Installation and Configuration

Step-by-step guide to installing the PUQ Proxmox KVM module in WHMCS, connecting it to a Proxmox server or cluster, preparing VM templates, and configuring the noVNC console, email templates and cron. Follow the pages in order for a clean setup.

Basic concepts and requirements

Proxmox KVM module **WHMCS**

Order now | Download | FAQ

System Requirements

Requirement	Supported Versions
WHMCS	8.x, 9.x
PHP	7.4, 8.1, 8.2
Proxmox VE	7.x, 8.x
ionCube Loader	v13 or newer

Required PHP Extensions

The following PHP extensions must be enabled on the WHMCS server:

- **cURL** (`curl`) — required for API communication with Proxmox
- **JSON** (`json`) — required for parsing API responses

Network Requirements

The WHMCS server must be able to reach the Proxmox API over the network on **port 8006** (HTTPS). Ensure that any firewalls between the WHMCS server and the Proxmox host allow outbound TCP connections on this port.

Module Components

The PUQ Proxmox KVM module consists of **two components**. Both are **required** and must be installed for the module to function.

Component	Type	Directory
Server Module	`puqProxmoxKVM`	`modules/servers/puqProxmoxKVM/`
Addon Module	`puq_proxmox_kvm`	`modules/addons/puq_proxmox_kvm/`

The **Server Module** handles VM provisioning, client area interface, admin service management, and all direct Proxmox API operations.

The **Addon Module** manages IP address pools, DNS zones, VM management dashboard, cron task orchestration, and global settings. The server module depends on the addon module for IP allocation, cron processing, and centralized configuration.

> **Note:** The **PUQ Customization** addon module is **no longer required**. All functionality previously provided by PUQ Customization has been replaced by the built-in addon module (`puq_proxmox_kvm`). If you are upgrading from a version prior to v3.0, you may safely remove PUQ Customization after installing the new addon module.

Proxmox Requirements

- API access enabled on the Proxmox host (enabled by default)
- A user account with appropriate permissions for VM management (e.g., `root@pam` or a dedicated API token user)
- At least one storage configured for VM disks
- At least one network bridge configured (e.g., `vmbr0`)
- Cloud-init support on VM templates (recommended)

WHMCS Requirements

- Administrator access to the WHMCS admin area
- File upload permissions to the WHMCS installation directory

- A valid PUQ Proxmox KVM license key
- WHMCS cron job properly configured (for automated provisioning)

Supported Languages

The module includes translations for 25 languages:

Arabic	Azerbaijani	Catalan	Chinese	Croatian
Czech	Danish	Dutch	English	Estonian
Farsi	French	German	Hebrew	Hungarian
Italian	Macedonian	Norwegian	Polish	Romanian
Russian	Spanish	Swedish	Turkish	Ukrainian

Additional operational requirements

- **Continuous and stable network connectivity** between the WHMCS host, the Proxmox cluster and the VNCproxy host. Brief network drops cause deployments to pause and resume on the next cron tick — in v3.0 that's handled by the state machine, but a persistently flaky network will stall provisioning.
- **Static IPs** — if you use static IPv4/IPv6, you need the required number of free IP addresses reserved for virtual machines.
- **DHCP** — if the VM network uses a DHCP server, it must be configured correctly. When the module is set to DHCP, it does **not** manage IP allocation or firewall rules, only bandwidth, bridge and VLAN on the network card.
- **VLANs** — if the network uses VLANs, your internal networking must carry the VLAN to every node of the cluster.
- **noVNC WEB console** — requires a separate VNCproxy installation with access both to the internet and to the Proxmox cluster's VNC port range (5900–5999). See the VNCproxy / noVNC chapter.
- **DNS synchronization** — for forward/reverse DNS sync you need a supported DNS provider (in v3.0: **Cloudflare**, **HestiaCP** or **PowerDNS**, configured in the addon). For legacy setups a DNS API proxy / external automation against the `dns.php` endpoint is still supported.
- **Single-node installs** — a **Directory** or **NFS** datastore is required for VM disks.
- **ISO storage** — ISO images can live on a separate network storage configured as ISO

storage in the product.

- **Backup storage** — backups also need network storage that is reachable from every node. Proxmox Backup Server is supported. Make sure the datastore intended for backups does **not** aggressively rotate copies, or that its rotation is aligned with the backup count defined in the client's package.
- **Anti-spoofing firewall** — if you want firewall rules that protect against IP spoofing, the firewall on the Proxmox server/cluster must be preconfigured with an incoming/outgoing **DENY** policy. The module then adds permissive rules matching the VM's own IP.

The logic of the module

This section is a high-level overview of what happens for each lifecycle operation. In v3.0 every stage is driven by a **state machine** with resume-on-failure semantics; in v2.x the same steps were executed as one monolithic cron call.

Creating a new virtual machine

1. After the client orders and pays for a virtual machine service, WHMCS calls the `CreateAccount` function.
2. An available IP address is selected from the server's IP pool. *Note: IPs of **Terminated** services are recycled back into the free pool and may be reused.*
3. A free virtual machine **VMID** is chosen — unique both in WHMCS and in Proxmox.
4. The hostname and VM name are generated from the package template (`<prefix>-<client_id>-<service_id>`).
5. The module starts cloning the virtual machine from the configured template.
6. The client is notified by email that the virtual machine is being created (**Welcome email**).
7. From this point the internal **cron** takes over and walks the VM through the deploy state machine. Each run of cron advances one or more steps depending on what's ready.

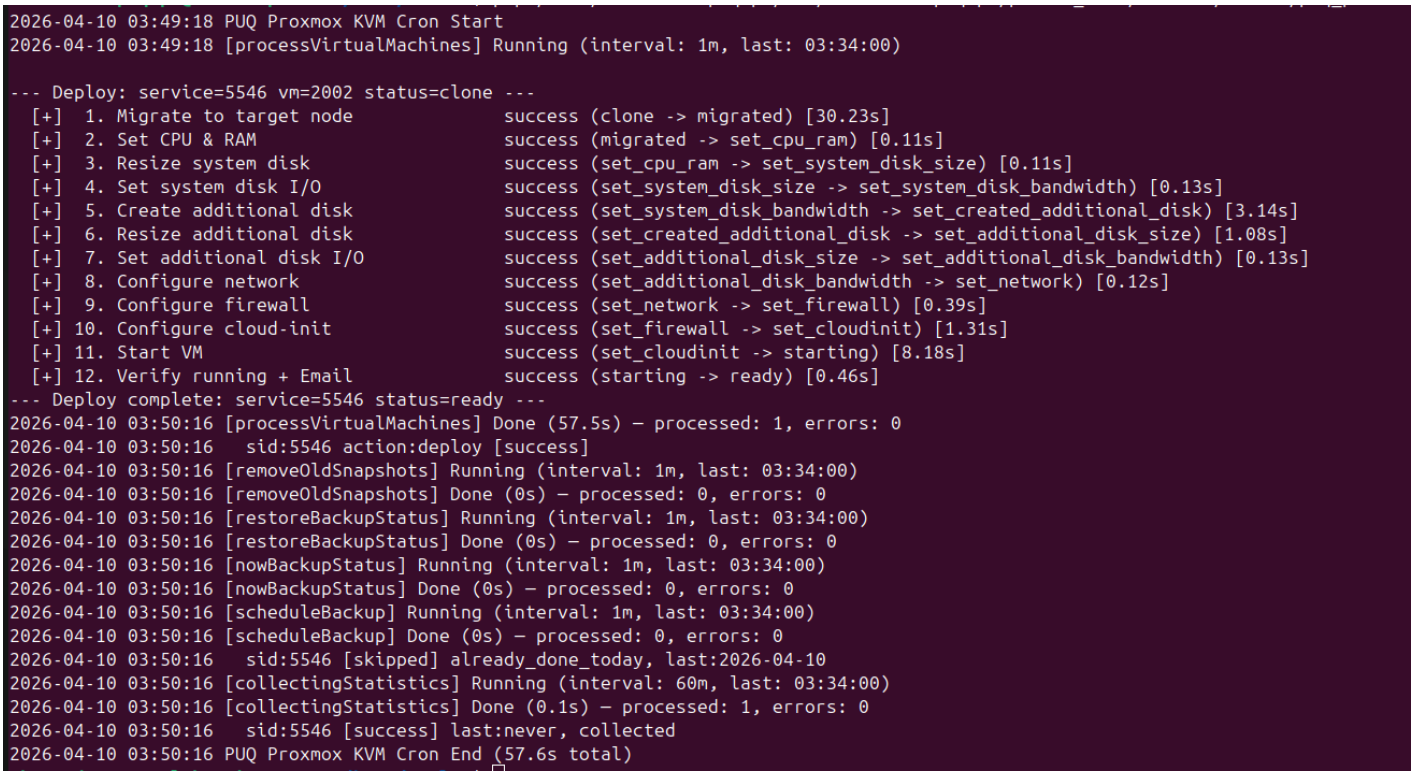
> **Changed in v3.0.** The deploy pipeline is a proper state machine — on any failure the VM stays in the last successful state and the next cron tick resumes from there. Earlier versions ran all steps in one go and would stall the whole service on a single transient error.

Deploy steps (v3.0):

1. `VMSetDedicatedIp` — allocate a free IP from the pool
2. `VMSetDNSRecords` — create forward/reverse DNS records (non-blocking — a failed DNS provider does not stop deployment)
3. `VMClone` — clone from the template (always on the template node)

4. `migrateToTargetNode` *(new in v3.0)* — offline migrate the freshly cloned VM to the target node / target storage
5. `VMSetCpuRam`
6. `VMSetSystemDiskSize`
7. `VMSetSystemDiskBandwidth`
8. `VMSetCreatedAdditionalDisk` (skipped if additional disk is not configured)
9. `VMSetAdditionalDiskSize`
10. `VMSetAdditionalDiskBandwidth`
11. `VMSetNetwork` — bridge, VLAN, MAC, bandwidth
12. `VMSetFirewall` — options, anti-spoofing IPSet, policies *(extended in v3.0 via the new `VMFirewall` class)*
13. `VMSetCloudinit` — user, password, network config, hostname
14. `VMStart`
15. **Verify running +** `ServiceSendEmailVMReady` — success email with access parameters (IP, user, password)

