

Webunit

- [Was ist Webunit](#)
- [Anleitungen](#)
 - [Hintergrund-Prozesse \(Supervisor\)](#)
 - [PHP-Tuning \(Custom-INI\)](#)
 - [Protokollierung / Logging](#)
 - [MariaDB-Tuning \(Custom-Config\)](#)
 - [CLI-Befehle im Container](#)
 - [Passwoerter selbst aendern](#)

Was ist Webunit

Webunit ist eine schlanke, Docker-basierte Webhosting-Plattform. Sie richtet sich an Agenturen, Reseller und technisch versierte Betreiber, die mehrere Websites, Shops oder Anwendungen auf eigenen Servern hosten wollen — ohne die Komplexitaet, die Sicherheitsbedenken und den Ressourcen hunger klassischer Panels wie Plesk oder cPanel.

Der Grundgedanke

Jede Website (bei uns **Subscription** genannt) laeuft in ihrem eigenen Container. Das gibt saubere Trennung: eine kompromittierte Site kann keine andere angreifen, PHP-Versionen und Extensions werden pro Sub konfiguriert, und Ressourcen (RAM, CPU) sind pro Container hart begrenzt und weich reserviert (Plesk-Modell). Ausfall einer Site zieht keine andere mit.

Der Kunde bekommt SSH-/SFTP-Zugang zu seinem Container, eine eigene MariaDB und optional phpMyAdmin. Er kann per CLI-Befehlen (z.B. `webunit-password ssh`, `pma-link`) selbstaendig Dinge tun, fuer die er sonst ein Ticket schreiben muesste.

Warum ich das entwickelt habe

Ich betreibe seit Jahren Webhosting fuer Kunden. Die grossen Panels sind entweder teuer, ueberladen oder beides — und sie geben mir zu wenig Kontrolle darueber, was auf meinen Servern laeuft. Genauso wichtig war mir das Thema **Sicherheit**: bei klassischem Shared-Hosting laufen alle Kunden auf demselben System, unter denselben Kernel-Prozessen, mit denselben PHP-Bibliotheken. Eine Sicherheitsluecke in einem Kunden-Skript kann im schlimmsten Fall andere Kunden treffen, jedes Panel-Update ist ein Risiko fuer alle gleichzeitig, und ein einzelner kompromittierter Zugang wird schnell zum Problem fuer die ganze Node.

Webunit ist mein Versuch, mit deutlich weniger Code und weniger Komplexitaet denselben Job zu machen — mit sauberer Container-Isolation pro Kunde als Grundprinzip, und nur den Features drin, die ich tatsaechlich brauche.

Ziele beim Bau:

- **Container-Isolation** statt geteilter Nutzer-Accounts auf einem System — keine Cross-Site-Angriffe, keine PHP-Versionskaempfe.
- **Verteilung ueber mehrere Nodes** (VMs, Bare-Metal, was auch immer), zentral gesteuert von einem Admin-Panel. Migration einer Sub zwischen Nodes ohne DNS-Chaos.
- **Kunden-Self-Service** wo sinnvoll: Passwoerter selbst rotieren, SSH-Keys selbst hinterlegen, PMA-Zugang selbst holen, ohne dass wir ins Panel muessen.

- **Klare, spartanische UI** fuer den Betreiber. Kein Feature-Bloat.
- **Automatisierung als Default:** Backups, Cert-Renewals, Image-Rebuilds, Kapazitaets-Checks laufen ohne dass jemand dran denken muss.
- **Datensouveraenitaet fuer Reseller und Agenturen:** wer Webunit einsetzt, kann seine **eigenen Web-Nodes** anbinden (bei welchem Cloud- oder Bare-Metal-Anbieter auch immer), seine **eigenen SMTP-Smarthosts** fuer den Mail-Ausgang nutzen und seine **eigenen S3-Buckets** fuer die Backups. Die Steuerungs-Ebene laeuft zentral, die Kunden-Daten liegen aber auf der eigenen Infrastruktur — kein Vendor-Lock-in, keine Speicherung wichtiger Assets bei uns.

