Cron and Automation

How the module's cron-driven state machines work: the deploy pipeline that walks a new VM from creation to ready, the change package pipeline for upgrades and downgrades, the asynchronous terminate pipeline introduced in v3.2, and all other scheduled tasks (snapshot cleanup, backup schedule, stats collection) with their intervals and locking behaviour. Every long-running lifecycle operation runs through cron with live per-step output, so clients never wait on synchronous HTTP requests and admins see real-time progress in the logs.

Deploy Process

Proxmox KVM module **WHMCS**

Order now | Download | FAQ

Overview

When a new virtual machine is provisioned (from a WHMCS order, an admin **Create** action, or the WHMCS API), the module does **not** try to do everything inside the HTTP request. Instead it just writes the VM record with status `creation` and returns immediately — WHMCS sees the order as "accepted" within milliseconds. The actual work is done by the cron task **Process VMs** as a 15-step state machine. Each step is idempotent and resumable: if anything fails, the VM stays at its current step and the next cron tick retries it.

This design makes the module resilient to:

- Proxmox API timeouts and slow storage operations
- Cluster-wide task queue back-pressure
- Transient network failures between the WHMCS host and Proxmox
- Long-running operations (disk resize, full clones, cross-node migration)

Deploy Pipeline

The pipeline progresses through the following states:

```
creation → set_ip → clone → set_dns → migrated →
set_cpu_ram → set_system_disk_size → set_system_disk_bandwidth →
set_created_additional_disk → set_additional_disk_size → set_additional_disk_bandwidth →
set_network → set_firewall → set_cloudinit → starting → ready
```

A state name represents the **last completed step** — not the current action. When a VM is in state `set_ip`, the IP has been allocated and the next action to run is "Clone VM".

Step descriptions

State (done)	Next action	What happens
`creation`	Allocate IP	Choose an IP pool matching the server/bridge/VLAN, reserve IPv4 and/or IPv6 addresses, write them into `tblhosting` and the VM record.
`set_ip`	Clone VM	Clone the template to the target storage. Supports both linked and full clones depending on product config.
`clone`	Configure DNS records	Create forward A/AAAA for the VM's FQDN and PTR records for all assigned IPs. Runs against every matching DNS zone (see DNS Zones & Integration). **DNS errors never block deployment** — they are logged and the pipeline moves on.
`set_dns`	Migrate to target node	If the template lives on a different Proxmox node than the target, do an offline migration with `targetstorage` mapping. Skipped when source and target nodes are the same.
`migrated`	Set CPU & RAM	Apply the cores / sockets / memory from the product configuration.
`set_cpu_ram`	Resize system disk	Expand the system disk to the configured size.
`set_system_disk_size`	Set system disk I/O	Apply `iops_rd` / `iops_wr` / `mbps_rd` / `mbps_wr` bandwidth limits.
`set_system_disk_bandwidth`	Create additional disk	If the product has an additional disk, create it on the configured storage.
`set_created_additional_disk`	Resize additional disk	Expand it to the configured size.
`set_additional_disk_size`	Set additional disk I/O	Apply bandwidth limits to the additional disk.
`set_additional_disk_bandwidth`	Configure network	Set bridge, VLAN, rate limit, and enable the firewall flag on the NIC.
`set_network`	Configure firewall	Apply per-product firewall options (enable, DHCP/NDP, MAC filter, IP filter, log levels), policies, and anti-spoofing IPSet with the allocated IPs.

State (done)	Next action	What happens
`set_firewall`	Configure cloud-init	Push hostname, IP addresses, gateway, DNS servers, username, password and SSH keys into thecloud-init drive.
`set_cloudinit`	Start VM	Power on the VM through the Proxmox API.
`starting`	Verify running + email	Wait up to 5 cron ticks for the guest to report `running`. Send the "VM is ready" email.
`ready`	—	Final state. Client has full access to the service.

Resumability & retry

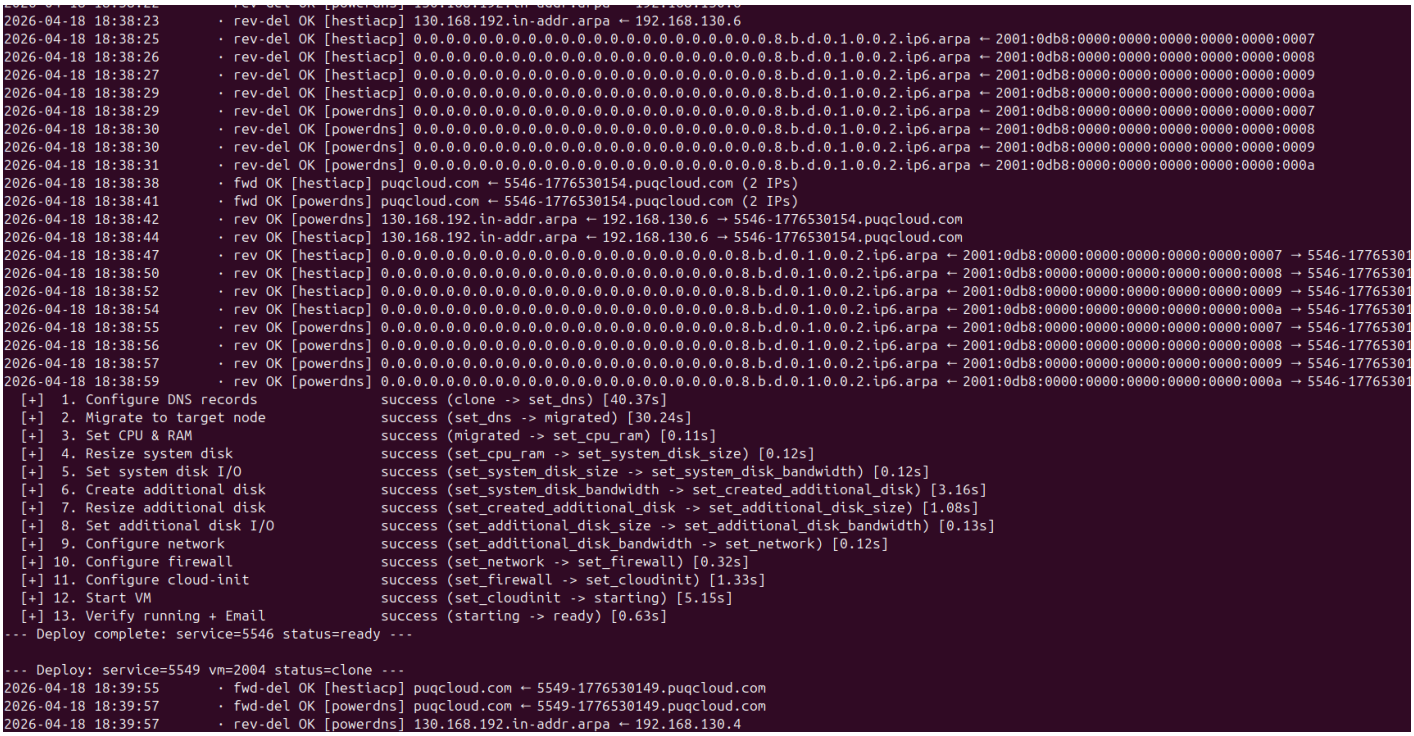
Every step is wrapped in this contract:

1. **Check current state** — guards against double execution if the cron fires twice.
2. **Do one API call (or a short sequence of related calls)** — deliberately small so the step either completes quickly or can be retried cheaply.
3. **On success** — advance `vm_status` to the next state.
4. **On failure** — return the error string; `vm_status` stays the same; the cron log records the step with the error; next tick retries.