Example of a successful cron run (v3.0 output format):




```
2026-04-10 03:49:18 PUQ Proxmox KVM Cron Start
2026-04-10 03:49:18 [processVirtualMachines] Running (interval: 1m, last: 03:34:00)
--- Deploy: service=5546 vm=2002 status=clone ---
[+] 1. Migrate to target node success (clone ->
migrated) [30.23s]
[+] 2. Set CPU & RAM success (migrated ->
set_cpu_ram) [0.11s]
[+] 3. Resize system disk success (set_cpu_ram ->
set_system_disk_size) [0.12s]
```

```
[+]  4. Set system disk I/O          success (set_system_disk_size ->
set_system_disk_bandwidth) [0.13s]
[+]  5. Create additional disk       success (set_system_disk_bandwidth ->
set_created_additional_disk) [3.14s]
[+]  6. Resize additional disk       success (set_created_additional_disk ->
set_additional_disk_size)  [1.08s]
[+]  7. Set additional disk I/O      success (set_additional_disk_size ->
set_additional_disk_bandwidth)[0.13s]
[+]  8. Configure network            success (set_additional_disk_bandwidth ->
set_network)    [0.12s]
[+]  9. Configure firewall           success (set_network ->
set_firewall)                     [0.39s]
[+] 10. Configure cloud-init         success (set_firewall ->
set_cloudinit)                   [1.31s]
[+] 11. Start VM                     success (set_cloudinit ->
starting)                       [8.18s]
[+] 12. Verify running + Email       success (starting ->
ready)                               [0.46s]
--- Deploy complete: service=5546 status=ready ---
2026-04-10 03:50:16 [processVirtualMachines] Done (57.5s) — processed: 2, errors: 0
2026-04-10 03:50:16   sid:5546 action:deploy result:success
2026-04-10 03:50:16 PUQ Proxmox KVM Cron End (57.6s total)
```

Key things to notice in the v3.0 format:

- Every cron tick is wrapped between `PUQ Proxmox KVM Cron Start` / `... Cron End` lines, so you can see exactly which tick did what.
- Each deploy is introduced by `--- Deploy: service=<sid> vm=<vmid> status=<current_status> ---` and terminated by `--- Deploy complete: ... status=ready ---`.
- Every step prints a human label (`1. Migrate to target node`, `2. Set CPU & RAM`, ...), the `success` keyword, a **state transition** (`clone -> migrated`) and the **duration in seconds** in square brackets. This makes it trivial to spot the one step that is slow.
- After the deploy block the cron task summary is printed: `[processVirtualMachines] Done (X.Xs) — processed: N, errors: M`. Non-zero `errors` means at least one VM ran into a problem — look at the per-sid lines directly below.
- At the end every task's structured result (`processed`, `errors`) is rolled up into a short single line — useful for external monitors that tail `stdout`.

> **Changed in v3.0.** The v1.x–v2.x output was a flat list like `VMSetCpuRam: success`

> . It is gone entirely — if you see that format anywhere, you are running an older version.

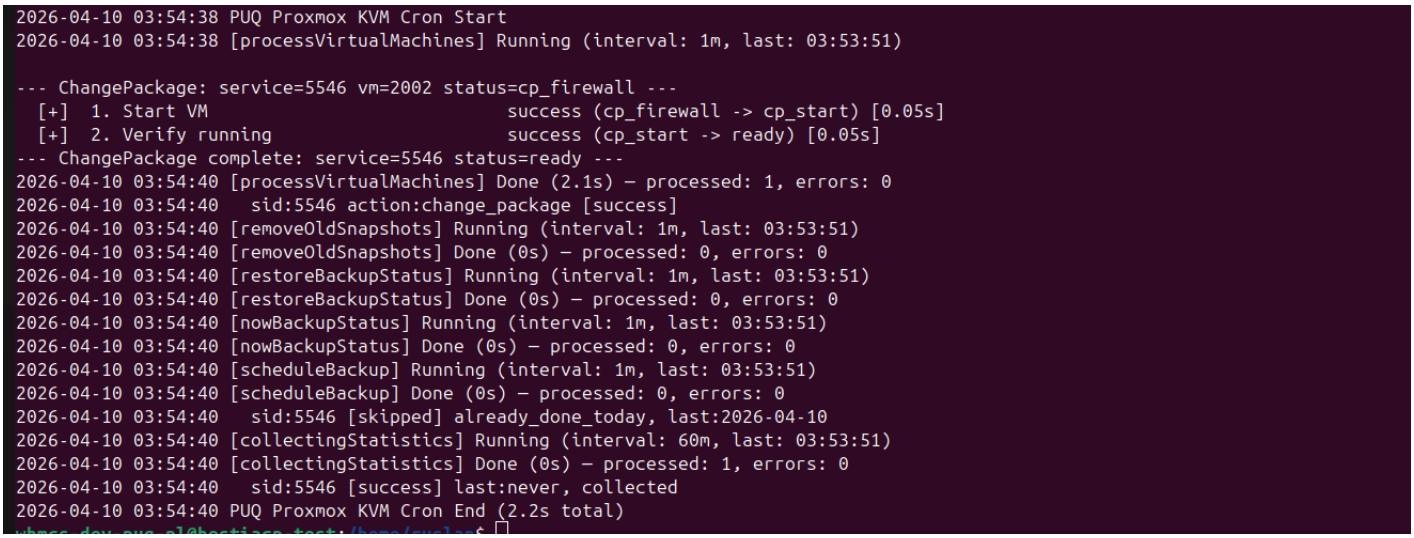
Changing the package of an existing virtual machine

Package upgrades and downgrades (WHMCS → client or admin → *Upgrade/Downgrade*) use the **same state machine pattern as deploy** — they walk the VM through a separate set of `cp_*` states, one per change to apply. See also the `change_package` admin section for the full step list and state values.

Every `change_package` step also:

- checks whether the new package value actually differs from what the VM has right now and **skips the step if there is no change** (you will see `skip — no change` in the output),
- prints its own timing and transition in the same bracketed `[x.xxs]` format as deploy,
- stops the VM once at the beginning (`cp_stop`) and starts it again at the end (`cp_start`) — this is the only part of the pipeline that is guaranteed to happen.

Example of a successful change-package cron run:



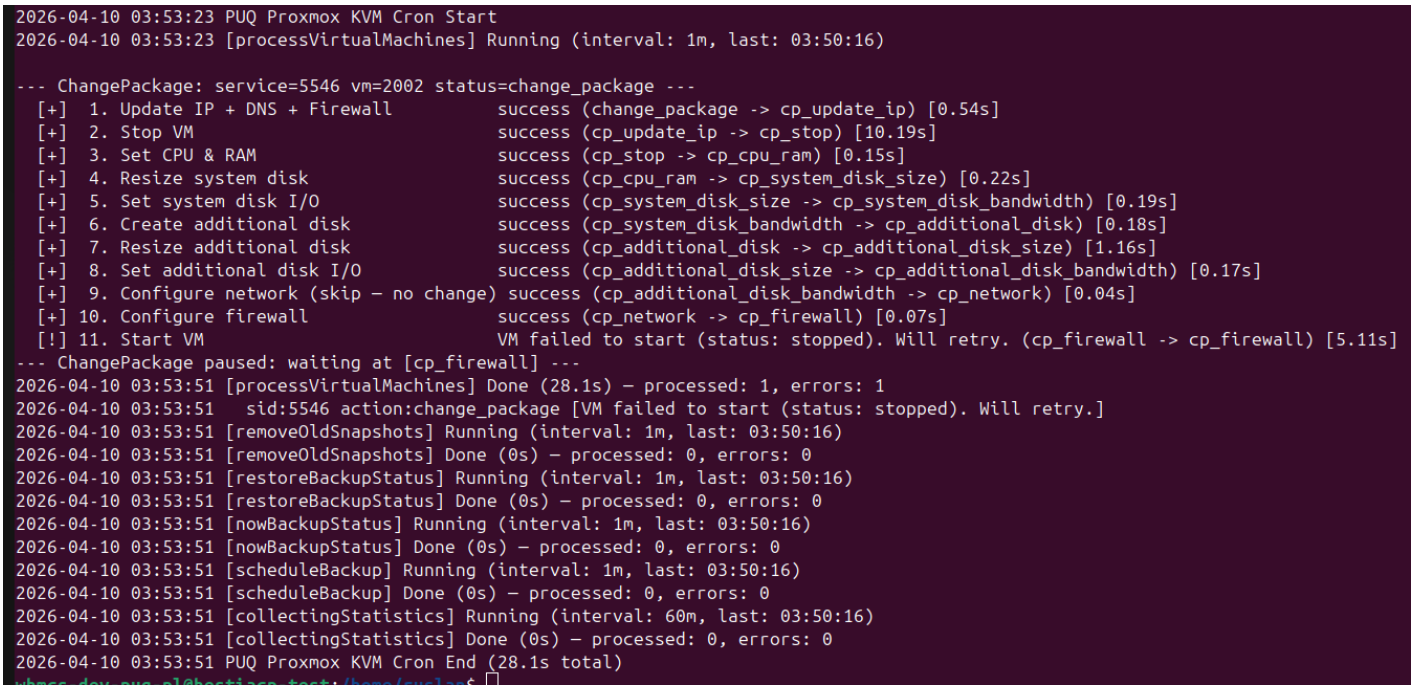
```
2026-04-10 03:54:38 PUQ Proxmox KVM Cron Start
2026-04-10 03:54:38 [processVirtualMachines] Running (interval: 1m, last: 03:53:45)

--- ChangePackage: service=5546 vm=2002 status=cp_firewall ---
  [+]  1. Configure firewall    success (cp_firewall -> cp_start) [0.05s]
  [+]  2. Verify VM             success (cp_start -> ready)       [0.05s]
--- ChangePackage complete: service=5546 status=ready ---
```

```
2026-04-10 03:54:40 [processVirtualMachines] Done (2.1s) — processed: 2, errors: 0
2026-04-10 03:54:40   sid:5546 action:change_package result:success
2026-04-10 03:54:40 PUQ Proxmox KVM Cron End (2.2s total)
```

This second run only does two steps because the **previous** cron tick had already performed the heavy part of the change (`cp_cpu_ram`, `cp_system_disk_*`, `cp_additional_disk_*`, `cp_network`) and the state machine stopped at `cp_firewall`. On resume it picks up from exactly where it was — which is the whole point of the state machine.