Was drin ist

- **Subscriptions** mit PHP 8.x, MariaDB, Traefik-basierten HTTPS-Zertifikaten (Let's Encrypt oder self-signed pro Hostname waehlbar).
- **Mail** ueber angebundene Postal- oder WMN-Server (unser eigenes Setup), mit DKIM-Verwaltung und pro-Sub-Routing.
- **Restic-Backups** pro Sub und pro Node, verschluesselt, mit automatischer Rotation.
- **Ressourcen-Reservation** im Plesk-Modell — Nodes lassen sich bewusst ueberprovisionieren, Kapazitaets-Uebersicht im Panel.
- **CLI im Kunden-Container:** `webunit-password`, `pma-link`, `webunit-supervisor` (Hintergrund-Prozesse), `webunit-ssh` (Auth-Info).
- **Wiki** direkt integriert (das hier), plus Kunden-Anleitungen und Entwickler-Doku.

Fuer wen es nicht ist

Webunit ist bewusst kein Massen-Panel. Wer ein Klick-und-fertig-Hosting sucht mit Chat-Support und tausend eingebauten One-Click-Installern, ist bei einem anderen Anbieter besser aufgehoben. Wer aber die volle Kontrolle ueber seine Server behalten will, technisch fit ist und lieber Container als PHP-Bibliotheken managt, findet hier ein schlankes, verstaendliches System.

Anleitungen

Hintergrund-Prozesse (Supervisor)

Hintergrund-Prozesse via Supervisor

Wenn dein Projekt einen Prozess braucht, der **dauerhaft im Hintergrund laeuft** (z.B. Symfony Messenger, Laravel Queue-Worker, ein Node.js-Daemon, ein eigener Watcher), kannst du ihn per Supervisor starten lassen. Der Prozess startet dann automatisch mit dem Container und wird bei Absturz neu gestartet — du musst nichts weiter tun.

“ **Cron vs. Supervisor** — Cron startet einen Prozess **zu festen Zeitpunkten** und laesst ihn nach getaner Arbeit sterben. Supervisor startet einen Prozess **einmal** und haelt ihn **dauerhaft am Leben**. Fuer Job-Queues (Messenger, Sidekiq, RQ etc.) willst du Supervisor.

Wo liegen die Config-Dateien?

Per SSH in deinen Container einloggen und dann:

```
cd ~/.webunit/supervisor.d/
```

Dieses Verzeichnis gehoert dir und ist vor Web-Zugriff geschuetzt (`https://deine-domain/.webunit/...` liefert 403).

Pro Prozess legst du eine `<name>.conf`-Datei an. Der Dateiname ist egal, die Endung `.conf` ist Pflicht.

Erstes Beispiel: Symfony Messenger

Datei: `~/.webunit/supervisor.d/messenger.conf`

```
[program:messenger-consume]
command=/usr/local/bin/php /var/www/current/bin/console messenger:consume async --time-limit=3600 --memory-limit=256M
```

```
autostart=true
autorestart=true
stdout_logfile=~/.webunit/logs/messenger.log
stderr_logfile=~/.webunit/logs/messenger.err
```

Aktivieren:

```
webunit-supervisor reload
```

Status pruefen:

```
webunit-supervisor status
```

Ausgabe:

apache2	RUNNING	pid 56, uptime 0:01:38
sshd	RUNNING	pid 57, uptime 0:01:38
messenger-consume	RUNNING	pid 129, uptime 0:00:05

Fertig — der Worker laeuft, wird bei Container-Restart automatisch gestartet und bei Absturz neu gestartet.

Weitere Beispiele

Laravel Queue-Worker

```
[program:laravel-queue]
command=/usr/local/bin/php /var/www/artisan queue:work --sleep=3 --tries=3 --max-time=3600
autostart=true
autorestart=true
stdout_logfile=~/.webunit/logs/queue.log
stderr_logfile=~/.webunit/logs/queue.err
```

Node.js-Daemon

```
[program:my-node-daemon]
command=/usr/bin/node /var/www/current/daemon.js
```

```
directory=/var/www/current
environment=NODE_ENV="production",PORT="4000"
autostart=true
autorestart=true
stdout_logfile=~/.webunit/logs/node.log
stderr_logfile=~/.webunit/logs/node.err
```

Mehrere Worker parallel

Manche Queue-Systeme profitieren von mehreren parallelen Workern. Supervisor kennt dafür `numprocs`:

```
[program:messenger-consume]
command=/usr/local/bin/php /var/www/current/bin/console messenger:consume async --time-limit=3600
numprocs=3
process_name=%(program_name)s_%(process_num)02d
autostart=true
autorestart=true
stdout_logfile=~/.webunit/logs/messenger_%(process_num)02d.log
stderr_logfile=~/.webunit/logs/messenger_%(process_num)02d.err
```

Das startet 3 Instanzen: `messenger-consume_00`, `_01`, `_02`.

