Use an external secret (e.g. managed by KSOPS, Sealed Secrets, or External Secrets Operator) instead of inline passwords
- Configure HTTPS with cert-manager
- Set the correct server protocol, host, and port for agent communication

1. Create your secrets

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

Required secrets:

Key	Description
`postgres-password`	PostgreSQL password
`redis-password`	Redis password
`jwt-secret`	JWT signing secret for the server
`ai-encryption-key`	Encryption key for AI provider credentials
`oidc-client-secret`	OIDC client secret (only if OIDC is enabled)

You can either:

- Set passwords directly in your values file (`database.auth.password`, `redis.auth.password`, `server.jwtSecret`, `server.aiEncryptionKey`), or
- Create a Kubernetes Secret separately and reference it with `existingSecret` / `existingSecretPasswordKey` fields.

Example -- creating a secret manually:

```
kubectl create namespace patchmon

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

Secret management tools for production:

- KSOPS -- encrypt secrets in Git using Mozilla SOPS
- Sealed Secrets -- encrypt secrets that only the cluster can decrypt
- External Secrets Operator -- sync secrets from external stores (Vault, AWS Secrets Manager, etc.)
- Vault -- enterprise-grade secret management

2. Create your values file

Start from `values-prod.yaml` and adjust to your environment:

```
global:
  storageClass: "your-storage-class"

fullnameOverride: "patchmon-prod"

server:
  env:
    serverProtocol: https
    serverHost: patchmon.example.com
    serverPort: "443"
    corsOrigin: https://patchmon.example.com
  existingSecret: "patchmon-secrets"
  existingSecretJwtKey: "jwt-secret"
  existingSecretAiEncryptionKey: "ai-encryption-key"

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

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

secret:
  create: false   # Disable chart-managed secret since we use an external one

ingress:
  enabled: true
  className: nginx
  annotations:
    cert-manager.io/cluster-issuer: letsencrypt-prod
    nginx.ingress.kubernetes.io/proxy-body-size: "0"
    nginx.ingress.kubernetes.io/proxy-read-timeout: "3600"
    nginx.ingress.kubernetes.io/proxy-send-timeout: "3600"
    nginx.ingress.kubernetes.io/proxy-connect-timeout: "60"
    nginx.ingress.kubernetes.io/proxy-http-version: "1.1"
```