Example of a change-package that partially failed and is scheduled for retry:




```
2026-04-10 03:53:23 PUQ Proxmox KVM Cron Start
2026-04-10 03:53:23 [processVirtualMachines] Running (interval: 1m, last: 03:50:16)
--- ChangePackage: service=5546 vm=2002 status=change_package ---
[+]  1. Update IP + DNS + Firewall            success (change_package ->
cp_stop)                                [0.54s]
[+]  2. Stop VM                                success (cp_stop ->
cp_cpu_ram)                                [10.19s]
[+]  3. Set CPU & RAM                          success (cp_cpu_ram ->
cp_system_disk_size)                    [0.22s]
[+]  4. Resize system disk                     success (cp_system_disk_size ->
cp_system_disk_bandwidth)       [0.19s]
[+]  5. Set system disk I/O                    success (cp_system_disk_bandwidth ->
cp_additional_disk)        [0.22s]
[+]  6. Create additional disk                 success (cp_additional_disk ->
cp_additional_disk_size)         [1.16s]
```

```
[+]  7. Resize additional disk               success (cp_additional_disk_size ->
cp_additional_disk_bandwidth)[0.17s]
[+]  8. Set additional disk I/O              success (cp_additional_disk_bandwidth ->
cp_network)            [0.04s]
[+]  9. Configure network (skip — no change)  success (cp_network ->
cp_firewall)                              [0.07s]
[+] 10. Configure firewall                   success (cp_firewall ->
cp_start)                                 [ ... ]
[+] 11. Start VM                             VM failed to start (status: stopped). Will
retry.               [5.11s]
--- ChangePackage paused: waiting at (cp_firewall) ---
2026-04-10 03:53:51 [processVirtualMachines] Done (28.1s) — processed: 2, errors: 1
2026-04-10 03:53:51   sid:5546 action:change_package VM failed to start (status: stopped).
Will retry. (cp_firewall -> cp_start) [5.11s]
2026-04-10 03:53:51 PUQ Proxmox KVM Cron End (28.1s total)
```

Things to notice in this retry run:

- Step 9 printed `(skip — no change)` because the product's network bridge/VLAN did not actually change — the state machine still advances the status (`cp_network -> cp_firewall`) but does not touch Proxmox at all.
- Step 11 failed on `VM failed to start`. Instead of rolling back the whole change, the state machine **pauses** and the overall cron summary ends with `ChangePackage paused: waiting at (cp_firewall)` + `errors: 1`.
- On the next cron tick the module picks up at `cp_firewall` and runs steps 10–11 again. That is exactly the previous successful run shown above.

> **Changed in v3.0.** `change_package` was an atomic single-shot operation in v1.x–v2.x — a failure during the resize or the start step would leave the VM in an inconsistent half-changed state and the admin had to fix it by hand. The v3.0 state machine makes the whole thing idempotent and recoverable on the next cron tick.

Reinstalling the virtual machine

The reinstallation procedure removes the VM and recreates it using the current package parameters while keeping the original IP, MAC address, VLAN and VMID.

1. Snapshots are deleted. **Backups are kept intact** — you can still restore a backup from the pre-reinstall state.

2. The VM is removed.
3. A fresh clone is started from the template chosen during the reinstall action (can be a different OS than before).
4. The state machine then runs through: `VMSetCpuRam → VMDeleteDNSRecords → VMSetDNSRecords → VMSetSystemDiskSize → VMSetSystemDiskBandwidth → additional disk steps → VMSetNetwork → VMSetFirewall → VMSetCloudinit → VMStart`.
5. A success email is sent to the client with the access parameters.

Snapshots

- The client can create, delete and restore snapshots of their VM from the client area.
- The number of snapshots is limited in the package configuration.
- Snapshot lifetime is configured per-product (1–10 days maximum).
- A cron task automatically deletes snapshots older than the configured lifetime.

Backups

- The client can create, delete and restore backups directly from the client area.
- The number of backups is limited in the package configuration.
- **Automatic backups**: the client selects the days on which the backup should run. The exact time-of-day is assigned automatically and randomly by the cron system each time the schedule is saved — this is done to spread the backup load across your Proxmox storage.
- On each cron tick, the module:
  1. Checks whether today's schedule entry for this VM is due (i.e. the target time is in the past compared to "now").
  2. Checks whether a backup for today already exists.
  3. Checks whether the backup slot limit has been reached — if yes, deletes the oldest backup to make room.
  4. Creates the new backup.

Backup recovery

- The VM must be **powered off** before the backup restore runs.
- Once the restore completes, the module:
  - Re-applies CPU/RAM if the package values differ from the restored VM.
  - Re-applies disk size and bandwidth.
  - Re-creates additional disks if needed.
  - Re-applies network configuration (bridge, VLAN, bandwidth, MAC).
  - Starts the VM.
  - Sends the **Backup restored** email to the client.
- If the restore fails, the client is given the option to retry the restore or to reinstall the VM.

- While a backup is being created or restored, **all other management operations on the VM are suspended**.

Reset password

The password reset procedure relies on cloud-init — it works only if the `cloud-init` packages have not been removed from the VM (see the Virtual Machine Templates chapter).

1. The VM must be **powered off**.
2. A new random password is generated and saved in the WHMCS service settings.
3. Cloud-init is rewritten with the new credentials.
4. The VM is started.
5. The **Reset password** email is sent to the client with the new access parameters.

Mounting an ISO image

ISO images are stored on Proxmox in the usual way (shared storage in clusters; directory storage is fine on single nodes). Upload ISOs to Proxmox in advance.

To make the selection easier in the client UI, the module groups ISO images by the part of the filename **before the first `-` character**:

- `Debian-12.5.0-amd64-netinst.iso` → group **Debian**
- `Ubuntu-22.04.4-live-server-amd64.iso` → group **Ubuntu**
- `proxmox-ve_8.1-2.iso` → group **proxmox** (no dash → the `_` separator is **not** used, so this file lands in the **OTHER** group — rename your files accordingly)
- `myfile.iso` (no dash) → group **OTHER**

Pay attention to your file naming conventions when uploading ISOs.

WHMCS Module Installation and Update

Proxmox KVM module **WHMCS**

Order now | Download | FAQ

Download

The module is distributed as a single ZIP archive. A separate build is published for each supported PHP major version — pick the one that matches the PHP runtime used by your WHMCS installation.

All versions and historical builds are available in the index:

- https://download.puqcloud.com/WHMCS/servers/PUQ_WHMCS-Proxmox-KVM/

Direct "latest" downloads

PHP 8.2

```
wget https://download.puqcloud.com/WHMCS/servers/PUQ_WHMCS-Proxmox-KVM/php82/PUQ_WHMCS-Proxmox-KVM-latest.zip
```

PHP 8.1

```
wget https://download.puqcloud.com/WHMCS/servers/PUQ_WHMCS-Proxmox-KVM/php81/PUQ_WHMCS-Proxmox-KVM-latest.zip
```

PHP 7.4

```
wget https://download.puqcloud.com/WHMCS/servers/PUQ_WHMCS-Proxmox-KVM/php74/PUQ_WHMCS-
```

```
Proxmox-KVM-latest.zip
```

> Not sure which PHP version your WHMCS runs on? Check **Utilities > System > PHP Info** in the WHMCS admin area.

Installation

Step 1: Unzip the Archive

On your WHMCS server (or locally, before uploading):

```
unzip PUQ_WHMCS-Proxmox-KVM-latest.zip
```

The archive extracts into a `PUQ_WHMCS-Proxmox-KVM/` directory containing two module folders: `puqProxmoxKVM` (server module) and `puq_proxmox_kvm` (addon module).

Step 2: Copy the Server Module

Copy and replace `puqProxmoxKVM` from the extracted `PUQ_WHMCS-Proxmox-KVM/` directory to your WHMCS installation:

```
PUQ_WHMCS-Proxmox-KVM/puqProxmoxKVM  →  WHMCS_WEB_DIR/modules/servers/puqProxmoxKVM/
```

Example:

```
cp -r PUQ_WHMCS-Proxmox-KVM/puqProxmoxKVM /var/www/html/whmcs/modules/servers/
```

Step 3: Copy the Addon Module

Copy and replace `puq_proxmox_kvm` from the extracted directory to your WHMCS installation:

```
PUQ_WHMCS-Proxmox-KVM/puq_proxmox_kvm  →  WHMCS_WEB_DIR/modules/addons/puq_proxmox_kvm/
```

Example:

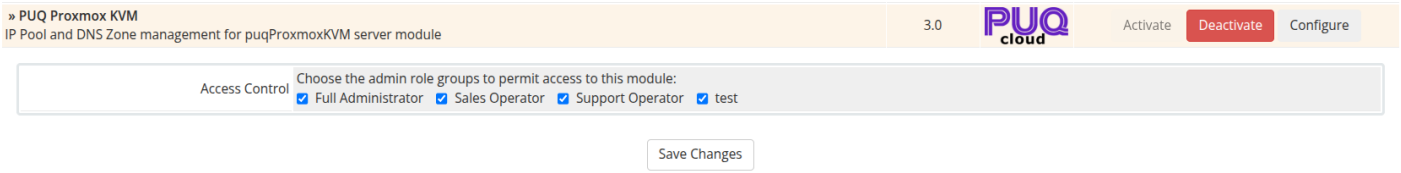
```

```

```
cp -r PUQ_WHMCS-Proxmox-KVM/puq_proxmox_kvm /var/www/html/whmcs/modules/addons/
```

Step 4: Activate the Addon Module

1. Log in to the WHMCS admin area
2. Navigate to **Setup > Addon Modules**
3. Find **PUQ Proxmox KVM** in the list
4. Click **Activate**
5. Enter your license key
6. Configure access control to grant the appropriate admin roles access to the addon



Step 5: Verify Installation

After activation, navigate to **Addons > PUQ Proxmox KVM** in the admin menu. You should see the addon dashboard confirming a successful installation.

File Structure

After installation, the module files should be located at:

```
whmcs/
├── modules/
│   ├── servers/
│   │   └── puqProxmoxKVM/          # Server module
│   │       ├── puqProxmoxKVM.php
│   │       └── ...
│   └── addons/
│       └── puq_proxmox_kvm/        # Addon module
│           ├── puq_proxmox_kvm.php
│           └── ...
```

Update Procedure

To update the module to a newer version:

1. **Deactivate** the addon module in **Setup > Addon Modules**
2. Download the latest module archive from puqcloud.com
3. Upload and overwrite the server module files in `modules/servers/puqProxmoxKVM/`
4. Upload and overwrite the addon module files in `modules/addons/puq_proxmox_kvm/`
5. **Reactivate** the addon module in **Setup > Addon Modules**

> **Important:** Database tables and all configuration data are preserved during the deactivate/reactivate cycle. Your IP pools, DNS zones, VM records, and settings will remain intact.

> **Tip:** Always back up your WHMCS installation before performing an update.

Addon Module Setup

Proxmox KVM module **WHMCS**

Order now | Download | FAQ

After Activation

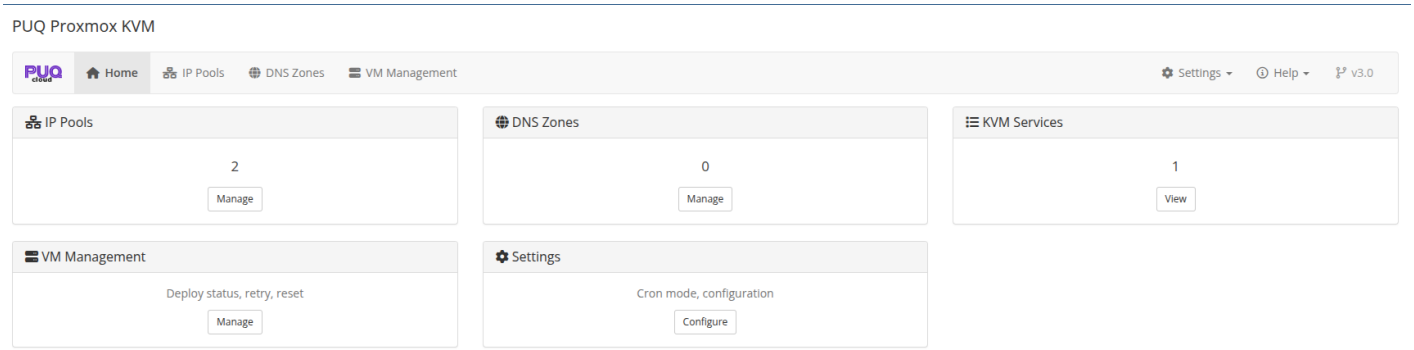
When the addon module is activated for the first time, it automatically:

- Creates all required database tables
- Sets default values for all settings
- Initializes the cron system

No manual database setup is required.