Einfacher Bash-Loop (zum Testen)

```
[program:testworker]
command=/bin/bash -c "while true; do date; sleep 5; done"
autostart=true
autorestart=true
stdout_logfile=~/.webunit/logs/testworker.log
stderr_logfile=~/.webunit/logs/testworker.err
```

Kommandos

Alle Kommandos rufst du im Container-SSH ohne `sudo` auf.

Kommando	Was es macht
----------	--------------

<code>webunit-supervisor reload</code>	Liest <code>*.conf</code> neu ein und startet neue/geaenderte Programme. Sanitize-Check laeuft vorher.
<code>webunit-supervisor status</code>	Zeigt Status aller Programme (deine + apache2 + sshd).
<code>webunit-supervisor restart <name></code>	Startet ein Programm neu.
<code>webunit-supervisor tail <name></code>	Zeigt letzte Zeilen von stdout.
<code>webunit-supervisor tail-err <name></code>	Zeigt letzte Zeilen von stderr.
<code>webunit-supervisor validate</code>	Prueft nur die Configs, ohne Reload.
<code>webunit-supervisor help</code>	Alle Kommandos.

“ `restart` und `tail` auf `apache2` oder `sshd` werden abgelehnt — das sind System-Prozesse.

Config-Format (Kurz-Referenz)

Wichtige Direktiven fuer `[program:...]`:

Direktive	Bedeutung
<code>command=</code>	Vollstaendiger Pfad zum Binary + Argumente. Kein <code>sudo</code> .
<code>directory=</code>	Arbeitsverzeichnis vor dem Start.
<code>environment=KEY="value",KEY2="value2"</code>	Zusaetzliche Umgebungsvariablen.
<code>autostart=true</code>	Beim Container-Start automatisch starten.
<code>autorestart=true</code>	Nach Absturz neu starten.
<code>startsecs=1</code>	Sekunden Wartezeit, bis der Start als erfolgreich gilt.
<code>startretries=3</code>	Nach so vielen Fehl-Starts aufgeben und <code>FATAL</code> markieren.
<code>stopsignal=TERM</code>	Signal zum Stoppen (<code>TERM</code> , <code>INT</code> , <code>KILL</code>).
<code>stopwaitsecs=10</code>	Wartezeit auf sauberes Beenden, bevor <code>KILL</code> .
<code>stdout_logfile=<pfad></code>	Datei fuer stdout. Standard sind Dateien unter <code>~/.webunit/logs/</code> .
<code>stderr_logfile=<pfad></code>	Datei fuer stderr.
<code>stdout_logfile_maxbytes=50MB</code>	Rotate nach 50 MB (Standard).
<code>stdout_logfile_backups=10</code>	Anzahl rotierter Backups (Standard 10).
<code>numprocs=N</code>	N parallele Instanzen des Programms.

Direktive	Bedeutung
<code>process_name=%(program_name)s_%(process_num)02d</code>	Namensschema bei numprocs>1.

Vollstaendige Referenz: <http://supervisord.org/configuration.html#program-x-section-values>

Was NICHT erlaubt ist

Der Sanitize-Check beim `reload` blockt:

- `user=root` oder `user=0` — Programme laufen als Container-User.
- `umask=...` — greift zu tief in Prozess-Umgebung ein.
- `stopasuser=root` oder `stopasuser=0` — analog.
- Programm-Namen `apache2`, `sshd`, `supervisord` — reserviert fuer System-Prozesse.
- Andere Sektionen als `[program:...]`, `[group:...]` oder `[eventlistener:...]` — die anderen (z.B. `[supervisord]`) wuerden die globale Container-Config beeinflussen.

Bei Fehlern zeigt `webunit-supervisor validate` (oder `reload`) genau welche Zeile in welcher Datei das Problem ist:

```
FEHLER: ~/.webunit/supervisor.d/messenger.conf:3 - user=root ist nicht erlaubt (nur Container-User)
```

Logs

- **stdout / stderr** deiner Prozesse landen in den Dateien, die du in der Config angibst.
Empfehlung: `~/.webunit/logs/<name>.log`.
- **Rotation:** Supervisor rotiert automatisch bei 50 MB, haelt 10 Backups. Aendern via `stdout_logfile_maxbytes=` und `stdout_logfile_backups=`.
- **Live mitlesen:** `webunit-supervisor tail <name>` (stdout) bzw. `tail-err` (stderr) — zeigt die letzten ~1000 Zeilen.
- **Selbst gucken:** `tail -f ~/.webunit/logs/messenger.log`.
- **Web-Zugriff geblockt:** `/.webunit/*` liefert immer 403, deine Logs sind also nicht von aussen einsehbar.