```
    nginx.ingress.kubernetes.io/client-body-buffer-size: "4m"
    nginx.ingress.kubernetes.io/websocket-services: "server"
  hosts:
    - host: patchmon.example.com
      paths:
        - path: /
          pathType: Prefix
          service:
            name: server
            port: 3000
  tls:
    - secretName: patchmon-tls
      hosts:
        - patchmon.example.com
```

3. Install

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

Configuration Reference

Global Settings

Parameter	Description	Default
`global.imageRegistry`	Override the image registry for all components	`""`
`global.imageTag`	Override the image tag for the server (takes priority over `server.image.tag` if set)	`""`
`global.imagePullSecrets`	Image pull secrets applied to all pods	`[]`
`global.storageClass`	Default storage class for all PVCs	`""`
`nameOverride`	Override the chart name used in resource names	`""`

Parameter	Description	Default
`fullnameOverride`	Override the full resource name prefix	`"patchmon-prod"`
`commonLabels`	Labels added to all resources	`{}`
`commonAnnotations`	Annotations added to all resources	`{}`

Database (PostgreSQL)

Parameter	Description	Default
`database.enabled`	Deploy the PostgreSQL StatefulSet	`true`
`database.image.registry`	Image registry	`docker.io`
`database.image.repository`	Image repository	`postgres`
`database.image.tag`	Image tag	`18-alpine`
`database.image.pullPolicy`	Image pull policy	`IfNotPresent`
`database.host`	External database host (overrides built-in service discovery when set)	`""`
`database.port`	External database port	`""`
`database.auth.database`	Database name	`patchmon_db`
`database.auth.username`	Database user	`patchmon_user`
`database.auth.password`	Database password (required if `existingSecret` is not set)	`""`
`database.auth.existingSecret`	Name of an existing secret containing the password	`""`
`database.auth.existingSecretPasswordKey`	Key inside the existing secret	`postgres-password`
`database.replicaCount`	Number of replicas	`1`
`database.updateStrategy.type`	StatefulSet update strategy	`RollingUpdate`
`database.persistence.enabled`	Enable persistent storage	`true`
`database.persistence.storageClass`	Storage class (falls back to `global.storageClass`)	`""`
`database.persistence.accessModes`	PVC access modes	`["ReadWriteOnce"]`
`database.persistence.size`	PVC size	`5Gi`
`database.resources.requests.cpu`	CPU request	`100m`
`database.resources.requests.memory`	Memory request	`128Mi`
`database.resources.limits.cpu`	CPU limit	`1000m`
`database.resources.limits.memory`	Memory limit	`1Gi`

Parameter	Description	Default
`database.livenessProbe.enabled`	Enable liveness probe	`true`
`database.readinessProbe.enabled`	Enable readiness probe	`true`
`database.service.type`	Service type	`ClusterIP`
`database.service.port`	Service port	`5432`
`database.podAnnotations`	Pod annotations	`{}`
`database.podSecurityContext`	Pod-level security context	see `values.yaml`
`database.securityContext`	Container-level security context	see `values.yaml`
`database.nodeSelector`	Node selector	`{}`
`database.tolerations`	Tolerations	`[]`
`database.affinity`	Affinity rules	`{}`

Redis

Parameter	Description	Default
`redis.enabled`	Deploy the Redis StatefulSet	`true`
`redis.image.registry`	Image registry	`docker.io`
`redis.image.repository`	Image repository	`redis`
`redis.image.tag`	Image tag	`8-alpine`
`redis.image.pullPolicy`	Image pull policy	`IfNotPresent`
`redis.auth.password`	Redis password (required if `existingSecret` is not set)	`""`
`redis.auth.existingSecret`	Name of an existing secret containing the password	`""`
`redis.auth.existingSecretPasswordKey`	Key inside the existing secret	`redis-password`
`redis.replicaCount`	Number of replicas	`1`
`redis.updateStrategy.type`	StatefulSet update strategy	`RollingUpdate`
`redis.persistence.enabled`	Enable persistent storage	`true`
`redis.persistence.storageClass`	Storage class (falls back to `global.storageClass`)	`""`
`redis.persistence.accessModes`	PVC access modes	`["ReadWriteOnce"]`
`redis.persistence.size`	PVC size	`5Gi`
`redis.resources.requests.cpu`	CPU request	`50m`
`redis.resources.requests.memory`	Memory request	`10Mi`

Parameter	Description	Default
`redis.resources.limits.cpu`	CPU limit	`500m`
`redis.resources.limits.memory`	Memory limit	`512Mi`
`redis.livenessProbe.enabled`	Enable liveness probe	`true`
`redis.readinessProbe.enabled`	Enable readiness probe	`true`
`redis.service.type`	Service type	`ClusterIP`
`redis.service.port`	Service port	`6379`
`redis.podAnnotations`	Pod annotations	`{}`
`redis.podSecurityContext`	Pod-level security context	see `values.yaml`
`redis.securityContext`	Container-level security context	see `values.yaml`
`redis.nodeSelector`	Node selector	`{}`
`redis.tolerations`	Tolerations	`[]`
`redis.affinity`	Affinity rules	`{}`

Server

The `server` component is a single Go binary that serves both the backend API and the React frontend on port `3000`. It is deployed as a **StatefulSet**.

Parameter	Description	Default
`server.enabled`	Deploy the server	`true`
`server.image.registry`	Image registry	`ghcr.io`
`server.image.repository`	Image repository	`patchmon/patchmon-server`
`server.image.tag`	Image tag (overridden by `global.imageTag` if set)	`2.0.0`
`server.image.pullPolicy`	Image pull policy	`IfNotPresent`
`server.replicaCount`	Number of replicas	`1`
`server.updateStrategy.type`	StatefulSet update strategy	`RollingUpdate`
`server.jwtSecret`	JWT signing secret (required if `existingSecret` is not set)	`""`
`server.aiEncryptionKey`	AI encryption key (required if `existingSecret` is not set)	`""`
`server.existingSecret`	Name of an existing secret for JWT and AI encryption key	`""`
`server.existingSecretJwtKey`	Key for JWT secret inside the existing secret	`jwt-secret`

Parameter	Description	Default
`server.existingSecretAiEncryptionKey`	Key for AI encryption key inside the existing secret	`ai-encryption-key`
`server.resources.requests.cpu`	CPU request	`10m`
`server.resources.requests.memory`	Memory request	`256Mi`
`server.resources.limits.cpu`	CPU limit	`2000m`