Accessing the Addon

Navigate to **Addons > PUQ Proxmox KVM** in the WHMCS admin menu. The addon dashboard provides a centralized management interface.



Addon Features

The addon module provides:

- **IP Pools** — Manage IPv4 and IPv6 address pools per server, with automatic allocation during VM deployment
- **DNS Zones** — Configure Cloudflare and HestiaCP integration for forward and reverse DNS automation
- **VM Management** — Overview of all provisioned VMs with deploy logs and status tracking
- **Cron Tasks** — Configure cron intervals, view task status, and manage lock files
- **Settings** — Global module settings including API timeouts, migration behavior, and cron configuration

PUQ Customization Addon No Longer Required

Starting from v3.0, the PUQ Proxmox KVM module includes its own dedicated addon module (`puq_proxmox_kvm`). The old PUQ Customization addon with the ModulePuqProxmoxKVM extension is **no longer needed**.

If you are upgrading from an earlier version that used PUQ Customization:

1. Install and activate the new standalone addon module
2. Existing data (IP pools, DNS zones) will be migrated automatically on activation
3. Verify that all IP pools and DNS zones are present in the new addon
4. You can then safely deactivate the ModulePuqProxmoxKVM extension in PUQ Customization

> **Note:** The server module supports both the new standalone addon and the old PUQ Customization addon simultaneously during the migration period, so there is no downtime during the transition.

Create new server for Proxmox in WHMCS

Proxmox KVM module **WHMCS**

Order now | Download | FAQ

Preface

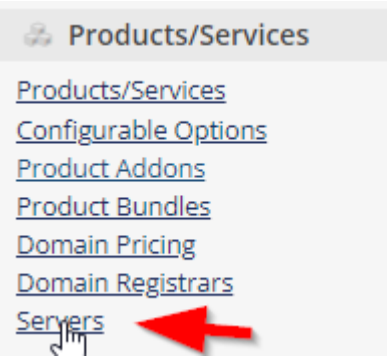
For the module to work properly, you must configure the server settings in your main WHMCS panel. This is the place where you register a Proxmox server (or Proxmox cluster) which will then be used by the module to build KVM virtual machines. Here you define access credentials, IP ranges and additional settings.

> **Attention.** If you have only one server, or you do not use server groups, you need to make this server the **active default** for new signups by opening the server entry in WHMCS and ticking "*Make this server the active default for new signups*".

Server creation

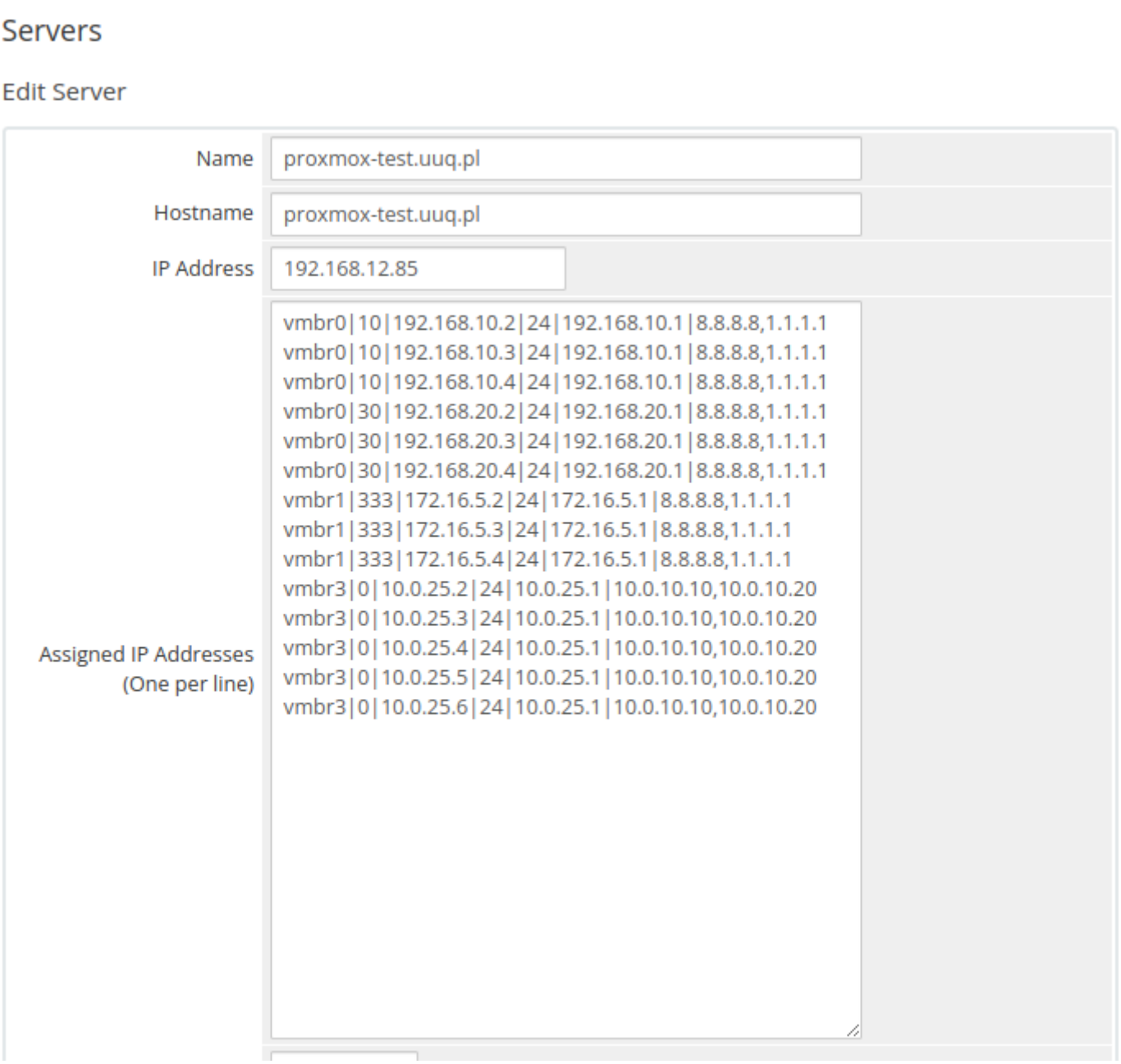
Log in to your WHMCS panel and create a new Proxmox server:

System Settings → Products/Services → Servers → Add New Server



Step 1: Name, Hostname and Assigned IP Addresses

- Enter the correct **Name** and **Hostname** of the Proxmox node.
- In the **Assigned IP Addresses** field enter the list of IP addresses that will be reserved for virtual machines built on this server.



> **Note.** Starting with module version **1.3**, the module supports IPv4/IPv6 pools managed in the addon. For new installations this is the recommended way to manage IP addresses — see the **IP Pools** chapter of this documentation. The "Assigned IP Addresses" field described below is the legacy format and is kept for backward compatibility.

Format to follow in the Assigned IP Addresses field

To define the available pool of IP addresses, enter one line per IP, with fields separated by the `|` character. Each line has the following structure:

```
<bridge>| <vlan_tag>| <IP_address>| <net_mask>| <Gateway>| <DNS1>, <DNS2>
```

Field	Description
`<bridge>`	The virtual bridge to which the VM network interface is connected (e.g. `vmbr0`).
`<vlan_tag>`	VLAN tag that will be set on the VM's network card. If VLANs are not used, enter `0`.
`<IP_address>`	IPv4 address that will be assigned to the VM.
`<net_mask>`	Network mask in CIDR form (e.g. `24`).
`<Gateway>`	Default gateway for the subnet.
`<DNS1>, <DNS2>`	Comma-separated list of DNS servers.

Example

```
vmbr0| 10| 192. 168. 10. 2| 24| 192. 168. 10. 1| 8. 8. 8. 8, 1. 1. 1. 1
vmbr0| 10| 192. 168. 10. 3| 24| 192. 168. 10. 1| 8. 8. 8. 8, 1. 1. 1. 1
vmbr0| 10| 192. 168. 10. 4| 24| 192. 168. 10. 1| 8. 8. 8. 8, 1. 1. 1. 1
vmbr0| 30| 192. 168. 20. 2| 24| 192. 168. 20. 1| 8. 8. 8. 8, 1. 1. 1. 1
vmbr0| 30| 192. 168. 20. 3| 24| 192. 168. 20. 1| 8. 8. 8. 8, 1. 1. 1. 1
vmbr0| 30| 192. 168. 20. 4| 24| 192. 168. 20. 1| 8. 8. 8. 8, 1. 1. 1. 1
vmbr1| 333| 172. 16. 5. 2| 24| 172. 16. 5. 1| 8. 8. 8. 8, 1. 1. 1. 1
vmbr1| 333| 172. 16. 5. 3| 24| 172. 16. 5. 1| 8. 8. 8. 8, 1. 1. 1. 1
vmbr1| 333| 172. 16. 5. 4| 24| 172. 16. 5. 1| 8. 8. 8. 8, 1. 1. 1. 1
vmbr3| 0| 10. 0. 25. 2| 24| 10. 0. 25. 1| 10. 0. 10. 10, 10. 0. 10. 20
vmbr3| 0| 10. 0. 25. 3| 24| 10. 0. 25. 1| 10. 0. 10. 10, 10. 0. 10. 20
vmbr3| 0| 10. 0. 25. 4| 24| 10. 0. 25. 1| 10. 0. 10. 10, 10. 0. 10. 20
vmbr3| 0| 10. 0. 25. 5| 24| 10. 0. 25. 1| 10. 0. 10. 10, 10. 0. 10. 20
vmbr3| 0| 10. 0. 25. 6| 24| 10. 0. 25. 1| 10. 0. 10. 10, 10. 0. 10. 20
```

Step 2: Server Details — module and credentials

In the **Server Details** section select the **PUQ Proxmox KVM** module and enter the correct credentials for the Proxmox API. Then click **Test connection** to verify.