There is **no retry count limit**. A VM that genuinely cannot deploy (misconfigured IP pool, no free node with matching storage) will stay stuck at its current step and keep appearing in the cron log — visible immediately. Admins are expected to fix the root cause (add IPs to the pool, adjust storage mapping) rather than fight a phantom max-retry counter.

DNS configuration during deploy

The `clone → set_dns` transition is where forward and reverse DNS records are registered. With many IPs or many DNS zones, this can involve dozens of API calls. Starting with v3.2 each DNS operation is logged live to the cron output, so admins can watch records being created in real time:



Lines prefixed with `· fwd OK` are forward (A/AAAA) creations, `· rev OK` are reverse (PTR) creations. Per-zone failures (for example, one DNS provider is temporarily unreachable) are logged as `· fwd ERR` / `· rev ERR` but do **not** abort deployment — other zones and subsequent steps still run.

Full deploy walkthrough

A successful deploy looks like this in the standalone cron output (`php cron.php`). Each step shows its duration and the state transition:

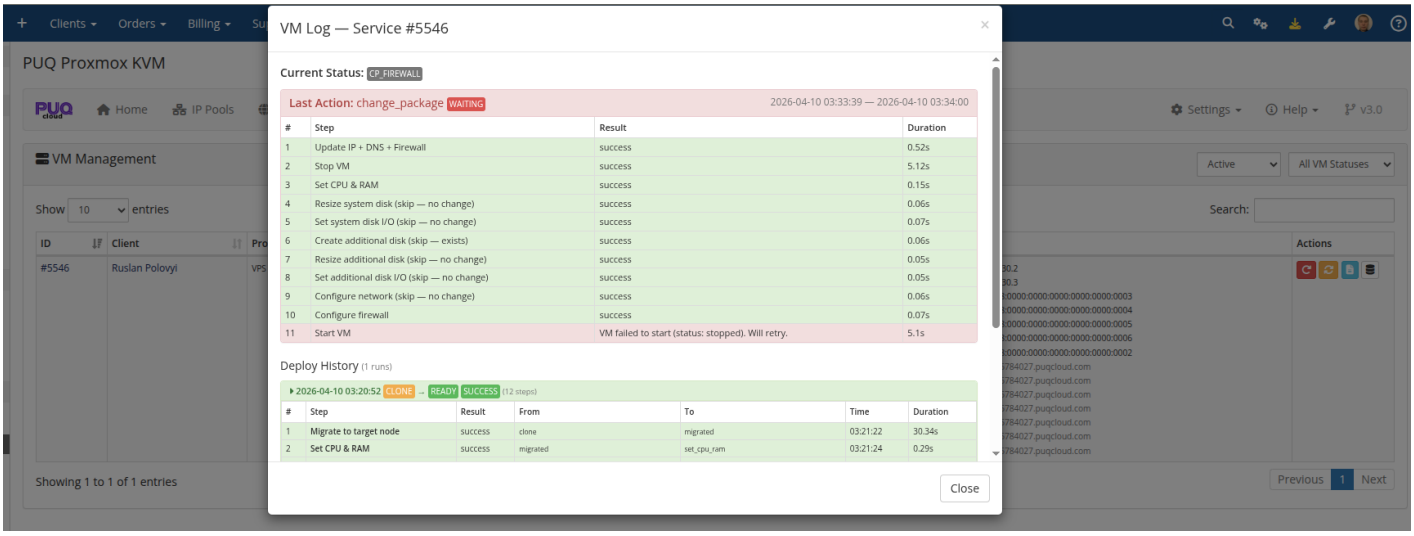
```
2026-04-18 18:38:18 PUQ Proxmox KVM Cron Start
2026-04-18 18:38:18 [processVirtualMachines] Running (interval: 1m, last: 16:04:52)

--- Deploy: service=5546 vm=2005 status=clone ---
2026-04-18 18:38:20       · fwd-del OK [hestiacp] puqcloud.com ← 5546-1776530154.puqcloud.com
2026-04-18 18:38:21       · fwd-del OK [powerdns] puqcloud.com ← 5546-1776530154.puqcloud.com
2026-04-18 18:38:22       · rev-del OK [powerdns] 130.168.192.in-addr.arpa ← 192.168.130.6
2026-04-18 18:38:23       · rev-del OK [hestiacp] 130.168.192.in-addr.arpa ← 192.168.130.6
2026-04-18 18:38:25       · rev-del OK [hestiacp] 0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.8.b.d.0.1.0.0.2.ip6.arpa ← 2001:0db8:0000:0000:0000:0000:0000:0007
2026-04-18 18:38:26       · rev-del OK [hestiacp] 0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.8.b.d.0.1.0.0.2.ip6.arpa ← 2001:0db8:0000:0000:0000:0000:0000:0008
2026-04-18 18:38:27       · rev-del OK [hestiacp] 0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.8.b.d.0.1.0.0.2.ip6.arpa ← 2001:0db8:0000:0000:0000:0000:0000:0009
2026-04-18 18:38:29       · rev-del OK [hestiacp] 0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.8.b.d.0.1.0.0.2.ip6.arpa ← 2001:0db8:0000:0000:0000:0000:0000:000a
2026-04-18 18:38:29       · rev-del OK [powerdns] 0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.8.b.d.0.1.0.0.2.ip6.arpa ← 2001:0db8:0000:0000:0000:0000:0000:0007
2026-04-18 18:38:30       · rev-del OK [powerdns] 0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.8.b.d.0.1.0.0.2.ip6.arpa ← 2001:0db8:0000:0000:0000:0000:0000:0008
2026-04-18 18:38:30       · rev-del OK [powerdns] 0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.8.b.d.0.1.0.0.2.ip6.arpa ← 2001:0db8:0000:0000:0000:0000:0000:0009
2026-04-18 18:38:31       · rev-del OK [powerdns] 0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.8.b.d.0.1.0.0.2.ip6.arpa ← 2001:0db8:0000:0000:0000:0000:0000:000a
2026-04-18 18:38:38       · fwd OK [hestiacp] puqcloud.com ← 5546-1776530154.puqcloud.com (2 IPs)
2026-04-18 18:38:41       · fwd OK [powerdns] puqcloud.com ← 5546-1776530154.puqcloud.com (2 IPs)
2026-04-18 18:38:42       · rev OK [powerdns] 130.168.192.in-addr.arpa ← 192.168.130.6 → 5546-1776530154.puqcloud.com
2026-04-18 18:38:44       · rev OK [hestiacp] 130.168.192.in-addr.arpa ← 192.168.130.6 → 5546-1776530154.puqcloud.com
2026-04-18 18:38:47       · rev OK [hestiacp] 0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.8.b.d.0.1.0.0.2.ip6.arpa ← 2001:0db8:0000:0000:0000:0000:0000:0007 → 5546-1776530154.puqcloud.co
2026-04-18 18:38:50       · rev OK [hestiacp] 0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.8.b.d.0.1.0.0.2.ip6.arpa ← 2001:0db8:0000:0000:0000:0000:0000:0008 → 5546-1776530154.puqcloud.co
2026-04-18 18:38:52       · rev OK [hestiacp] 0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.8.b.d.0.1.0.0.2.ip6.arpa ← 2001:0db8:0000:0000:0000:0000:0000:0009 → 5546-1776530154.puqcloud.co
2026-04-18 18:38:54       · rev OK [hestiacp] 0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.8.b.d.0.1.0.0.2.ip6.arpa ← 2001:0db8:0000:0000:0000:0000:0000:000a → 5546-1776530154.puqcloud.co
2026-04-18 18:38:55       · rev OK [powerdns] 0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.8.b.d.0.1.0.0.2.ip6.arpa ← 2001:0db8:0000:0000:0000:0000:0000:0007 → 5546-1776530154.puqcloud.co
2026-04-18 18:38:56       · rev OK [powerdns] 0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.8.b.d.0.1.0.0.2.ip6.arpa ← 2001:0db8:0000:0000:0000:0000:0000:0008 → 5546-1776530154.puqcloud.co
2026-04-18 18:38:57       · rev OK [powerdns] 0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.8.b.d.0.1.0.0.2.ip6.arpa ← 2001:0db8:0000:0000:0000:0000:0000:0009 → 5546-1776530154.puqcloud.co
2026-04-18 18:38:59       · rev OK [powerdns] 0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.8.b.d.0.1.0.0.2.ip6.arpa ← 2001:0db8:0000:0000:0000:0000:0000:000a → 5546-1776530154.puqcloud.co
  [+]  1. Configure DNS records              success (clone -> set_dns) [40.37s]
  [+]  2. Migrate to target node             success (set_dns -> migrated) [30.24s]
  [+]  3. Set CPU & RAM                      success (migrated -> set_cpu_ram) [0.11s]
  [+]  4. Resize system disk                 success (set_cpu_ram -> set_system_disk_size) [0.12s]
  [+]  5. Set system disk I/O                success (set_system_disk_size -> set_system_disk_bandwidth) [0.12s]
  [+]  6. Create additional disk             success (set_system_disk_bandwidth -> set_created_additional_disk) [3.16s]
  [+]  7. Resize additional disk             success (set_created_additional_disk -> set_additional_disk_size) [1.08s]
  [+]  8. Set additional disk I/O            success (set_additional_disk_size -> set_additional_disk_bandwidth) [0.13s]
  [+]  9. Configure network                  success (set_additional_disk_bandwidth -> set_network) [0.12s]
  [+] 10. Configure firewall                 success (set_network -> set_firewall) [0.32s]
  [+] 11. Configure cloud-init               success (set_firewall -> set_cloudinit) [1.33s]
  [+] 12. Start VM                           success (set_cloudinit -> starting) [5.15s]
  [+] 13. Verify running + Email             success (starting -> ready) [0.63s]
--- Deploy complete: service=5546 status=ready ---
```