`server.resources.limits.memory`	Memory limit	`2Gi`
`server.autoscaling.enabled`	Enable HPA	`false`
`server.autoscaling.minReplicas`	Minimum replicas	`1`
`server.autoscaling.maxReplicas`	Maximum replicas	`10`
`server.autoscaling.targetCPUUtilizationPercentage`	Target CPU utilisation	`80`
`server.autoscaling.targetMemoryUtilizationPercentage`	Target memory utilisation	`80`
`server.service.type`	Service type	`ClusterIP`
`server.service.port`	Service port	`3000`
`server.service.annotations`	Service annotations	`[]`
`server.livenessProbe.enabled`	Enable liveness probe (TCP on port 3000)	`true`
`server.readinessProbe.enabled`	Enable readiness probe (`GET /health` on port 3000)	`true`
`server.initContainers.waitForDatabase.enabled`	Wait for database before starting	`true`
`server.initContainers.waitForRedis.enabled`	Wait for Redis before starting	`true`
`server.initContainers.waitForGuacd.enabled`	Wait for guacd before starting	`true`
`server.extraEnv`	Extra environment variables to inject into the server container	`[]`
`server.extraVolumeMounts`	Extra volume mounts for the server container	`[]`
`server.extraVolumes`	Extra volumes to add to the server pod	`[]`
`server.podAnnotations`	Pod annotations	`{}`
`server.podSecurityContext`	Pod-level security context	see `values.yaml`
`server.securityContext`	Container-level security context	see `values.yaml`
`server.nodeSelector`	Node selector	`{}`
`server.tolerations`	Tolerations	`[]`

Parameter	Description	Default
`server.affinity`	Affinity rules	`{}`
`server.topologySpreadConstraints`	Topology spread constraints	`[]`

Server Environment Variables

Parameter	Description	Default
`server.env.enableLogging`	Enable application logging	`true`
`server.env.logLevel`	Log level (`trace`, `debug`, `info`, `warn`, `error`)	`info`
`server.env.logToConsole`	Log to stdout	`true`
`server.env.serverProtocol`	Protocol used by agents to reach the server (`http` or `https`)	`http`
`server.env.serverHost`	Hostname used by agents to reach the server	`patchmon.example.com`
`server.env.serverPort`	Port used by agents (`80` or `443`)	`80`
`server.env.corsOrigin`	CORS allowed origin (should match the URL users access in a browser)	`http://patchmon.example.com`
`server.env.dbConnectionLimit`	Database connection pool limit	`30`
`server.env.dbPoolTimeout`	Pool timeout in seconds	`20`
`server.env.dbConnectTimeout`	Connection timeout in seconds	`10`
`server.env.dbIdleTimeout`	Idle connection timeout in seconds	`300`
`server.env.dbMaxLifetime`	Max connection lifetime in seconds	`1800`
`server.env.rateLimitWindowMs`	General rate limit window (ms)	`900000`
`server.env.rateLimitMax`	General rate limit max requests	`5000`
`server.env.authRateLimitWindowMs`	Auth rate limit window (ms)	`600000`
`server.env.authRateLimitMax`	Auth rate limit max requests	`500`
`server.env.agentRateLimitWindowMs`	Agent rate limit window (ms)	`60000`
`server.env.agentRateLimitMax`	Agent rate limit max requests	`1000`
`server.env.redisDb`	Redis database index	`0`
`server.env.trustProxy`	Trust proxy headers -- set to a CIDR range or `true` when behind a reverse proxy	`10.0.0.0/8,172.16.0.0/12,192.168.0.0/16`
`server.env.enableHsts`	Enable HSTS header	`false`
`server.env.defaultUserRole`	Default role for new users	`user`
`server.env.autoCreateRolePermissions`	Auto-create role permissions	`false`

OIDC / SSO Configuration

Parameter	Description	Default
`server.oidc.enabled`	Enable OIDC authentication	`false`
`server.oidc.issuerUrl`	OIDC issuer URL	`""`
`server.oidc.clientId`	OIDC client ID	`""`
`server.oidc.clientSecret`	OIDC client secret (required if `existingSecret` not set)	`""`
`server.oidc.existingSecret`	Existing secret containing the OIDC client secret	`""`
`server.oidc.existingSecretClientSecretKey`	Key inside the existing secret	`oidc-client-secret`
`server.oidc.scopes`	OIDC scopes	`openid profile email`
`server.oidc.buttonText`	Login button text	`Login with SSO`
`server.oidc.autoCreateUsers`	Auto-create users on first OIDC login	`true`
`server.oidc.defaultRole`	Default role for OIDC-created users	`user`
`server.oidc.syncRoles`	Sync roles from OIDC group claims	`true`
`server.oidc.disableLocalAuth`	Disable local username/password authentication	`false`
`server.oidc.sessionTtl`	OIDC session TTL in seconds	`86400`
`server.oidc.groups.superadmin`	OIDC group mapped to the superadmin role	`""`
`server.oidc.groups.admin`	OIDC group mapped to the admin role	`""`
`server.oidc.groups.hostManager`	OIDC group mapped to the hostManager role	`""`
`server.oidc.groups.user`	OIDC group mapped to the user role	`""`
`server.oidc.groups.readonly`	OIDC group mapped to the readonly role	`""`

Guacd

Apache Guacamole proxy daemon used for browser-based SSH and remote desktop sessions.

Parameter	Description	Default
`guacd.enabled`	Deploy guacd	`true`
`guacd.image.registry`	Image registry	`docker.io`
`guacd.image.repository`	Image repository	`guacamole/guacd`

Parameter	Description	Default
`guacd.image.tag`	Image tag	`latest`
`guacd.image.pullPolicy`	Image pull policy	`IfNotPresent`
`guacd.replicaCount`	Number of replicas	`1`
`guacd.updateStrategy.type`	Deployment update strategy	`RollingUpdate`
`guacd.resources.requests.cpu`	CPU request	`10m`
`guacd.resources.requests.memory`	Memory request	`32Mi`
`guacd.resources.limits.cpu`	CPU limit	`1000m`
`guacd.resources.limits.memory`	Memory limit	`512Mi`
`guacd.livenessProbe.enabled`	Enable liveness probe (TCP)	`true`
`guacd.readinessProbe.enabled`	Enable readiness probe (TCP)	`true`
`guacd.service.type`	Service type	`ClusterIP`
`guacd.service.port`	Service port	`4822`
`guacd.podAnnotations`	Pod annotations	`{}`
`guacd.podSecurityContext`	Pod-level security context	see `values.yaml`
`guacd.securityContext`	Container-level security context	see `values.yaml`
`guacd.nodeSelector`	Node selector	`{}`
`guacd.tolerations`	Tolerations	`[]`
`guacd.affinity`	Affinity rules	`{}`

Ingress

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

The default ingress annotations enable WebSocket support and tune proxy timeouts for agent connections:

```
ingress:
  annotations:
```

```
    nginx.ingress.kubernetes.io/proxy-body-size: "0"
    nginx.ingress.kubernetes.io/proxy-read-timeout: "3600"
    nginx.ingress.kubernetes.io/proxy-send-timeout: "3600"
    nginx.ingress.kubernetes.io/proxy-connect-timeout: "60"
    nginx.ingress.kubernetes.io/proxy-http-version: "1.1"
    nginx.ingress.kubernetes.io/client-body-buffer-size: "4m"
    nginx.ingress.kubernetes.io/websocket-services: "server"
```

Other

Parameter	Description	Default
`serviceAccount.create`	Create a ServiceAccount	`false`
`serviceAccount.annotations`	ServiceAccount annotations	`{}`
`serviceAccount.name`	ServiceAccount name	`""`
`configMap.create`	Create the application ConfigMap	`true`
`configMap.annotations`	ConfigMap annotations	`{}`
`secret.create`	Create the chart-managed Secret (disable when using an external secret)	`true`
`secret.annotations`	Secret annotations	`{}`

Persistent Volumes

The chart creates the following PersistentVolumeClaims:

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

All PVCs respect the `global.storageClass` setting unless overridden at the component level.

Updating PatchMon

Using global.imageTag

The simplest way to update the server image is to set `global.imageTag`:

```
helm upgrade patchmon oci://ghcr.io/ruthlessbeat200/charts/patchmon \
  -n patchmon \
  -f values-prod.yaml \
  --set global.imageTag=2.1.0
```

When `global.imageTag` is set it overrides `server.image.tag`.

Pinning the server tag individually

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

Upgrading the chart version

```
# Upgrade with new values
helm upgrade patchmon oci://ghcr.io/ruthlessbeat200/charts/patchmon \
  --namespace patchmon \
  --values values-prod.yaml

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

Check the releases page for version-specific changes and migration notes.

Uninstalling

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

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

Advanced Configuration

Custom Image Registry

Override the registry for all images (useful for air-gapped environments or private mirrors):

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

This changes every image pull to use the specified registry:

- `registry.example.com/postgres:18-alpine`
- `registry.example.com/redis:8-alpine`
- `registry.example.com/guacamole/guacd:latest`
- `registry.example.com/patchmon/patchmon-server:2.0.0`
- `registry.example.com/busybox:latest` (init containers)

Without `global.imageRegistry`, components use their default registries (`docker.io` for database/Redis/guacd, `ghcr.io` for server).