> **Attention.** Starting from module version **2.3**, authentication has been changed

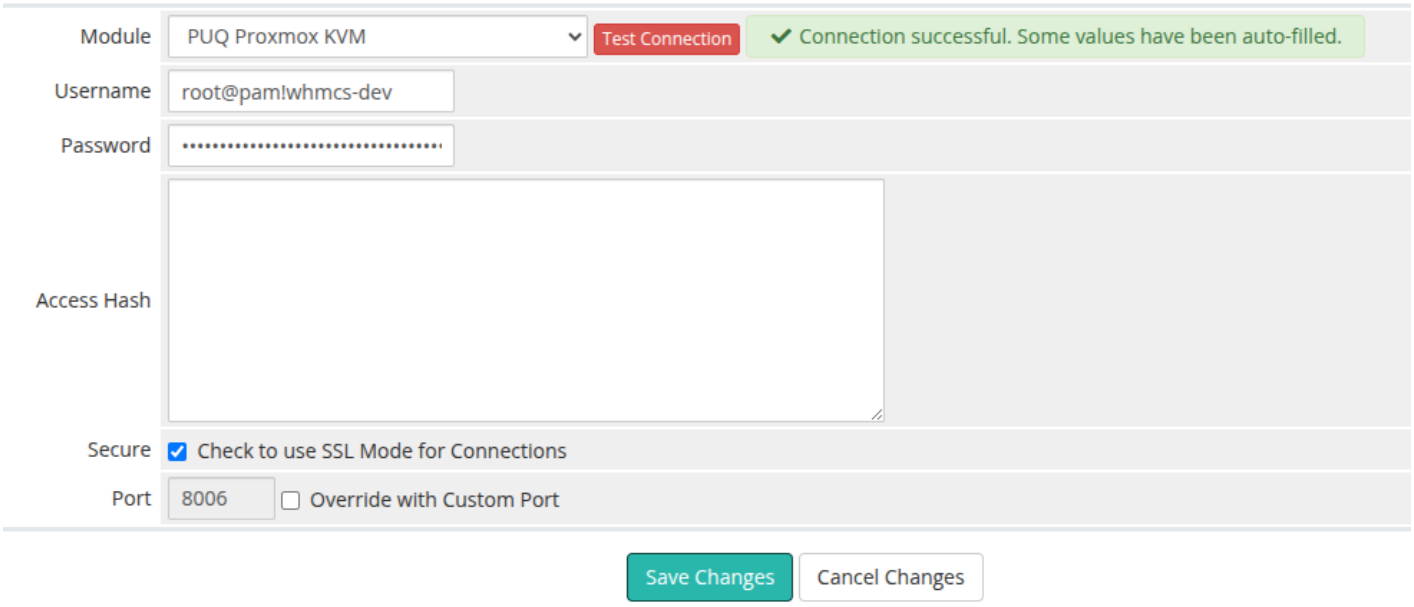
to **token-based**.

- **Username** — Proxmox token ID in the format `root@pam!whmcs-dev`
- **Password** — the token secret value

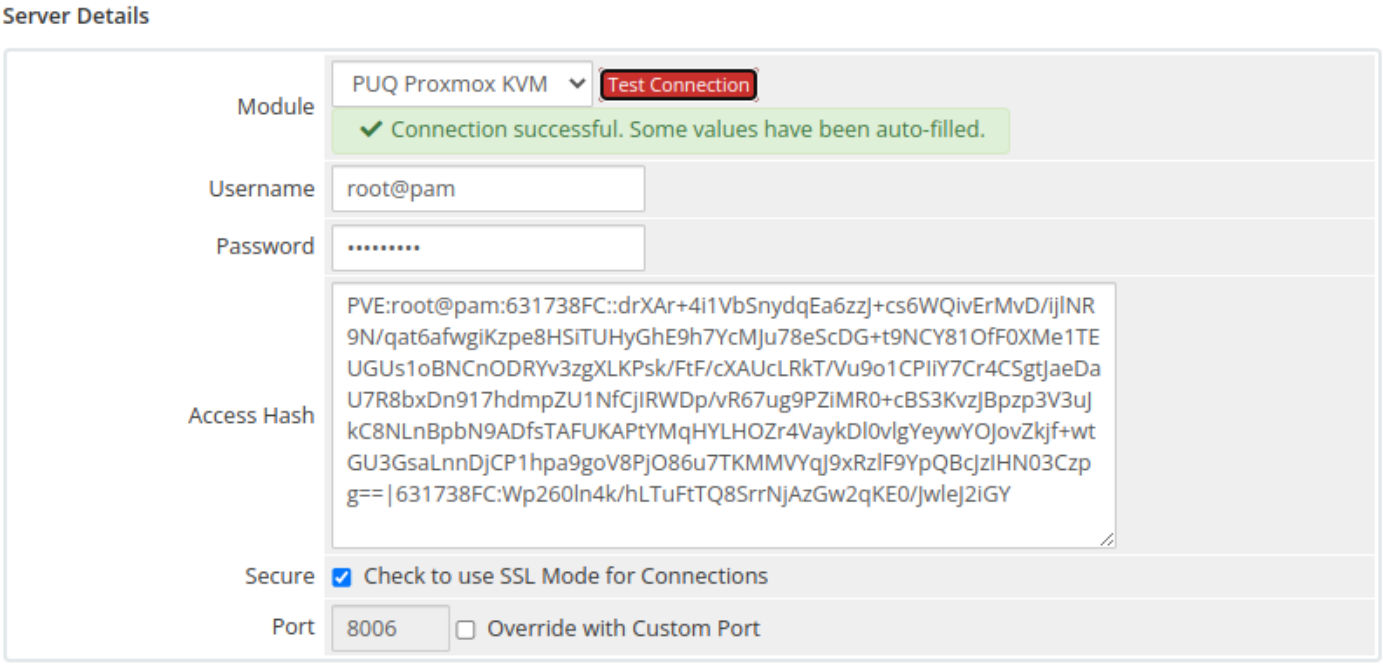
If you are using a version earlier than 2.3, enter the Proxmox username in the format `root@pam` in the **Username** field and the corresponding password in the **Password** field.

During operation, the module will automatically fill in the **Access Hash** field. You do not need to fill it manually.

Version 2.3+ — Token authentication



Version 2.2 and earlier — Password authentication



Creating a Proxmox API Token

1. Log in to the Proxmox web UI.
2. Go to **Datacenter → Permissions → API Tokens**.
3. Click **Add**.
4. Select the **User** (e.g. `root@pam`).
5. Enter a **Token ID** (e.g. `whmcs`).
6. **Uncheck** *Privilege Separation* if the token should inherit the user's full permissions. If privilege separation is enabled, you must assign permissions to the token itself.
7. Click **Add**.
8. **Copy the generated token secret immediately** — it is displayed only once and cannot be retrieved later.

The resulting username for WHMCS will look like `root@pam! whmcs` and the password will be the token secret (a UUID-like string).

Step 3: Make the server default (single-server installs)

If you have only one Proxmox server, or you do not use server groups, open the server entry and tick **"Make this server the active default for new signups"**. Otherwise newly ordered products will not be assigned to this server automatically.

Test Connection

After saving the server configuration, always use the **Test Connection** button to verify:

- Network connectivity to the Proxmox host on port **8006**
- Authentication credentials are valid (token ID + secret, or username + password)
- The API user / token has sufficient permissions on the target nodes and storages

If the test fails, check:

- The WHMCS server can reach the Proxmox host on port `8006`
- The username and password/token are correct
- The Proxmox API service (`pveproxy`) is running
- No firewall is blocking the connection between WHMCS and Proxmox

Server Groups

You can organize multiple Proxmox servers into **Server Groups** for automatic server selection during provisioning. This is useful when you have a Proxmox cluster with multiple nodes.

1. Go to **System Settings → Products/Services → Servers**
2. Click the **Server Groups** tab
3. Create a new group and assign your Proxmox servers to it
4. Set the **Fill Type**:
   - **Fill** — fills one server before moving to the next
   - **Round Robin** — distributes VMs evenly across servers
5. When configuring a product, select the server group instead of a specific server

> **Tip.** When using server groups with a Proxmox cluster, ensure the required storages exist on every node, or enable the VM migration step in the addon settings so the module moves freshly-cloned VMs to the correct target node automatically.

Virtual Machine Templates

Proxmox KVM module **WHMCS**

Order now | Download | FAQ

Overview

The module provisions virtual machines by cloning existing Proxmox VM templates. Before using the module, you need to prepare at least one VM template in your Proxmox environment. Templates must be properly prepared inside the Proxmox panel — the module does not install an operating system from ISO, it only clones a ready template and applies configuration via cloud-init.

Physical Requirements

VM templates must meet the following specifications:

- **Resource sizing**: all template parameters (CPU cores, RAM, system disk size) must be **smaller** than the smallest package you plan to offer clients. The module can grow disks and increase CPU/RAM during deployment, but it **cannot shrink** them.
- **Multi-disk layout**: if you plan to offer VMs with multiple disks on different storage locations, create those disks on the appropriate storage already in the template. Otherwise Proxmox may consolidate all disks on the same storage during clone.
- **Cloud-init drive**: a cloud-init disk is mandatory for automatic VM configuration after cloning.
- **Partition layout**: partitions of the system disk must be arranged so that the **root partition is the LAST** partition in the table. This is required for automatic root filesystem expansion by `cloud-initramfs-growroot` during first boot.

Cloud-Init Requirement

Cloud-init is **required** for automatic VM configuration. The module uses cloud-init to set:

- Hostname
- IP address and network configuration
- DNS servers
- User account and password
- SSH keys

Without cloud-init, the module cannot automatically configure the VM's network and credentials after cloning.

Creating a Template

Step 1: Create a Base VM

Create a new VM in Proxmox with your desired operating system. When partitioning the system disk, ensure the **root partition is the LAST** one in the partition table so that it can be automatically expanded on first boot.

Step 2: Enable Root SSH Access

The module uses the `root` user to push cloud-init configuration and manage the VM. Enable root login via SSH inside the template:

```
# /etc/ssh/sshd_config
PermitRootLogin yes
PasswordAuthentication yes

systemctl restart sshd
```

Step 3: Install Cloud-Init

Install cloud-init together with the growroot and utility packages inside the VM:

```
# Debian/Ubuntu
apt update && apt install -y cloud-initramfs-growroot cloud-init cloud-utils

# CentOS/RHEL/AlmaLinux
yum install -y cloud-init cloud-utils-growpart
```

```
# openSUSE
zypper install cloud-init growpart
```

> `cloud-initramfs-growroot` / `cloud-utils-growpart` is what actually expands the root partition to fill the resized disk during deployment. Without it the client VM will boot with the original template disk size.