Every line is flushed to stdout immediately — nothing is buffered until the end. This means even during a long-running step (full clone of a large template, cross-node migration) you can tell whether the job is still making progress or has truly stalled.

Deploy log in the admin UI

In the addon's **VM Management → Log** modal, every run is recorded with per-step duration, state transitions, and any errors:



The most recent 50 runs are kept per VM. If the last attempt paused partway, the log is marked `waiting` with an `error` field showing why — useful for diagnosing a sticky step.

Migration step in detail

`set_dns → migrated` handles the common Proxmox cluster topology where templates live on one node (often for storage cost reasons) but client VMs should run on another:

1. `set_ip → clone` creates the VM on **Template Node A** using the template's local storage.
2. `clone → set_dns` configures DNS while the VM is still on Node A.
3. `set_dns → migrated` migrates the VM from Node A to the **Target Node B** with a `targetstorage` remap so disks end up on the right backend on the target.
4. All subsequent steps (CPU, disk, network, firewall, cloud-init, start) run against the VM on Node B.

Target node selection considers storage availability and free RAM on candidate nodes. If no suitable target is found, the VM stays on Node A and deployment completes there — the VM is still fully functional, just not on the preferred node.

What triggers a deploy

A deploy starts the moment `vm_status` is set to `creation`. That happens when:

- A client's order is provisioned (automatic after payment, or manual accept/provision by an admin).
- An admin clicks **Create** or **Module Create** in the service's module commands.
- `ModuleCreateAccount` is called through the WHMCS API.
- A **Redeploy** action resets the VM record (deletes the existing VM on Proxmox, clears logs, sets status to `creation`).

From the moment `creation` is written, the next cron tick picks up the VM. At the default 1-minute **Process VMs** interval, that means provisioning starts within 60 seconds of the trigger.

Related reading

- Change Package — how upgrades and downgrades use a similar state machine.
- Terminate Process — how services are torn down asynchronously.
- DNS Zones & Integration — configuring the providers that the `clone → set_dns` stepwrites to.
- Scheduled Tasks — all cron tasks including Process VMs.

Change Package

Proxmox KVM module **WHMCS**

Order now | Download | FAQ

Overview

A package change (upgrade or downgrade) reconfigures a live VM to match a different product plan — new CPU/RAM values, larger disks, different network settings, changed firewall options. Just like Deploy, this runs asynchronously as a resumable state machine driven by the cron. The client does not wait for all the Proxmox API calls to finish — WHMCS accepts the upgrade order immediately and the module walks the VM through the change over the next one or two cron ticks.

Change Package Pipeline

```
change_package → cp_update_ip → cp_stop → cp_cpu_ram →
cp_system_disk_size → cp_system_disk_bandwidth →
cp_additional_disk → cp_additional_disk_size → cp_additional_disk_bandwidth →
cp_network → cp_firewall → cp_start → ready
```

Step descriptions

State (done)	Next action	What happens
`change_package`	Update IP + DNS + firewall	Reload config, reload remote VM data, update IP allocation if the new package changes IP requirements, refresh anti-spoofing IPSet, and refresh forward + reverse DNS records to reflect any IP changes.

State (done)	Next action	What happens
`cp_update_ip`	Stop VM	Stop the VM. Required because some downstream steps (disk resize, CPU/RAM limits) cannot be applied to a running VM. Polls until the VM reports `stopped`.
`cp_stop`	Set CPU & RAM	Apply new CPU cores / sockets / memory from the new package. Skipped if the values are unchanged.
`cp_cpu_ram`	Resize system disk	Grow the system disk to the new size (Proxmox cannot shrink disks — a smaller target is silently skipped).
`cp_system_disk_size`	System disk I/O	Apply new bandwidth limits to the system disk.
`cp_system_disk_bandwidth`	Additional disk	Create an additional disk if the new package includes one and the VM does not have it yet.
`cp_additional_disk`	Additional disk size	Grow the additional disk.
`cp_additional_disk_size`	Additional disk I/O	Apply new bandwidth limits.
`cp_additional_disk_bandwidth`	Network	Update bridge, VLAN and NIC rate limit.
`cp_network`	Firewall	Re-apply per-product firewall options and refresh the anti-spoofing IPSet with the current IP list.
`cp_firewall`	Start VM	Power the VM back on.
`cp_start`	Verify running	Wait up to 5 ticks for the guest to report `running`.
`ready`	—	Done. VM is live with the new package.