Was passiert bei Container-Neubau?

Deine `~/webunit/supervisor.d/`-Configs bleiben erhalten (liegen im `/var/www`-Volume). Nach dem Neubau starten sie automatisch mit dem Container.

Die generierte `README.txt` im Verzeichnis kommt nur **einmal** beim allerersten Container-Start rein — wenn du sie loeschst, wird sie nicht neu geschrieben.

Troubleshooting

Prozess ist im Status `FATAL`

`webunit-supervisor status` zeigt `FATAL`. Das heisst Supervisor hat den Prozess mehrfach versucht zu starten und aufgegeben. Ursache findest du in `stderr`:

```
webunit-supervisor tail-err <name>
```

Typische Fehler:

- Falscher Pfad im `command=`
- Fehlende Berechtigungen auf einer Datei
- Datenbank nicht erreichbar
- Node-Modules / vendor/ nicht installiert

Fix, dann:

```
webunit-supervisor restart <name>
```

Prozess ist im Status `EXITED`

Er ist normal beendet (Exit-Code 0), Supervisor startet ihn aber nicht neu. Vermutlich hast du `autorestart=false` oder der Command endet tatsaechlich (z.B. `time-limit` bei Messenger). Bei Messenger ist das gewollt — `autorestart=true` startet ihn dann wieder.

`webunit-supervisor: command not found`

Das kann eigentlich nicht passieren — das Binary liegt im PATH. Wenn doch, ist entweder dein Container aelter als 2026-09-03 (dann sag uns Bescheid, wir rebuilden) oder der PATH ist in deinem `.bashrc` ueberschrieben.

Config aendert sich nicht

Nach dem Editieren einer `.conf` musst du **explizit** `reload` fahren. Supervisor sieht Datei-Änderungen nicht von allein.

```
webunit-supervisor reload
```

"Sanitize-Check fehlgeschlagen — Reload abgebrochen"

Der Reload wird abgebrochen wenn **irgendeine** deiner `.conf` einen Fehler hat. Fix die betroffene Datei (die genaue Zeile steht in der Fehlermeldung), dann nochmal `reload`.

PHP-Tuning (Custom-INI)

PHP-Tuning per Custom-INI

Die haeufigsten PHP-Werte (Memory, OPcache, APCu) sind pro Hostname konfigurierbar. Manche haben eigene Formularfelder, den Rest setzt du im **Custom-INI**-Textblock.

Wo?

Admin-Panel → Subscription → **Bearbeiten** → Tab **Hostnames** → beim gewuenschten Hostname aufklappen.

Direkte Formularfelder (kein Custom-INI noetig):

- **PHP Memory Limit** (`memory_limit`, z.B. `1024M`)
- **Upload Max Filesize** (`upload_max_filesize`)
- **Post Max Size** (`post_max_size`)
- **Max Execution Time** (`max_execution_time`)
- **Max Input Vars** (`max_input_vars`)

Alles andere → **Custom-INI**-Textblock.

Copy-&-Paste-Snippets

Grosse Symfony/Shopware/Laravel-Anwendung

```
memory_limit = 1024M
opcache.memory_consumption = 256
opcache.max_accelerated_files = 20000
opcache.revalidate_freq = 0
opcache.validate_timestamps = 1
apcu.shm_size = 128M
realpath_cache_size = 4096K
realpath_cache_ttl = 600
```

WordPress mit vielen Plugins

```
memory_limit = 512M
opcache.memory_consumption = 192
opcache.max_accelerated_files = 15000
apcu.shm_size = 64M
```

Produktion (Deploy-basiert, kein File-Watcher noetig)

```
opcache.validate_timestamps = 0
opcache.enable_cli = 1
```

“ validate_timestamps = 0 heisst: nach Code-Aenderungen musst du PHP neu laden (docker compose exec app apache2ctl graceful — oder wir exponieren dafuer noch einen Button im Admin). Bringt spuerbar Performance.