Step 4: Enable Cloud-Init Services

Cloud-init is split into four systemd services — all of them must be enabled so the VM picks up configuration on every boot:

```
systemctl enable cloud-init-local.service
systemctl enable cloud-init.service
systemctl enable cloud-config.service
systemctl enable cloud-final.service
```

Step 5: Clean Up the VM

Before converting to a template, clean up the VM so that every clone starts fresh:

```
# Remove default users created during OS install (ubuntu, debian, centos, etc.)
# The module creates the user defined in the product configuration.
userdel -r ubuntu 2>/dev/null || true
userdel -r debian 2>/dev/null || true

# Remove SSH host keys (they will be regenerated on first boot)
rm -f /etc/ssh/ssh_host_*

# Clean cloud-init state
cloud-init clean --logs

# Remove machine ID (will be regenerated)
truncate -s 0 /etc/machine-id
rm -f /var/lib/dbus/machine-id
ln -s /etc/machine-id /var/lib/dbus/machine-id
```

```
# Clear logs
find /var/log -type f -exec truncate -s 0 {} \;

# Clear bash history
cat /dev/null > ~/.bash_history && history -c
```

Step 6: Add Cloud-Init Drive

In the Proxmox web UI:

1. Select the VM
2. Go to **Hardware**
3. Click **Add > CloudInit Drive**
4. Select the storage for the cloud-init drive

Step 7: Convert to Template

In the Proxmox web UI:

1. Right-click the VM
2. Select **Convert to Template**

Template Configuration Tips

- **Use a unique VMID** for each template to avoid conflicts
- **Keep templates on shared storage** if you have a multi-node cluster, or use local storage with migration enabled in the module settings
- **Install the QEMU Guest Agent** (`qemu-guest-agent` package) for improved VM management and status reporting
- **Configure serial console** if you want noVNC console access to work properly
- **Minimize the template disk size** — the module can resize disks during deployment, but it cannot shrink them
- **Test the template** by manually cloning it and verifying that cloud-init applies the configuration correctly

Pre-built Templates

If you don't want to build templates from scratch, PUQcloud publishes two separate sets of ready-made VM templates as Proxmox backup archives (VMA format, `.vma.zst`):

1. **Prebuild Proxmox OS templates** — custom minimal installations built by PUQcloud from scratch.
2. **Official cloud images with root access** — upstream cloud images (Debian Cloud, Ubuntu Cloud, CentOS GenericCloud) modified so that root SSH login works out of the box.

Both sets are hosted under the same root folder:

- https://files.puqcloud.com/Proxmox_OS_Templates/

Download Prebuild Proxmox OS Templates

Custom PUQcloud builds — 5 GB `virtio` disk, no swap, minimal install, root SSH enabled, default password `puqcloud`, timezone Europe/Warsaw.

Debian

- Debian 10
- Debian 11
- Debian 12

Ubuntu

- Ubuntu 18
- Ubuntu 20
- Ubuntu 22

CentOS

- CentOS 7
- CentOS 8
- CentOS 9

Proxmox

- PBS 2.2

Official Cloud Images with Root Access

These are the **upstream cloud images** from Debian, Ubuntu and CentOS, modified by PUQcloud so that root SSH login is enabled out of the box. Use them if you prefer the official distribution builds over custom ones.

> Stock upstream cloud images disable root SSH login by default and create a non-root user (`debian`, `ubuntu`, `centos`). The module requires the `root` account to push cloud-init configuration and run management commands — that's why the "with root access" variants exist.

Debian

- Debian 10 — ~245 MB
- Debian 11 — ~275 MB
- Debian 12 — ~298 MB
- Debian 13 — ~307 MB

Ubuntu

- Ubuntu 20.04 — ~893 MB
- Ubuntu 22.04 — ~626 MB
- Ubuntu 23.10 — ~731 MB

CentOS

- CentOS 7 — ~1.01 GB
- CentOS 9 — ~1.10 GB

> File names may change over time. If a direct link returns 404, open the parent folder on files.puqcloud.com and grab the latest archive for the OS version you need.

Importing a Pre-built Template

1. Copy the `.vma.zst` file to your Proxmox node, for example into `/var/lib/vz/dump/`:

   ```
   cd /var/lib/vz/dump/
   wget https://files.puqcloud.com/Proxmox_OS_Templates/Debian/Debian-12/vzdump-qemu-1012-2024_03_23-17_11_11.vma.zst
   ```

2. Restore the backup to a new VMID (pick a free ID, e.g. `9012`) and target storage (e.g. `local-lvm`):

   ```
   qmrestore /var/lib/vz/dump/vzdump-qemu-1012-2024_03_23-17_11_11.vma.zst 9012 --storage local-lvm
   ```

   Alternatively, use the Proxmox web UI: **Datacenter > Storage > Backups > Restore**.
3. Open the restored VM, change the default root password (`puqcloud` for the prebuild set), verify the cloud-init drive is present, then right-click the VM and choose **Convert to Template**.

> **Disclaimer:** Your use of these operating systems is at your own risk. PUQcloud does not guarantee the correct operation or security of the pre-built templates or the root-access cloud images. Always review, update and harden them before offering to clients.

Supported Operating Systems

The module supports any operating system that Proxmox can run as a KVM virtual machine, provided it has working cloud-init (or cloudbase-init on Windows). Tested and known to work:

- **Linux**: Debian, Ubuntu, CentOS, AlmaLinux, Rocky Linux, openSUSE, Proxmox Backup Server
- **Windows**: Server / Desktop editions with `cloudbase-init` installed

Configuring Templates in WHMCS

Templates are selected in the product configuration under the **Module Settings** tab. You can also offer multiple templates to clients via Configurable Options, allowing them to choose their preferred operating system during order.

Install VNCproxy and noVNC

Proxmox KVM module **WHMCS**

Order now | Download | FAQ

Preface

The module supports the ability to connect to and use a browser-based console to manage a specific KVM virtual machine. To connect to the VM console we use third-party software.

noVNC — the open-source VNC client. noVNC is both a VNC client JavaScript library and an application built on top of that library. It runs well in any modern browser, including mobile browsers (iOS and Android).

- Project site: https://novnc.com
- Project GitHub: https://github.com/novnc/noVNC

> As we only use an external project, we do not take any responsibility for data leaks, hacks, etc.

The PUQ `vncwebproxy` binary itself is written in Go and uses the following libraries:

- go-vncproxy (MIT License)
- gin (MIT License)
- golang.org/x/net/websocket (BSD License)

How it works

The `vncwebproxy` sits between the client browser and your Proxmox server. It terminates the WebSocket from noVNC and forwards traffic to the Proxmox VNC port.

- The proxy must have stable network connectivity to the Proxmox server; TCP ports **5900–5999** to Proxmox are sufficient.
- If you use a **domain name** (not an IP) for the Proxmox server in the WHMCS server settings, that domain must resolve correctly **from the vncproxy host as well**.
- Each console session uses a one-time authentication ticket generated on demand and validated by the Proxmox API before the connection is established.
- All traffic between the client browser and the proxy is encrypted with SSL/TLS.

Public PUQcloud proxy (default)

If you have any difficulties setting up your own proxy, you can use the public PUQcloud vncproxy server. **However, we strongly recommend setting up and using your own vncproxy server** — this way you retain full control over performance and security.

Setting	Value
noVNC WEB proxy server	`vncproxy.puqcloud.com`
noVNC WEB proxy key	`puqcloud`
WEB ports	`80` / `443`
VNC ports	`5900–5999`

These values go into the WHMCS product settings under **Module Settings → Integrations Configuration**:

Setting	Description
noVNC Proxy Domain	The URL of your noVNC proxy (e.g. `https://vncproxy.puqcloud.com`)
noVNC Proxy Key	Authentication key configured on the proxy (e.g. `puqcloud`)

Installation process — your own VNCproxy server

The sections below describe the full installation of a dedicated vncproxy server. The example uses **Debian 11** and the domain `vncproxy.puqcloud.com` — in your own deployment, substitute your domain everywhere.

Step 1: Domain definition

First, choose a domain name for the vncproxy server (in our example: `vncproxy.puqcloud.com`). Create an `A`/`AAAA` record in your DNS pointing to the server's public IP address. Wait until the record propagates before requesting the SSL certificate.

Step 2: Prepare the server

Provision a VM or dedicated host with your favorite Linux distribution — the example uses **Debian 11**. Make sure the server can reach your Proxmox nodes on TCP ports `5900–5999`, and that inbound ports `80/443` are open for clients.

Update the package database:

```
sudo apt update
```

Install the NGINX web server, Certbot and `zip`:

```
sudo apt install certbot nginx python3-certbot-nginx zip -y
```

Step 3: Download the noVNC client

```
cd /root/
wget https://github.com/novnc/noVNC/archive/refs/tags/v1.3.0.zip
unzip v1.3.0.zip
cp -R noVNC-1.3.0/* /var/www/html/
rm v1.3.0.zip
rm -r noVNC-1.3.0/
```

After this step, opening `http://vncproxy.puqcloud.com/vnc.html` will load the noVNC client page.

Step 4: Generate an SSL certificate with Certbot

```
certbot --nginx -d vncproxy.puqcloud.com
```

To renew the certificate automatically, add a cron job:

```
crontab -e
```

```
0 12 * * * /usr/bin/certbot renew --quiet
```

Step 5: NGINX virtual host configuration

Edit the default site configuration:

```
nano /etc/nginx/sites-available/default
```

Use the following config — remember to replace `vncproxy.puqcloud.com` with your own domain:

```
server {
    listen 80 default_server;
    listen [::]:80 default_server;

    root /var/www/html;

    index index.html index.htm index.nginx-debian.html;

    server_name _;

    location / {
        try_files $uri $uri/ =404;
    }
}

server {

    root /var/www/html;

    index index.html index.htm index.nginx-debian.html;
    server_name vncproxy.puqcloud.com; # managed by Certbot

    location / {
        try_files $uri $uri/ =404;
    }
```

```
    listen [::]:443 ssl ipv6only=on; # managed by Certbot
    listen 443 ssl; # managed by Certbot
    ssl_certificate /etc/letsencrypt/live/vncproxy.puqcloud.com/fullchain.pem; # managed by
Certbot
    ssl_certificate_key /etc/letsencrypt/live/vncproxy.puqcloud.com/privkey.pem; # managed by
Certbot
    include /etc/letsencrypt/options-ssl-nginx.conf; # managed by Certbot
    ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem; # managed by Certbot

    location /vncproxy {
        proxy_pass http://127.0.0.1:8080/vncproxy;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "Upgrade";
        proxy_set_header Host $host;
        proxy_set_header    X-Real-IP        $remote_addr;
        proxy_set_header    X-Forwarded-For  $proxy_add_x_forwarded_for;
    }
}

server {
    if ($host = vncproxy.puqcloud.com) {
        return 301 https://$host$request_uri;
    } # managed by Certbot

    listen 80 ;
    listen [::]:80 ;
    server_name vncproxy.puqcloud.com;
    return 404; # managed by Certbot
}
```

Reload NGINX:

```
service nginx restart
```

Step 6: Install the `vncwebproxy` binary

Download the PUQ `vncwebproxy` binary from the official download server and make it executable:

```
```

```
apt-get install screen -y
cd /root/
wget https://download.puqcloud.com/WHMCS/servers/PUQ_WHMCS-Proxmox-KVM/vncproxy/vncwebproxy
chmod +x vncwebproxy
```

Step 7: Run the proxy

Run the script inside a `screen` session so it keeps running in the background. The **first argument** is a unique key — this is exactly the value you will later put into the **noVNC Proxy Key** field in the WHMCS module.

```
screen
./vncwebproxy puqcloud
```

After a successful launch you can watch the request log directly in the console:

```
root@vncproxy:~# ./vncwebproxy puqcloud
[./vncwebproxy puqcloud]
proxmox-test.uuq.pl59002022/09/11 19:11:08 [vncproxy][debug] ServeWS
2022/09/11 19:11:08 [vncproxy][debug] request url: /vncproxy/proxmox-
test.uuq.pl/5900/d91bac199c2ce79392d8e175076e3780
2022/09/11 19:11:13 [vncproxy][info] close peer
[GIN] 2022/09/11 - 19:11:13 | 200 | 4.740249024s | 79.184.10.217 | GET
"/vncproxy/proxmox-test.uuq.pl/5900/d91bac199c2ce79392d8e175076e3780"
```

Detach from `screen` with `Ctrl+A` then `D`. Reattach later with `screen -r`.