Skip-if-unchanged optimization

Every `cp_*` step first compares the **current** VM configuration with the **target** package configuration. If they match, the step logs `skip (no change)` and advances immediately. A downgrade that only reduces RAM, for example, doesn't touch the disks, network, or firewall — it stops the VM, applies RAM, restarts. In practice most package changes complete in 20-40 seconds real time.

In the log this shows up as lines like `cp_system_disk_size skip → cp_system_disk_bandwidth`. Useful for auditing what actually changed during a given upgrade.

DNS refresh during change package

The first step (`change_package → cp_update_ip`) triggers a full DNS sync — old records for removed IPs are deleted, new records for added IPs are created. All matching DNS zones are updated. With v3.2 each operation is logged live so admins can watch the refresh happen:

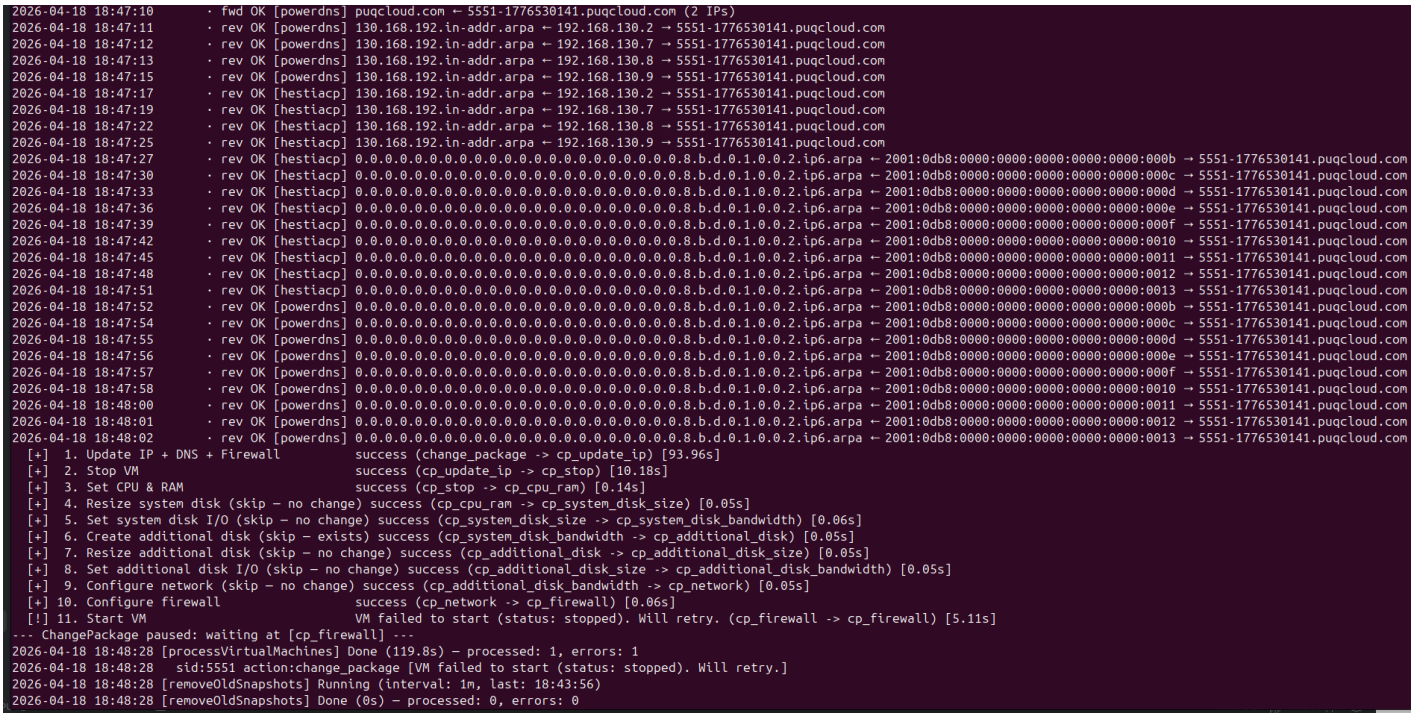
```
2026-04-18 18:46:28 PUQ Proxmox KVM Cron Start
2026-04-18 18:46:28 [processVirtualMachines] Running (interval: 1m, last: 18:43:56)

--- ChangePackage: service=5551 vm=2002 status=change_package ---
2026-04-18 18:46:33       · fwd-del OK [hestiacp] puqcloud.com ← 5551-1776530141.puqcloud.com
2026-04-18 18:46:34       · fwd-del OK [powerdns] puqcloud.com ← 5551-1776530141.puqcloud.com
2026-04-18 18:46:35       · rev-del OK [powerdns] 130.168.192.in-addr.arpa ← 192.168.130.2
2026-04-18 18:46:36       · rev-del OK [powerdns] 130.168.192.in-addr.arpa ← 192.168.130.7
2026-04-18 18:46:36       · rev-del OK [powerdns] 130.168.192.in-addr.arpa ← 192.168.130.8
2026-04-18 18:46:37       · rev-del OK [powerdns] 130.168.192.in-addr.arpa ← 192.168.130.9
2026-04-18 18:46:41       · rev-del OK [hestiacp] 130.168.192.in-addr.arpa ← 192.168.130.2
2026-04-18 18:46:42       · rev-del OK [hestiacp] 130.168.192.in-addr.arpa ← 192.168.130.7
2026-04-18 18:46:43       · rev-del OK [hestiacp] 130.168.192.in-addr.arpa ← 192.168.130.8
2026-04-18 18:46:44       · rev-del OK [hestiacp] 130.168.192.in-addr.arpa ← 192.168.130.9
2026-04-18 18:46:46       · rev-del OK [hestiacp] 0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.8.b.d.0.1.0.0.2.ip6.arpa ← 2001:0db8:0000:0000:0000:0000:0000:000b
2026-04-18 18:46:47       · rev-del OK [hestiacp] 0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.8.b.d.0.1.0.0.2.ip6.arpa ← 2001:0db8:0000:0000:0000:0000:0000:000c
2026-04-18 18:46:49       · rev-del OK [hestiacp] 0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.8.b.d.0.1.0.0.2.ip6.arpa ← 2001:0db8:0000:0000:0000:0000:0000:000d
2026-04-18 18:46:50       · rev-del OK [hestiacp] 0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.8.b.d.0.1.0.0.2.ip6.arpa ← 2001:0db8:0000:0000:0000:0000:0000:000e
2026-04-18 18:46:52       · rev-del OK [hestiacp] 0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.8.b.d.0.1.0.0.2.ip6.arpa ← 2001:0db8:0000:0000:0000:0000:0000:000f
2026-04-18 18:46:53       · rev-del OK [hestiacp] 0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.8.b.d.0.1.0.0.2.ip6.arpa ← 2001:0db8:0000:0000:0000:0000:0000:0010
2026-04-18 18:46:55       · rev-del OK [hestiacp] 0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.8.b.d.0.1.0.0.2.ip6.arpa ← 2001:0db8:0000:0000:0000:0000:0000:0011
2026-04-18 18:46:56       · rev-del OK [hestiacp] 0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.8.b.d.0.1.0.0.2.ip6.arpa ← 2001:0db8:0000:0000:0000:0000:0000:0012
2026-04-18 18:46:57       · rev-del OK [hestiacp] 0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.8.b.d.0.1.0.0.2.ip6.arpa ← 2001:0db8:0000:0000:0000:0000:0000:0013
2026-04-18 18:46:58       · rev-del OK [powerdns] 0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.8.b.d.0.1.0.0.2.ip6.arpa ← 2001:0db8:0000:0000:0000:0000:0000:000b
2026-04-18 18:46:59       · rev-del OK [powerdns] 0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.8.b.d.0.1.0.0.2.ip6.arpa ← 2001:0db8:0000:0000:0000:0000:0000:000c
2026-04-18 18:46:59       · rev-del OK [powerdns] 0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.8.b.d.0.1.0.0.2.ip6.arpa ← 2001:0db8:0000:0000:0000:0000:0000:000d
2026-04-18 18:47:00       · rev-del OK [powerdns] 0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.8.b.d.0.1.0.0.2.ip6.arpa ← 2001:0db8:0000:0000:0000:0000:0000:000e
2026-04-18 18:47:00       · rev-del OK [powerdns] 0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.8.b.d.0.1.0.0.2.ip6.arpa ← 2001:0db8:0000:0000:0000:0000:0000:000f
2026-04-18 18:47:01       · rev-del OK [powerdns] 0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.8.b.d.0.1.0.0.2.ip6.arpa ← 2001:0db8:0000:0000:0000:0000:0000:0010
2026-04-18 18:47:01       · rev-del OK [powerdns] 0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.8.b.d.0.1.0.0.2.ip6.arpa ← 2001:0db8:0000:0000:0000:0000:0000:0011
2026-04-18 18:47:02       · rev-del OK [powerdns] 0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.8.b.d.0.1.0.0.2.ip6.arpa ← 2001:0db8:0000:0000:0000:0000:0000:0012
2026-04-18 18:47:03       · rev-del OK [powerdns] 0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.8.b.d.0.1.0.0.2.ip6.arpa ← 2001:0db8:0000:0000:0000:0000:0000:0013
2026-04-18 18:47:08       · fwd OK [hestiacp] puqcloud.com ← 5551-1776530141.puqcloud.com (2 IPs)
2026-04-18 18:47:10       · fwd OK [powerdns] puqcloud.com ← 5551-1776530141.puqcloud.com (2 IPs)
2026-04-18 18:47:11       · rev OK [powerdns] 130.168.192.in-addr.arpa ← 192.168.130.2 → 5551-1776530141.puqcloud.com
2026-04-18 18:47:12       · rev OK [powerdns] 130.168.192.in-addr.arpa ← 192.168.130.7 → 5551-1776530141.puqcloud.com
2026-04-18 18:47:13       · rev OK [powerdns] 130.168.192.in-addr.arpa ← 192.168.130.8 → 5551-1776530141.puqcloud.com
2026-04-18 18:47:15       · rev OK [powerdns] 130.168.192.in-addr.arpa ← 192.168.130.9 → 5551-1776530141.puqcloud.com
2026-04-18 18:47:17       · rev OK [hestiacp] 130.168.192.in-addr.arpa ← 192.168.130.2 → 5551-1776530141.puqcloud.com
2026-04-18 18:47:19       · rev OK [hestiacp] 130.168.192.in-addr.arpa ← 192.168.130.7 → 5551-1776530141.puqcloud.com
```