Sessions in APCu statt Dateien (kleine Latenz, kein Disk-IO)

```
session.save_handler = files
session.save_path = /tmp
; ODER (schneller, braucht apcu):
; session.save_handler = apcu
```

Whitelist — was ist erlaubt?

Der Custom-INI-Block wird beim Speichern gefiltert. Erlaubt sind unter anderem:

- memory_limit, max_execution_time, max_input_time
- post_max_size, upload_max_filesize, max_input_vars, max_file_uploads
- date.timezone, default_charset, intl.default_locale
- display_errors, display_startup_errors, log_errors, error_reporting
- session.gc_maxlifetime, session.cookie_lifetime, session.cookie_secure, session.cookie_httponly, session.cookie_samesite
- default_socket_timeout, realpath_cache_size, realpath_cache_ttl

- `output_buffering`, `output_compression`
- `mail.add_x_header`
- `opcache.*` — alle OPcache-Einstellungen
- `apcu.*` — alle APCu-Einstellungen
- `gd.*` — GD-Bibliothek

Alles andere wird verworfen (mit Hinweis im Speicher-Feedback).

Was kann ich abfragen?

`phpinfo()` in einem PHP-File — zeigt alle aktiven Werte. Aber:

“ **Achtung:** `phpinfo()`-Ausgabe **niemals oeffentlich** stehen lassen (nur kurz aufrufen, dann sofort loeschen oder mit `.htaccess` blocken). Sie enthaelt viele interne Umgebungs-Details.

Protokollierung / Logging

Protokollierung / Logging

Was wird protokolliert, wo landen die Logs, und was ist geplant.

Aktueller Stand

Logs im Container

Jeder Sub-Container schreibt lokal:

Log	Pfad im Container	Inhalt	Persistent?
Apache Access	<code>/var/log/apache2/<vhost>-access.log</code> (auch per <code>~/logs/</code> erreichbar)	HTTP-Requests pro vhost	Ja — Bind-Mount + im Restic-Backup
Apache Error	<code>/var/log/apache2/<vhost>-error.log</code>	Apache-Fehler + PHP-Fehler (via <code>log_errors = On</code>)	Ja — Bind-Mount + im Restic-Backup
PHP Errors	in Apache Error (nicht separat)	fatal/warning/notice — steuerbar per <code>error_reporting</code> in Custom-INI	siehe Apache Error
msmtp	<code>/var/log/msmtp.log</code>	ausgehende Mails (Sender, Empfaenger, Ergebnis)	Ja — Bind-Mount + im Restic-Backup
Supervisor-Kunden-Prozesse	<code>~/webunit/logs/*.log</code>	stdout/stderr eigener [program:...]-Configs	Ja — Home-Volume + im Restic-Backup
Container stdout/stderr	via <code>docker logs</code>	Apache-Startup + Supervisor-Events + PHP <code>error_log()</code> ohne Datei	Nein — Docker-Default (10 MB, 3 Files), Debug-only
SSH-Logins in den Container	<code>/var/log/auth.log</code> (Container-intern)	SSH-Sessions dieses Kunden	Nein — Redundant zu Node- <code>ssh-proxy</code> -Log
SSH-Routing (alle Kunden)	Node: <code>ssh-proxy</code> Container-Log + <code>/var/log/hosting-sshpiper.log</code>	SSH-Verbindungen aller Subs auf der Node, Fail2Ban-Quelle	Ja — Node-persistent

Container-Logs ueberleben Container-Recreate + Image-Rebuild (Bind-Mount ins Host-`/srv/hosting/subscriptions/<cid>/logs/`) und sind im taeglichen Restic- Backup enthalten — auch nach Node-Umzug also verfuegbar.

Logs auf der Node (ausserhalb Container)

Log	Pfad	Inhalt
Hosting-API	<code>journalctl -u hosting-api</code>	Node-API-Requests, Rebuilds, Sync-Runs
Backup	<code>/srv/hosting/backup-state.json</code> + <code>/var/log/hosting-backup-*.log</code>	Zeitpunkte, Erfolg/Fehler, Dauer, letzte 50 Events pro Sub
SSH-Piper	Container-Logs <code>ssh-proxy</code>	SSH-Routing-Entscheidungen (welcher User, welcher Sub)
Rebuild-Images	<code>/var/log/hosting-rebuild-images.log</code>	Image-Build-Output, Sub-Recreate
Self-Update	<code>/var/log/hosting-self-update.log</code>	Node zieht Binary + Templates vom Admin