Step 8: Configure WHMCS

In the WHMCS product settings, under **Module Settings → Integrations Configuration**, fill in:

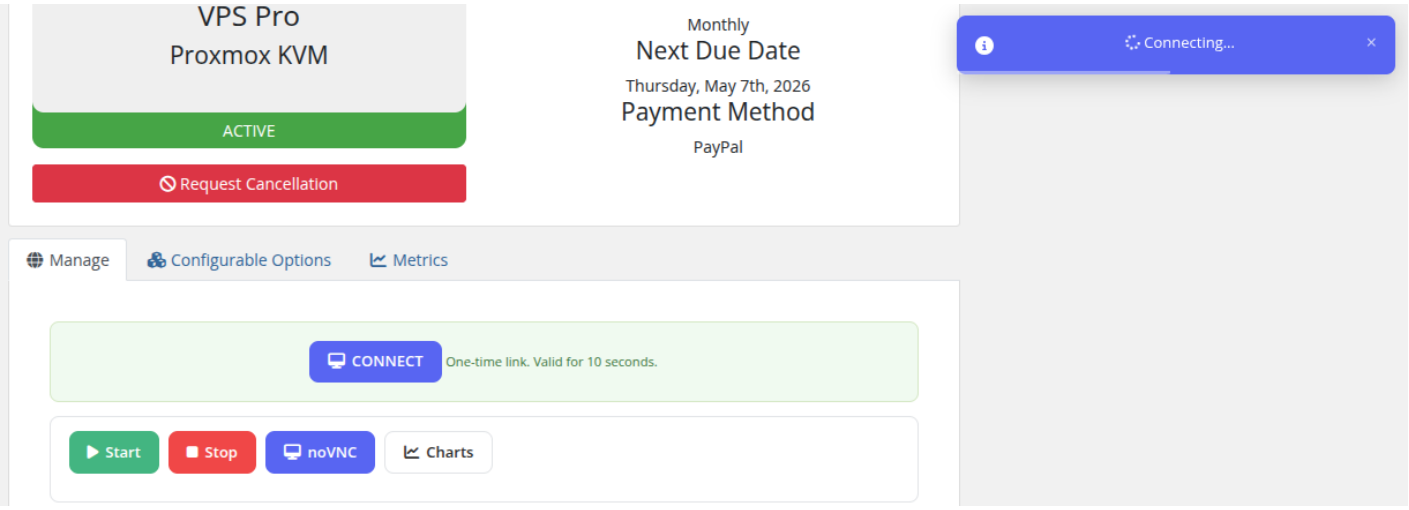
- **noVNC Proxy Domain** → `https://vncproxy.your-domain.tld`
- **noVNC Proxy Key** → the key you passed to `./vncwebproxy` (in our example: `puqcloud`)

Save the product and try opening the console from the client area.

Client Access

When noVNC is configured, clients see a **Console** button in their VM management area. Clicking it

opens a new browser window with the noVNC console, providing full keyboard and mouse access to the virtual machine.



Security

The security configuration of the vncproxy server should meet your own standards. A few mandatory points:

- Allow inbound TCP **80/443** from the internet (clients need HTTPS access to noVNC).
- Allow outbound TCP **5900–5999** from the vncproxy host to your Proxmox nodes.
- Keep the OS, NGINX and the `vncwebproxy` binary up to date.
- Each console session uses a **one-time ticket** — tickets are generated on demand, expire after a short period, and are validated against the Proxmox API before the connection is established.
- All traffic between the client browser and the proxy is encrypted via SSL/TLS (Let's Encrypt certificate).

Do not forget that for correct operation you must allow HTTPS to the proxy and outgoing connections from the proxy to the Proxmox server.

Email Templates

Proxmox KVM module **WHMCS**

Order now | Download | FAQ

Overview

The module uses five email templates to notify clients about VM lifecycle events. These templates must be created manually in WHMCS before the module can send the corresponding notifications.

Creating Email Templates in WHMCS

1. Navigate to **Setup > Email Templates** in the WHMCS admin area
2. Click **Create New Email Template**
3. Set the **Type** to **Product/Service**
4. Set the **Unique Name** to the exact template name listed below
5. Fill in the subject line and email body using the available variables
6. Click **Save Changes**

Repeat for all five templates.

> **Important:** The template unique names must match exactly as listed below. The module looks up templates by their unique name, so any mismatch will prevent the email from being sent.

Email Templates

1. puqProxmoxVKM Welcome Email

> **Note the spelling.** The original template name is literally `puqProxmoxVKM Welcome Email` (with `VKM`, not `KVM`). It has stayed like this since v1.0 for backwards compatibility — if you rename it the module will stop sending the welcome email.

Sent to the client as an order confirmation when a new Proxmox KVM service is created. At this point the VM is not yet ready — this email just tells the client their order has been accepted.

When sent: Upon service creation / order acceptance, before the deploy pipeline runs.

Unique Name: `puqProxmoxVKM Welcome Email` **Email Type:** Product/Service **Subject:** `Virtual Machine Order Information`

Body:

```
Dear {$client_name},

Your order has been accepted for implementation.
Installing and pre-configuring the virtual machine will take some time.
Please wait for an e-mail with information that the virtual machine is ready for use, also
with access parameters.

Product/Service: {$service_product_name}
Payment Method: {$service_payment_method}
Amount: {$service_recurring_amount}
Billing Cycle: {$service_billing_cycle}
Next Due Date: {$service_next_due_date}

Important note - if you have also purchased the backup options, do not forget to configure
the schedule in the service's subpage.

Thank you for choosing us.
```

```
{$signature}
```

---

2. puqProxmoxKVM VM is ready

Sent to the client when a new virtual machine has been successfully deployed and is ready for use. **This is the most important template** — it contains the VM access credentials (IP, username, password).

When sent: After the deploy pipeline reaches the `ready` state and the VM is confirmed running.

Unique Name: `puqProxmoxKVM VM is ready` **Email Type:** Product/Service **Subject:** `Virtual Machine is ready`

Body:

```
Dear {$client_name},

Your virtual machine is already ready.
You can connect to it using data.

Product/Service: {$service_product_name}
Payment Method: {$service_payment_method}
Amount: {$service_recurring_amount}
Billing Cycle: {$service_billing_cycle}
Next Due Date: {$service_next_due_date}

IP address: {$service_dedicated_ip} or {$service_domain}
Username: {$service_username}
Password: {$service_password}

Thank you for choosing us.

{$signature}
```

---

3. puqProxmoxKVM Reset password

Sent to the client after a password reset operation on their virtual machine.

When sent: After a password reset is performed via the admin area or client area — once cloud-

init has re-applied the new credentials and the VM has been restarted.

Unique Name: `puqProxmoxKVM Reset password` **Email Type:** Product/Service **Subject:** `Reset password is ready`

Body:

```
Dear {$client_name},

Password reset successful.

IP address: {$service_dedicated_ip} or {$service_domain}
Username: {$service_username}
Password: {$service_password}

Thank you for choosing us.

{$signature}
```

---

4. puqProxmoxKVM Backup restored

Sent to the client after a backup has been successfully restored.

When sent: After a backup restore operation completes and the module has re-applied CPU/RAM/disk/network settings and restarted the VM.

Unique Name: `puqProxmoxKVM Backup restored` **Email Type:** Product/Service **Subject:** `Backup restored successful`

Body:

```
Dear {$client_name},

Backup restored successful.

Product/Service: {$service_product_name}
Payment Method: {$service_payment_method}
Amount: {$service_recurring_amount}
Billing Cycle: {$service_billing_cycle}
Next Due Date: {$service_next_due_date}
```

```
IP address: {$service_dedicated_ip} or {$service_domain}

Thank you for choosing us.

{$signature}
```

---

5. puqProxmoxKVM Upgrade Email

Sent to the client after a package upgrade or downgrade completes.

When sent: After the `change_package` state machine finishes applying the new product parameters to the VM and starts it back up.

Unique Name: `puqProxmoxKVM Upgrade Email` **Email Type:** Product/Service **Subject:** `Virtual Machine upgrade is ready`

Body:

```
Dear {$client_name},

Virtual Machine upgrade is successful.

Product/Service: {$service_product_name}
Payment Method: {$service_payment_method}
Amount: {$service_recurring_amount}
Billing Cycle: {$service_billing_cycle}
Next Due Date: {$service_next_due_date}

IP address: {$service_dedicated_ip} or {$service_domain}

Thank you for choosing us.

{$signature}
```

> “

> The bodies above are the original PUQcloud templates shipped with v1.0 through v3.0. You are free to customize subjects and bodies to match your brand — only the **Unique Name** and **Email Type** must stay as documented, because the module looks up templates by their unique name.

Available Template Variables

The following merge fields are available in all email templates:

Standard WHMCS Variables

Variable	Description
`{$client_name}`	Client's full name
`{$service_product_name}`	Product/service name
`{$service_dedicated_ip}`	Primary IPv4 address assigned to the VM
`{$service_domain}`	Service domain / hostname
`{$service_username}`	Operating system username
`{$service_password}`	Operating system password
`{$service_recurring_amount}`	Recurring billing amount
`{$service_billing_cycle}`	Billing cycle (Monthly, Quarterly, etc.)
`{$service_next_due_date}`	Next payment due date
`{$signature}`	Email signature configured in WHMCS

Module-Specific Variables

Variable	Description
`{$service_assigned_ips}`	All assigned IP addresses
`{$vm_id}`	Proxmox virtual machine ID (VMID)
`{$server_hostname}`	Proxmox server hostname

In addition, all standard WHMCS client and service merge fields are available.