`fwd-del OK` / `rev-del OK` lines are the old records being removed; `fwd OK` / `rev OK` lines below are the new records being created. Failures in one provider do not block the rest — DNS errors are non-blocking exactly like they are during deploy.

Full change package walkthrough

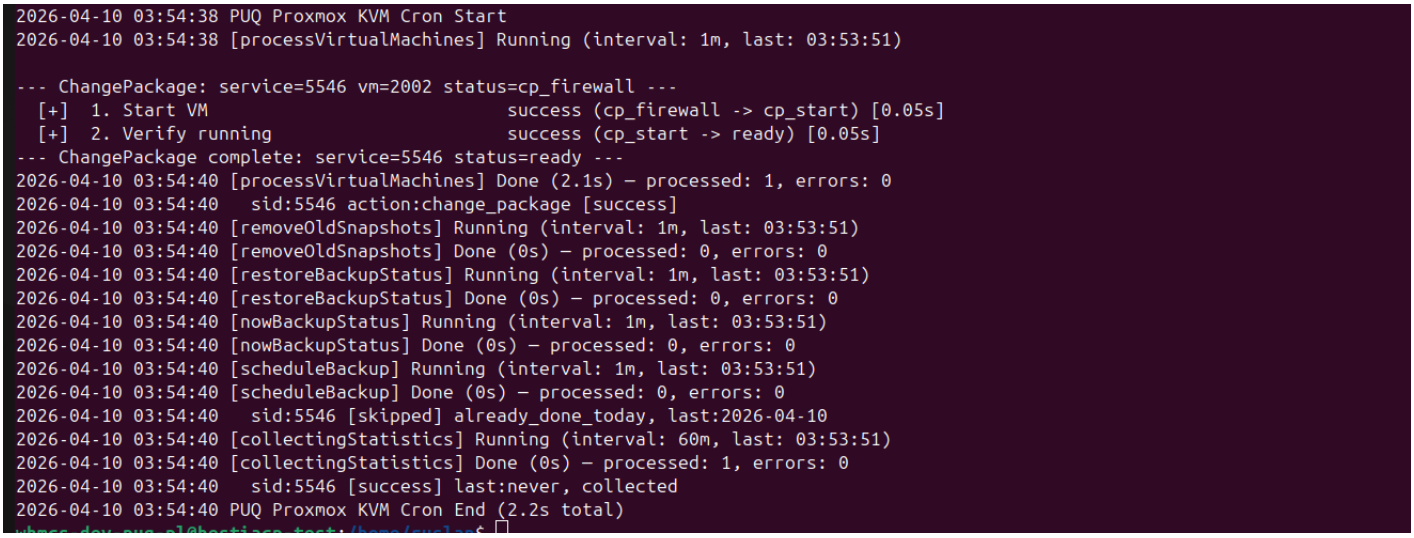
A complete upgrade in the cron output, including a step that failed to start the VM on the first attempt and was retried on the next tick:



Successful steps show `success`, skipped steps show `skip (no change)`, and the failed `cp_start` step gets retried until it succeeds. At no point is the state machine forced to restart from the beginning — retry only re-runs the step that didn't complete.

Log viewing

Every change package run writes a structured entry to `vm_last_action_log` on the VM record. In the addon's VM Management the **Log** modal shows each step with its duration, result, and any skip markers:



If the last run had any failures the modal shows a red error banner at the top with the failure reason.