Logs im Admin-Panel

Log	Ort	Inhalt
Aktivitaets-Log pro Sub	<code>subscription_events</code> -Tabelle, sichtbar auf Subscription-Show	Create/Update/Delete, Hostname/DB/Mail-Aenderungen, Backup-Trigger, Install, Restore — wer, wann, was
Access-Tokens fuer Kunden	pro Sub sichtbar	Erzeugung + Verwendung von Zugangs-Links
Logs-Viewer pro Sub	<code>subscriptions/{id}/logs</code>	Live-Tail Apache Access / Error / Container stdout (letzte 50 Zeilen)
Monitoring	pro Sub + pro Node	5-Min-Samples: CPU/RAM/Storage/Load, letzte 31 Tage

Zugriff fuer den Kunden

- **SSH in den Container** — direkt `tail -f ~/logs/<vhost>-access.log` (`~/logs` ist Symlink auf `/var/log/apache2/`).
- **Log-Viewer im Admin-Panel** — Reseller/Admin sieht letzte 50 Zeilen jedes Log-Typs (Apache Access, Error, Container-stdout) per Web ohne SSH.
- **Aktivitaets-Log** — Wer hat wann was im Panel gemacht (nur fuer eigene Subs).

Geplant / in Arbeit

- **Log-Rotation aktiv steuerbar** — heute unbegrenzt wachsend; geplant konfigurierbare Werte pro Sub (Groesse + Anzahl Backups + Retention-Tage).
- **Zentrale Log-Aggregation** — vermutlich Loki/Grafana als Node-uebergreifende Sammelstelle mit Retention, Suche und Alerts. Prioritaet mittel.
- **DB-Slow-Query-Log** — heute nicht aktiv; per Custom-DB-Config einzeln aktivierbar (`slow_query_log = 1`, `long_query_time = 1`).
- **Auditierbare Backup-/Restore-Events als CSV/JSON-Export** — heute nur im Admin sichtbar, nicht exportierbar.
- **Web-Log-Viewer fuer Endkunden** — heute nur Reseller sieht Logs im Panel; geplant eigene Kunden-Ansicht mit den letzten N Zeilen der eigenen Sub.

Datenschutz-Hinweis

Access-Logs enthalten IP-Adressen der Website-Besucher. Diese werden nach DSGVO-Standard 30 Tage aufbewahrt (per Docker-Log-Rotation faktisch weniger). Wenn Sie kuerzere Aufbewahrung brauchen: sagen Sie Bescheid, wir konfigurieren Log-Rotation entsprechend.

Fragen?

Bei Fragen zu Log-Zugriff / Retention / Exports einfach melden — die meisten Punkte oben sind Config-Aenderungen, keine Software-Aenderungen.

MariaDB-Tuning (Custom-Config)

MariaDB-Tuning per Custom-Config

Fuer jede Datenbank kannst du eigene MariaDB-Einstellungen setzen — analog zum PHP-Custom-INI. Die Werte ueberschreiben die Webunit-Defaults.

Wo?

Admin-Panel → Subscription → **Bearbeiten** → Tab **Datenbanken** → pro Datenbank ein Feld **Custom MariaDB-Config**.

Webunit-Defaults (bereits aktiv fuer alle DBs)

```
[mysqld]
max_allowed_packet = 64M
wait_timeout       = 600
interactive_timeout = 600
max_connections    = 200
```

Alles was du ins Custom-Feld schreibst, ueberschreibt diese Werte.

Copy-&-Paste-Snippets

Grosse Import-/Export-Operationen (grosse Bloetter, PDF-Anhaenge in DB)

```
max_allowed_packet = 256M
```

Shopware / grosser Katalog mit vielen Produkten

```
max_allowed_packet = 128M
innodb_buffer_pool_size = 512M
innodb_log_file_size = 128M
tmp_table_size = 128M
max_heap_table_size = 128M
sort_buffer_size = 8M
join_buffer_size = 8M
```

WordPress mit vielen Plugins

```
max_allowed_packet = 64M
innodb_buffer_pool_size = 256M
tmp_table_size = 64M
max_heap_table_size = 64M
```

Slow-Query-Log fuer Performance-Analyse

```
slow_query_log = 1
long_query_time = 1
log_queries_not_using_indexes = 1
```

Slow-Log landet im DB-Container unter `/var/lib/mysql/*.slow.log` und ist per SSH oder `docker exec` einsehbar.