Cron Configuration

Proxmox KVM module **WHMCS**

Order now | Download | FAQ

Overview

The module requires a cron system to process VM deployments, package changes, backups, and other automated tasks. Two cron modes are available, and you can choose the one that best fits your environment.

Cron Modes

Mode 1: WHMCS Hook (Default)

In this mode, the module hooks into the standard WHMCS cron and executes its tasks automatically each time the WHMCS cron runs.

Advantages:

- No additional configuration required
- Works out of the box after module activation
- Uses the existing WHMCS cron schedule

When to use: This is the recommended mode for most installations. If your WHMCS cron runs every 5 minutes (the standard recommendation), this provides timely task execution.

No additional crontab entries are needed. Just ensure the standard WHMCS cron is running:

```
*/5 * * * * php -q /path/to/whmcs/cron/cron.php
```

Mode 2: Standalone

In this mode, the module's cron runs independently from the WHMCS cron via a separate crontab entry. This gives you independent control over the module's cron frequency.

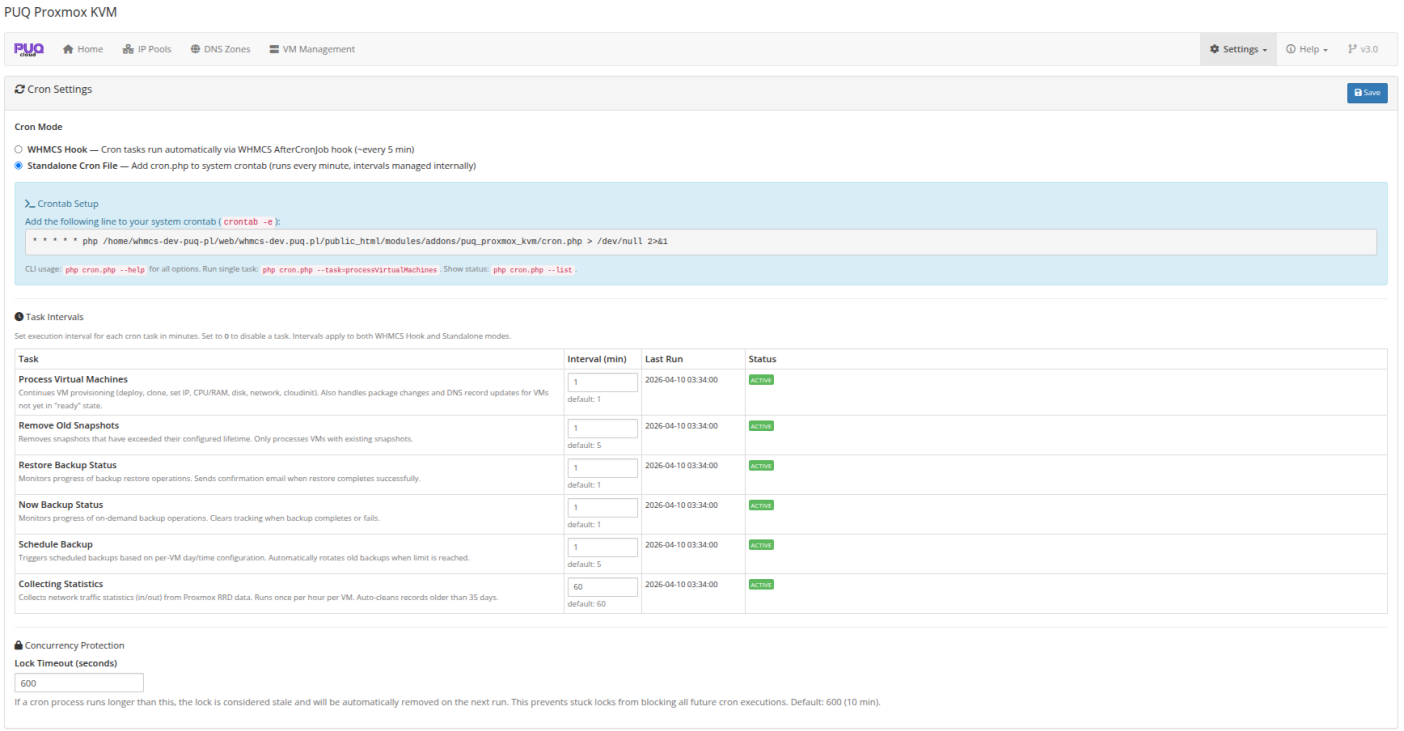
Advantages:

- Independent schedule from WHMCS cron
- Can run more frequently for faster VM provisioning
- Useful if your WHMCS cron runs less frequently

When to use: Use standalone mode if you need the module to process tasks more frequently than your WHMCS cron runs, or if you want to separate the module's workload from the main WHMCS cron.

To set up standalone cron, add the following to your server's crontab:

```
*/5 * * * * php -q /path/to/whmcs/modules/addons/puq_proxmox_kvm/cron.php
```

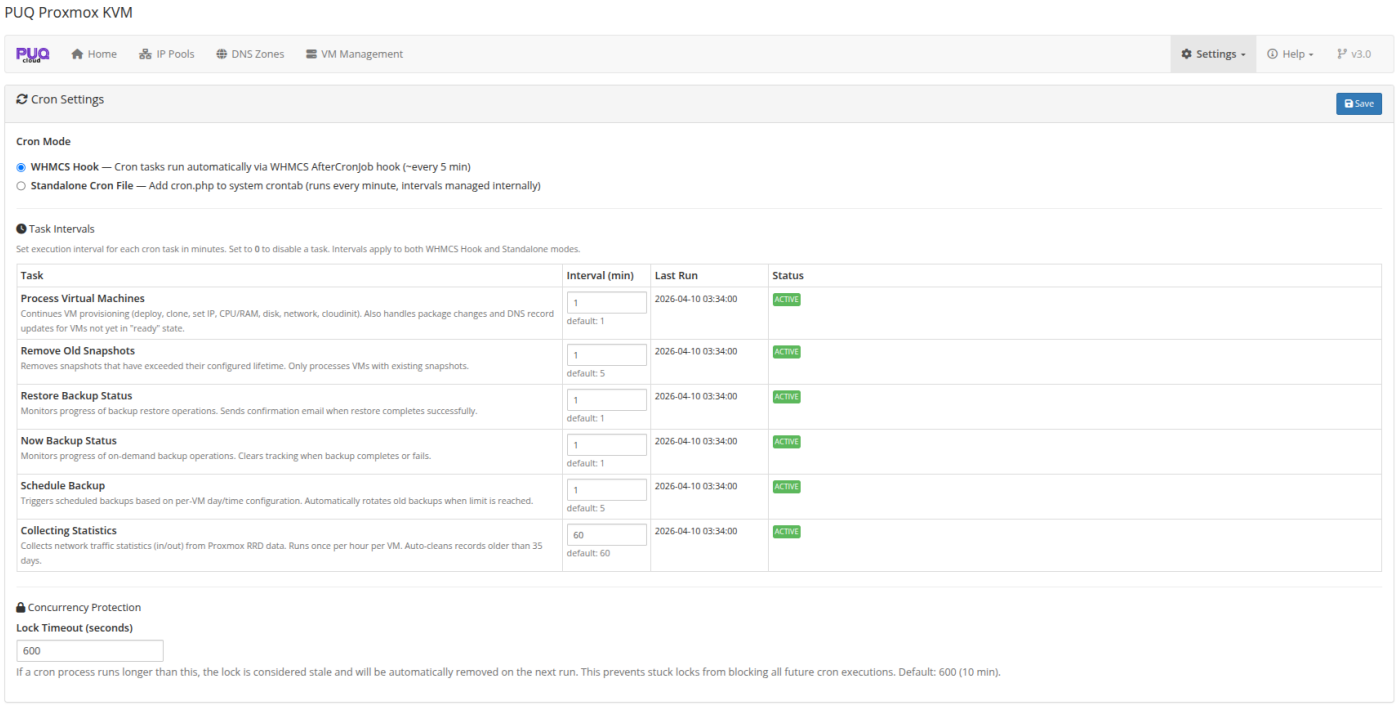


Configuring Cron Mode

The cron mode is configured in the addon settings:

1. Navigate to **Addons > PUQ Proxmox KVM**

2. Go to the **Settings** page
3. Select the **Cron** tab
4. Choose your preferred cron mode: **WHMCS Hook** or **Standalone**
5. Save settings



Task Intervals

Each cron task has a configurable interval that controls how often it runs. These intervals can be adjusted in the Cron settings page. For details on individual tasks and their intervals, see the Cron and Automation section.

Verifying Cron Operation

To confirm the cron is running correctly:

1. Navigate to the addon **Settings > Cron** page
2. Check the **Last Run** timestamp for each task
3. Verify there are no stale lock files

If tasks are not executing, check:

- The WHMCS cron is running (for Hook mode)
- The standalone crontab entry is correct (for Standalone mode)
- PHP CLI is available at the path specified in the crontab

- File permissions allow the cron script to execute

Migration from PUQ Customization

Overview

Version 3.0 introduces a standalone addon module (`puq_proxmox_kvm`) that replaces the PUQ Customization extension (`ModulePuqProxmoxKVM`).

Migration Steps

1. Install New Modules

Upload both modules to your WHMCS installation:

- `modules/servers/puqProxmoxKVM/` (updated server module v3.0)
- `modules/addons/puq_proxmox_kvm/` (new addon module)

2. Activate Addon

1. Go to **Setup > Addon Modules**
2. Activate **PUQ Proxmox KVM**
3. Enter your license key

3. Automatic Data Migration

On activation, the addon automatically:

- Creates new database tables (`puq_proxmox_kvm_ip_pools`, `puq_proxmox_kvm_dns_zones`)
- Detects old tables (`puq_customization_module_puq_proxmox_kvm_ip_pools`, `puq_customization_module_puq_proxmox_kvm_dns_zones`)
- Copies all data from old tables to new tables (if new tables are empty)

4. Verify Migration

1. Open **Addons > PUQ Proxmox KVM**
2. Check that all IP Pools are present
3. Check that all DNS Zones are present
4. Verify Services Summary shows correct data

5. Deactivate Old Extension

Once verified:

1. Go to PUQ Customization addon
2. Deactivate the ModulePuqProxmoxKVM extension

> **Important:** The server module v3.0 supports both the new and old addon simultaneously, so services will continue working during migration.

Backward Compatibility

The updated server module (v3.0) uses a dual-path approach:

1. First checks for the new standalone addon (`puq_proxmox_kvm`)
2. Falls back to the old PUQ Customization extension if new addon is not found

This ensures zero downtime during the migration process.