Retry semantics

Change package uses the same "no retry limit, no time bomb" design as deploy:

- A failed step keeps the VM in its current `cp_*` state.
- The next cron tick retries **only that step**, not the whole pipeline.
- Earlier successful steps are never repeated — disk resizes, for example, are not redone on retry.
- A persistent failure is visible in every cron log entry until an admin addresses the root cause.

During the `cp_stop` → `cp_start` window the VM is offline for as long as the hardware changes take. For most upgrades this is under a minute. For large disk resizes it can be longer — Proxmox needs to finish the storage operation before `cp_start` can proceed.

What triggers a change package

A package change starts when `vm_status` is set to `change_package`. That happens when:

- A client completes an upgrade/downgrade order in the client area.
- An admin clicks **Change Package** in the service module commands.
- `ChangePackage` is called through the WHMCS API.

The module verifies the current `vm_status` is either `ready` (normal path) or already `change_package` (idempotent) before setting state. An in-progress deploy or terminate will be respected — the change package request waits until the VM returns to `ready`.

Caveats

- **Disks cannot be shrunk.** Proxmox does not support shrinking virtual disks safely. A downgrade to a smaller disk size logs "skip (new size is smaller)" and keeps the existing larger disk. Billing is unaffected — WHMCS tracks the package, not the disk size on disk.
- **VM must be healthy to stop gracefully.** If the guest OS is unresponsive, `cp_stop` may take longer and eventually force-stop.
- **Cross-node migration during upgrade is not performed.** The VM stays on its current node. If you need to move a VM to a different node during an upgrade, do the

migration separately in Proxmox first.

Related reading

- Deploy Process — first-time provisioning using the same state-machine pattern.
- Terminate Process — async service teardown.
- DNS Zones & Integration — how the DNS refresh during `change_package → cp_update_ip` works.

Terminate Process

Proxmox KVM module **WHMCS**

Order now | Download | FAQ

Overview

Terminating a service means destroying the virtual machine on Proxmox, removing its backups, deleting its DNS records across every configured provider, and cleaning up the WHMCS records. On a service with many backups or many DNS entries this easily takes over a minute — more than a typical PHP request limit allows.

Starting with v3.2 terminate runs asynchronously. When an admin clicks **Terminate**, the module:

1. Sends a fire-and-forget **stop** request to Proxmox so the VM starts shutting down right away.
2. Sets `vm_status = 'terminate'` on the VM record.
3. Returns `success` to WHMCS.

WHMCS then marks the service **Terminated** immediately — the client loses client-area access within the same request. The heavy work (polling for stop, removing backups, deleting DNS records, the Proxmox DELETE call, clearing `tblhosting` and the VM record) is done by the cron task **Process VMs** on the next tick.

Terminate Pipeline

The pipeline is a single cron handler, not a multi-step state machine — but each internal phase is logged as a distinct event.

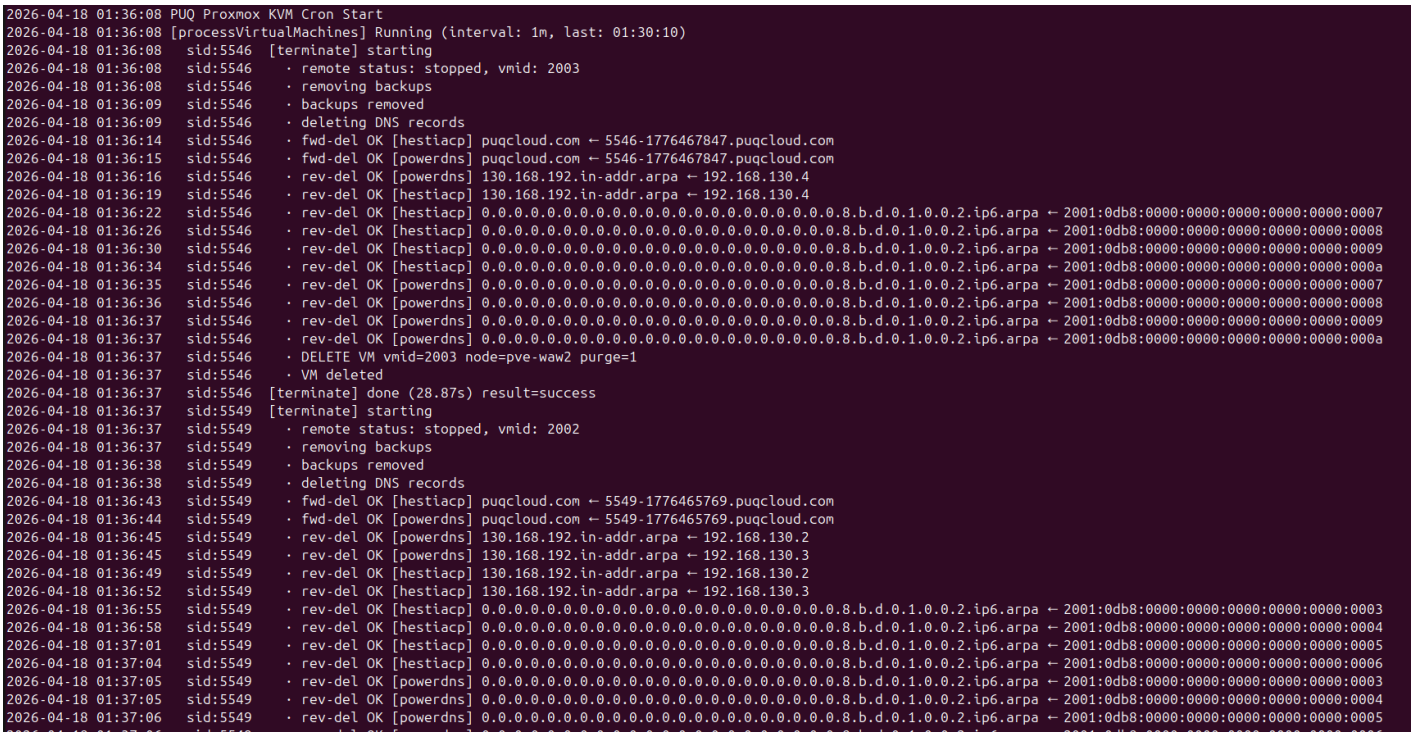
```
terminate → [stop VM] → [remove backups] → [delete DNS] → [DELETE VM] → [clean DB] → remove
                                                                  └─ on error → error_terminate
```

Phases

Phase	What happens	Failure handling
Stop VM	Single stop request, then poll the remote status every 5 seconds for up to 120 seconds (graceful). If still `running`, send a force-stop and poll another 60 seconds.	If the VM is still running after both windows, proceed to DELETE anyway — `purge=1` can reap a hung VM.
Remove backups	Best-effort delete of every backup snapshot for the VM across all configured storages.	Backup deletion errors are caught, logged, ignored.
Delete DNS	For every DNS zone whose name matches the VM's domain or an assigned IP, remove the forward A/AAAA and reverse PTR records.	Per-zone, per-IP errors are non-blocking — caught, logged, the next record continues.
DELETE VM	The Proxmox `DELETE /nodes/<node>/qemu/<vmid>?purge=1` call. **This is the only phase that can cause failure** — everything else is best-effort.	On error the VM goes to `error_terminate`. The DB is **not** cleaned.
Clean DB	Only on DELETE success. Wipes `tblhosting.dedicatedip/assignedips/domain` and clears identity fields on the VM record.	—