Multi-Tenant Deployment

Deploy multiple isolated instances in separate namespaces using `fullnameOverride`:

```
fullnameOverride: "patchmon-tenant-a"

server:
  env:
    serverHost: tenant-a.patchmon.example.com
    corsOrigin: https://tenant-a.patchmon.example.com

ingress:
  hosts:
    - host: tenant-a.patchmon.example.com
      paths:
        - path: /
          pathType: Prefix
          service:
```

```
            name: server
            port: 3000
```

Horizontal Pod Autoscaling

```
server:
  autoscaling:
    enabled: true
    minReplicas: 2
    maxReplicas: 10
    targetCPUUtilizationPercentage: 70
    targetMemoryUtilizationPercentage: 80
```

Using an External Database

Disable the built-in database and point the server at an external PostgreSQL instance:

```
database:
  enabled: false
  host: "my-postgres.example.com"
  port: 5432
  auth:
    database: patchmon_db
    username: patchmon_user
    existingSecret: patchmon-secrets
    existingSecretPasswordKey: postgres-password
```

Injecting Extra Environment Variables

Use `server.extraEnv` to pass additional environment variables, for example to trust a custom CA for OIDC:

```
server:
  extraEnv:
    - name: NODE_EXTRA_CA_CERTS
      value: /etc/ssl/certs/my-ca.crt
  extraVolumeMounts:
    - name: my-ca
```

```
      mountPath: /etc/ssl/certs/my-ca.crt
      subPath: ca.crt
      readOnly: true
  extraVolumes:
    - name: my-ca
      configMap:
        name: my-ca-configmap
```

OIDC / SSO Integration

```
server:
  oidc:
    enabled: true
    issuerUrl: "https://auth.example.com/realms/master"
    clientId: "patchmon"
    clientSecret: "your-client-secret"
    scopes: "openid profile email groups"
    buttonText: "Login with SSO"
    autoCreateUsers: true
    syncRoles: true
    groups:
      superadmin: "patchmon-admins"
      admin: ""
      hostManager: ""
      user: ""
      readonly: ""
```

Troubleshooting

Check pod status

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

Check init container logs

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

Check service connectivity

```
# Test database connection
kubectl exec -n patchmon -it statefulset/patchmon-prod-server -- nc -zv patchmon-prod-database 5432

# Test Redis connection
kubectl exec -n patchmon -it statefulset/patchmon-prod-server -- nc -zv patchmon-prod-redis 6379

# Test guacd connection
kubectl exec -n patchmon -it statefulset/patchmon-prod-server -- nc -zv patchmon-prod-guacd 4822

# Check server health
kubectl exec -n patchmon -it statefulset/patchmon-prod-server -- wget -qO- http://localhost:3000/health
```

Common issues

Symptom	Likely cause	Fix
Pods stuck in `Init` state	Database, Redis, or guacd not yet running	Check StatefulSet/Deployment events: `kubectl describe sts -n patchmon`
PVC stuck in `Pending`	No matching StorageClass or no available PV	Verify storage class exists: `kubectl get sc`
`ImagePullBackOff`	Registry credentials missing or incorrect image reference	Check `imagePullSecrets` and image path
Ingress returns 404 / 502	Ingress controller not installed or misconfigured path rules	Verify controller pods and ingress resource: `kubectl describe ingress -n patchmon`
WebSocket disconnects	Missing WebSocket annotations on ingress	Ensure `nginx.ingress.kubernetes.io/websocket-services: "server"` and proxy timeout annotations are set

Symptom	Likely cause	Fix
`secret ... not found`	Required secret was not created before install	Create the secret or set `secret.create: true` with inline passwords

Development

Lint the chart

```
helm lint .
```

Render templates locally

```
# Render with default values
helm template patchmon . --values values-quick-start.yaml

# Render with production values
helm template patchmon . --values values-prod.yaml

# Debug template rendering
helm template patchmon . --values values-quick-start.yaml --debug
```

Dry-run installation

```
helm install patchmon . \
  --namespace patchmon \
  --dry-run --debug \
  --values values-quick-start.yaml
```

Support

- GitHub Issues: github.com/RuTHlessBEat200/PatchMon-helm/issues
- Application repository: github.com/PatchMon/PatchMon

Revision #5
Created 2026-02-14 11:43:51 UTC by RuTHlessBEat200

Updated 2026-04-25 10:11:40 UTC by RuTHlessBEat200