Image not found or type unknown

Cron & Automation

Everything the module does in the background: the deploy pipeline, the task queue with retries and failure tickets, automatic SSL, usage metering & metric billing, and suspend/terminate. All of it is driven by the addon's cron runner.

- The Deploy Process
- Task Queue, Retries & Tickets
- SSL Automation
- Usage Sync & Metric Billing
- Suspend, Unsuspend & Terminate

The Deploy Process

PUQ Web Hosting module **WHMCS**

Order now | Download | Community

Provisioning is **not** done synchronously inside the WHMCS order. `_CreateAccount` simply records the desired service and enqueues the work; the cron-driven task runner then builds the service step by step. This is what keeps orders instant and makes every step retryable.

The chain

A new order fans out into a chain of idempotent tasks, roughly:

```
create package  →  create web user   →  add web domain   →  (PHP/SSL/proxy)
                →  create mail user   →  add mail domain  →  DKIM/SPF/DMARC
                →  create dns user    →  add zone         →  add records (× every DNS
server)
                →  mark service active
```

Each task does exactly one thing on one target, checks whether it's already done before acting, and reports back. Because every step is **idempotent**, re-running the chain (a retry, a Redeploy, a Factory reset) converges on the same end state instead of duplicating resources.

What the customer sees

While the chain runs, the customer sees a live progress splash with a timeline; on success they land on the dashboard, on failure an error screen with **Try again**. This is covered visually in **Deployment & Segmentation → How deployment works**.

Why async

- **Instant orders** — checkout doesn't block on SSH to several servers.
- **Resilience** — a momentarily unreachable node just delays its tasks; the rest proceed and the failed ones retry.
- **Observability** — every step is a row you can watch, retry or inspect in the Task Queue and Operation Log.

> See the next page for the retry/back-off policy and failure tickets.

Task Queue, Retries & Tickets

PUQ Web Hosting module **WHMCS**

Order now | Download | Community

Every action — deploys, client-area changes, status collection, SSL, usage sync — is a row in the task queue, processed by the cron runner.

How a task runs

1. The cron runner picks up due tasks (respecting per-server concurrency limits and the **batch size** per run).
2. Each task is dispatched to its **target server** (critical for the DNS active-active cluster — a task carries its `server_id`).
3. On success the row is marked done; on failure it's scheduled for **retry**.

Retry & back-off

The retry policy lives in **Settings → General**:

- **Max attempts** — how many times a failing task retries.
- **Back-off minutes** — growing delay between attempts, so a flaky node isn't hammered.

A task that exhausts its attempts is marked **error** and stops retrying.

Failure tickets

When **Settings → Notifications → failure ticket** is enabled, a task that fails terminally opens (or

notes on) a WHMCS **support ticket** in the chosen department/priority, so staff are alerted without watching the queue. Subsequent failures on the same operation append a note rather than spamming new tickets.

Watching & cleaning up

The addon **Logs → Task Queue** page shows the live queue with bulk controls (delete success/error/cancelled, clean completed, **force-fail stuck**). **Logs → Operation Log** keeps the persistent history. See **Addon Module → Task Queue & Logs**.

> Stuck tasks (node permanently gone) can be cleared with **Force-fail stuck**; the matching **Verify & Repair** on the service then rebuilds whatever's missing.

SSL Automation

PUQ Web Hosting module **WHMCS**

Order now | Download | Community

SSL is hands-off by default: the module issues and renews **Let's Encrypt** certificates automatically, and gets out of the way when a customer brings their own.

Acquisition

For each role that needs a certificate, an auto-SSL worker:

1. **Checks DNS** — confirms the domain (and `mail.` / `webmail.` for mail) resolves to the right server.
2. **Probes TLS** — checks whether a valid cert is already present.
3. **Issues** — runs the Let's Encrypt request only when the checks pass.

Cadence (Settings → SSL)

The check interval adapts to the situation:

- **Fast mode** — a few quick attempts shortly after provisioning (configurable count + interval), so a new site goes green within minutes of DNS pointing.
- **Normal interval** — the steady-state re-check.
- **Active-cert interval** — the slow re-check once a valid cert exists (renewals).

Rate-limit guard

To respect Let's Encrypt limits, a domain that fails repeatedly is **frozen** for a configurable window (freeze after N fails, for M hours) before trying again. All of these knobs live on **Settings → SSL**.

Custom certificates

When a customer uploads a custom certificate (client **SSL** page), auto-SSL is suspended for that role so the upload isn't overwritten, and a daily **custom-cert expiry** cron warns before it lapses.

> The per-service SSL state, plus manual **Issue** / **Renew now** buttons, are on the admin service panel's **SSL** tab and the client **SSL** page.

Usage Sync & Metric Billing

PUQ Web Hosting module **WHMCS**

Order now | Download | Community

The module keeps live resource figures (disk, bandwidth, mailboxes, databases…) in the local database so both the admin and client areas render instantly, and it feeds WHMCS metric billing.

Per-server batch sync

Usage is collected **per server**, not per service. A `server.syncUsage` worker lists all users on a node in one shot and recomputes every service on it, turning what used to be N × several SSH calls into one batch call per server. The cron schedules these per-server jobs; **Refresh all**, **Force sync** and bulk actions all enqueue the same per-server task.

Statistics page

The addon **Statistics** page reads purely from the database (no SSH at render time): an aggregate summary plus a per-service usage DataTable, with **Refresh all usage** to enqueue a fresh collection.

Metric billing

The module exposes usage to WHMCS's **MetricProvider**, so you can bill on actual consumption (e.g. disk or bandwidth over an included allowance) in addition to flat plan pricing. The figures billed are the same ones shown in the client dashboard gauges.

> A customer can pull fresh numbers on demand with **Sync now** on their dashboard; it enqueues the recompute→sync chain for just their service.

Suspend, Unsuspend & Terminate

PUQ Web Hosting module **WHMCS**

Order now | Download | Community

The WHMCS lifecycle commands map onto the same async, idempotent task model as provisioning.

Suspend / Unsuspend

- **Suspend** — disables the service's users across its servers (web/mail/dns) so sites and mail stop serving, without deleting anything. While suspended, client-area write actions are blocked.
- **Unsuspend** — re-enables them. Because the data was never removed, the service comes straight back.

These follow WHMCS automation (overdue suspend, etc.) or can be triggered manually from the service's Module Commands.

Terminate

Terminate removes the service's resources from every server it touches and cleans up the local rows in a transaction. A pending-terminate state is finalised by cron, and a **force-terminate** path exists for cases where a node is unreachable at termination time so the service can still be closed out cleanly.

The Vanity invariant

On a **Vanity** service these operations are deliberately scoped: they only ever touch that service's **own** subdomain, its **single** mailbox and its **single** DNS record. The shared provider domain, its

mail user and its DNS zone are never modified — that invariant holds through suspend, unsuspend, terminate, redeploy and factory reset alike. See **Vanity Mode → What it is & why**.

> Lifecycle commands always report `success` to WHMCS immediately and do the real work via the queue, mirroring the provisioning model — so a slow or temporarily unreachable node never blocks the WHMCS workflow.