Live cron output

Every phase is streamed to the cron output with timestamps and progress heartbeats. A completed terminate looks like this:



Lines you will see:

- `[terminate] starting` — pipeline began for this service.
- `remote status: running, vmid: 2003` — initial state.
- `stop request sent` — the graceful stop was issued.
- `still running, waited 15s / 120s, status: running` — heartbeat while waiting. Prints every 15 seconds.
- `stopped after ~45s` — the guest acknowledged shutdown.
- `removing backups` / `backups removed` — backup cleanup.
- `deleting DNS records` — beginning of the DNS phase.
- `rev-del OK [hestiacp] 130.168.192.in-addr.arpa ← 192.168.130.6` — individual reverse record deletion.
- `rev-del OK [powerdns] 0.0.0.0.…8.b.d.0.1.0.0.2.ip6.arpa ← …` — same, IPv6.
- `fwd-del OK [hestiacp] puqcloud.com ← 5546-1776530141.puqcloud.com` — forward record deletion.
- `DELETE VM vmid=2003 node=pve-wew2 purge=1` — the Proxmox destroy call.
- `VM deleted` — success.
- `[terminate] done (28.67s) result=success` — pipeline finished.

If the cron runs in `--verbose` mode (standalone `php cron.php` does by default) everything is flushed line-by-line in real time.

What happens on failure

If the **DELETE VM** API call returns an error (node unreachable, lock conflict, auth expired, etc.),the cron handler switches to the failure path:

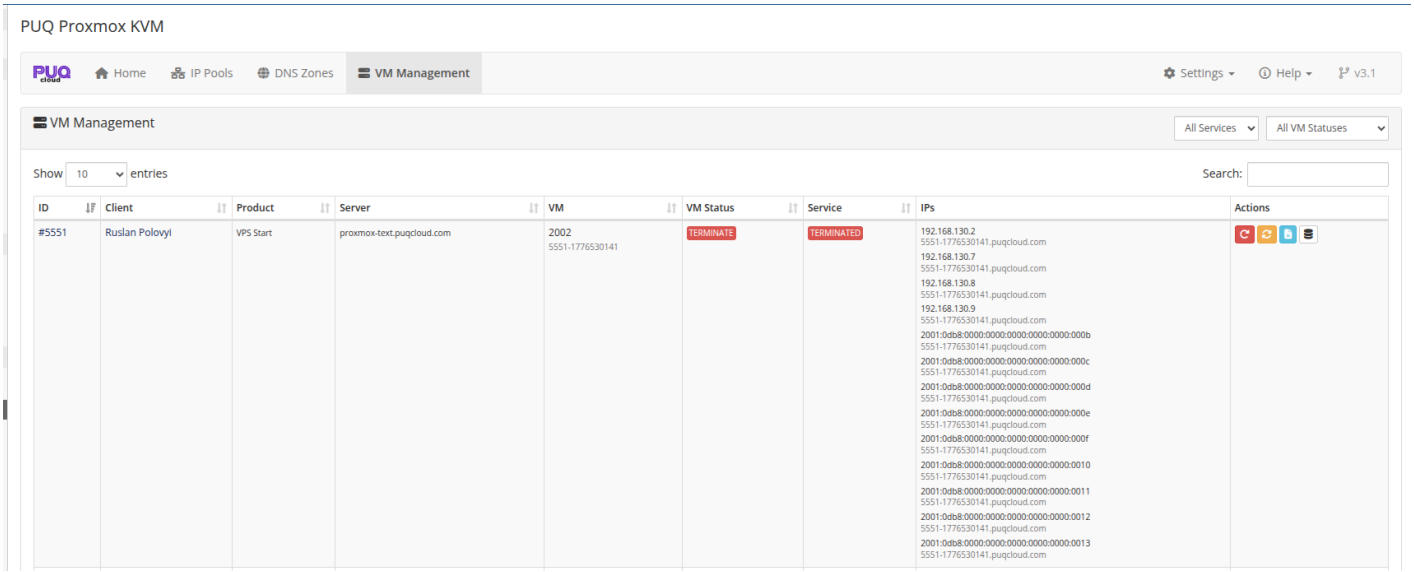
- `vm_status` is set to **`error_terminate`**.
- The VM record is **not** cleaned. `tblhosting.dedicatedip` / `assignedips` stay populated, the VM ID stays on the record, the domain is preserved.
- The client gets **one** entry in the Activity Log: `Service termination FAILED — admin attention required. Error: <reason>`.
- The VM Log modal in VM Management shows a red banner with the error.
- The cron will **not** automatically retry — `error_*` states are admin-manual.

Why IPs stay allocated on failure

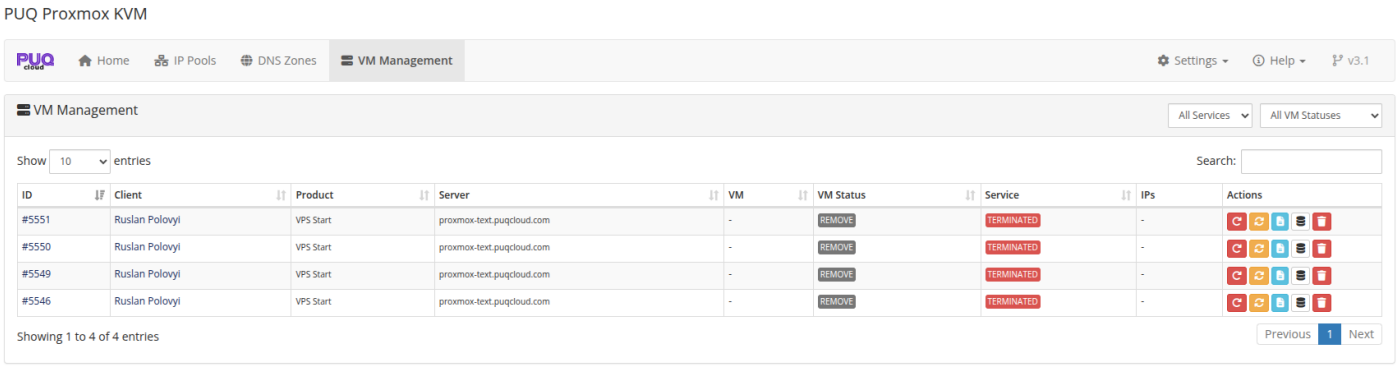
It is deliberate. If the VM still exists on Proxmox but the WHMCS record has been cleared, those IPs are free to be reassigned — and the IP pool will hand them out to the next client. That new client's VM will then conflict with a "zombie" VM still holding the IPs on Proxmox. Keeping the record intact until Proxmox confirms the VM is gone avoids this class of bug entirely.

Admin actions after `error_terminate`

Open **Addons → PUQ Proxmox KVM → VM Management**. Rows in `error_terminate` show a red status badge and a trash icon in the Actions column:



After the cron finishes:



Reset VM Status modal

Clicking the Reset button opens a modal with a full reference of available target statuses and when to use each:

- `terminate` — re-queue the termination. Use this after you've fixed whatever made the original attempt fail (restored node connectivity, re-authed with Proxmox, etc.).
- `remove` — force-mark the VM record as removed. **Does not touch Proxmox.** Use this only when you've manually deleted the VM from Proxmox and just want WHMCS to stop showing it.
- `ready`, `creation`, `set_ip`, `change_package`, `set_dns_records` — retry other state machines (see the Deploy and Change Package docs).

Delete Record button

Visible **only** for rows in `error_terminate` or `remove`. Removes the VM row from `puqProxmoxKVM_vm_info`. Does **not** touch Proxmox or `tblhosting`. Use this when the VM is long gone from Proxmox but you want to clean up leftover database rows. The confirmation dialog repeats this warning explicitly.

Guarantees

- **Client access revoked instantly.** The service is Terminated in WHMCS the same moment the admin clicks the button. The client cannot log back in while the actual teardown happens in the background.
- **IPs cannot be reassigned before the VM is gone from Proxmox.** A failing terminate preserves the allocation until a human confirms the cleanup.
- **One Activity Log entry per attempt.** Success → one "terminated successfully" entry.

Failure → one "termination FAILED" entry. Cron never writes duplicates on skipped `error_terminate` rows.

- **DNS errors never block termination.** A missing or broken DNS provider does not stop the VM from being destroyed.

Logs

- **Per-VM action log** — in the addon's VM Management → Log modal, every terminate attempt (successful or not) is recorded with duration, phase, and any errors.
- **Client Activity Log** — visible in the WHMCS client area under My Activity Log.
- **Module log** — all Proxmox API calls, DNS provider calls, and non-blocking errors go to WHMCS **Utilities → Logs → Module Log** with identifier `puqProxmoxKVM` and `puq_proxmox_kvm`.
- **Cron output** — when running cron in verbose mode, every step is streamed to stdout in real time.

Related reading

- Deploy Process — same state-machine pattern applied to provisioning.
- Change Package — async package changes.
- VM Management — the admin UI with the Reset and Delete Record actions.
- DNS Zones & Integration — what happens in the DNS deletion phase.

Scheduled Tasks

Proxmox KVM module **WHMCS**

Order now | Download | FAQ

Overview

The module runs six scheduled tasks through the cron system. Each task has a configurable interval and independent lock management to prevent overlapping executions.

Task List

Task	Default Interval	Description
Process VMs	1 minute	Processes the deploy and change package pipelines. Picks up VMs in non-ready states and executes the next step in their pipeline. Also handles DNS record creation and updates. This is the primary task responsible for VM provisioning and modification.
Remove Snapshots	60 minutes	Checks for expired snapshots based on the configured snapshot lifetime setting and automatically removes them from Proxmox. Keeps the snapshot count manageable and frees up storage.
Restore Backup	5 minutes	Monitors active backup restore tasks on Proxmox. When a restore operation completes, it updates the VM status and sends the "Backup restored" email notification to the client.

Task	Default Interval	Description
Backup Status	5 minutes	Monitors active manual backup tasks on Proxmox. When a backup operation completes, it updates the backup record with the result(success or failure).
Schedule Backup	60 minutes	Executes scheduled backups based on per-VM backup schedules. Checks each VM's configured backup days and initiates a backup if one is due. Runs once per day per VM per scheduled day.
Collect Statistics	60 minutes	Aggregates network traffic statistics (inbound and outbound bytes) from Proxmox RRD data. Used for WHMCS Metric Billing to enable usage-based network traffic billing.

Configuring Task Intervals

Task intervals can be adjusted in the addon settings:

1. Navigate to **Addons > PUQ Proxmox KVM**
2. Go to **Settings > Cron**
3. Adjust the interval for each task as needed
4. Save settings

The interval specifies the minimum time between executions of a task. For example, a 5-minute interval means the task will run no more frequently than once every 5 minutes.

> **Tip:** For faster VM provisioning, keep the **Process VMs** interval low (1-2 minutes). For less time-sensitive tasks like statistics collection, longer intervals reduce system load.

Lock Management

Each task uses a lock mechanism to prevent concurrent execution:

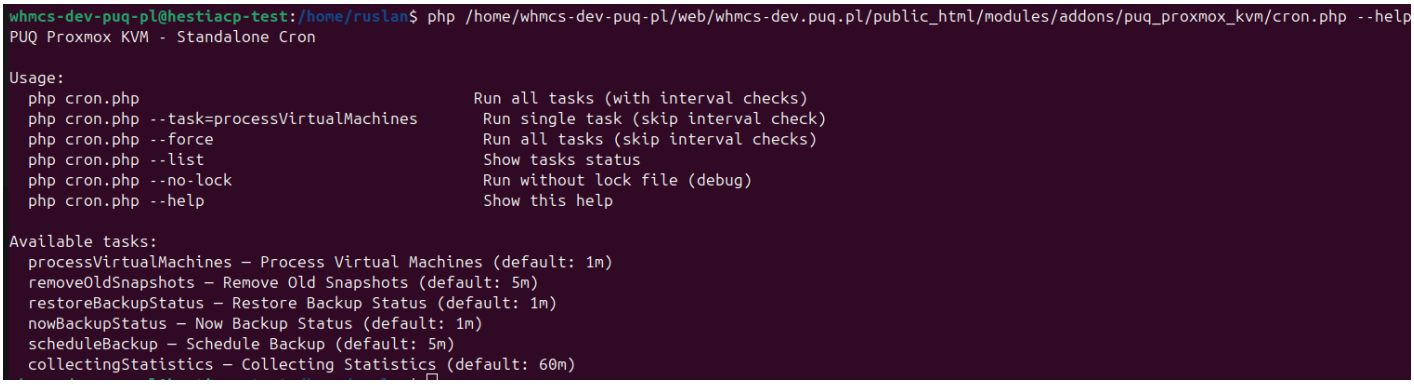
- When a task starts, it acquires a lock
- If the lock is already held, the task is skipped for that cron cycle

- When the task completes, the lock is released
- Stale locks (from crashed processes) are automatically detected and cleared based on a timeout

If a task appears to be stuck, you can check and manage locks from the addon's Cron settings page.

CLI Tools

The module provides command-line tools for manual task execution and diagnostics. These can be useful for troubleshooting or for running tasks on demand outside the normal cron schedule.



To see available CLI commands, run:

```
php /path/to/whmcs/modules/addons/puq_proxmox_kvm/cron.php --help
```

Common CLI Operations

Command	Description
`--help`	Display available commands and usage information
`--run-task=process_vms`	Manually run the Process VMs task
`--clear-locks`	Clear all stale lock files

Monitoring

To monitor cron health:

1. Check the **Last Run** timestamp for each task on the Cron settings page
2. Verify no tasks have stale locks

3. Review the WHMCS activity log for any cron-related errors
4. For deploy issues, check the per-VM deploy log in the VM Management section

If tasks are consistently failing or not running, refer to the Cron Configuration guide to verify your cron setup.