Utf8mb4 mit korrekter Collation (Emojis in Kunden-Kommentaren)

```
character-set-server = utf8mb4
collation-server      = utf8mb4_unicode_ci
```

Strikter SQL-Mode fuer neue Projekte

```
sql_mode =
```

```
STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO_DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION
```

Whitelist — was ist erlaubt?

Erlaubt sind gängige Tuning-Parameter fuer:

- **Verbindungen / Timeouts:** `max_connections`, `max_user_connections`, `wait_timeout`, `interactive_timeout`, `connect_timeout`, `net_read_timeout`, `net_write_timeout`
- **Packet-Groesse:** `max_allowed_packet`
- **Buffers / Caches:** `sort_buffer_size`, `join_buffer_size`, `read_buffer_size`, `read_rnd_buffer_size`, `tmp_table_size`, `max_heap_table_size`, `table_open_cache`, `thread_cache_size`, `key_buffer_size`
- **InnoDB:** `innodb_buffer_pool_size`, `innodb_buffer_pool_instances`, `innodb_log_file_size`, `innodb_log_buffer_size`, `innodb_flush_log_at_trx_commit`, `innodb_flush_method`, `innodb_file_per_table`, `innodb_io_capacity[_max]`, `innodb_lock_wait_timeout`, `innodb_open_files`, `innodb_read_io_threads`, `innodb_write_io_threads`, `innodb_thread_concurrency`, `innodb_strict_mode`
- **Logging:** `slow_query_log`, `long_query_time`, `log_queries_not_using_indexes`, `general_log`
- **SQL-Mode / Charsets:** `sql_mode`, `character[_-_]set[_-_]server`, `collation[_-_]server`, `default[_-_]time[_-_]zone`, `default_storage_engine`
- **Sonstige:** `group_concat_max_len`, `lower_case_table_names`, `transaction_isolation`, `skip_name_resolve`, `open_files_limit`

Nicht-erlaubte Keys werden **still verworfen** — landen als Kommentar in der generierten `99-webunit-custom.cnf` (im DB-Container unter `/etc/mysql/mariadb.conf.d/`). Verworfen werden aus Sicherheitsgrunden z.B. `secure_file_priv`, `plugin_dir`, `log_bin`, `datadir` — die wuerden den Container-Betrieb kompromittieren.

Konfigurations-Format

- Eine Direktive pro Zeile: `key = value` oder `key value`
- Kommentare mit `#` oder `;`
- Sektions-Header wie `[mysqld]` werden ignoriert — alles landet automatisch im `[mysqld]`-Block der generierten Config
- `-` und `_` in Key-Namen sind aequivalent (MariaDB-Konvention)

Was passiert nach dem Speichern?

1. Werte werden gegen die Whitelist geprueft (Verwerfen bei Verstoß).
2. Neue `99-webunit-custom.cnf` wird auf der Node erzeugt.
3. Der **DB-Container wird noch nicht** automatisch neu gestartet — du siehst den neuen Wert erst nach einem Restart:

Admin-Panel → Subscription → Container → "DB-Container neustarten"

(oder wir bauen das noch als Auto-Restart bei Config-Aenderung — im Zweifel: uns ansprechen).

Aktuelle Werte pruefen

Per SSH ins Sub-Container:

```
mariadb -u <username> -p<pass> -e "SHOW VARIABLES LIKE 'max_allowed_packet'"
```

Oder via phpMyAdmin (Zahnrad-Icon rechts oben, dann "Server-Variablen").

Wichtige Anmerkungen

- `innodb_log_file_size` **aendern** heisst: beim naechsten Start regeneriert MariaDB das Redo-Log. Das kann 1-2 Minuten dauern. Nur aendern wenn's einen Grund gibt.
- `innodb_buffer_pool_size` sollte ~50-70% des DB-Container-Speichers sein. Wenn du die Container-Memory-Limit unter dem Wert setzt, wird MariaDB OOM-killed.
- `max_connections` **hochziehen** hilft nur wenn dein Framework wirklich so viele parallele Verbindungen macht. Standard 200 reicht meist — hoehere Werte kosten RAM ohne Nutzen.

CLI-Befehle im Container

CLI-Befehle im Container

Diese Befehle stehen im SSH-Zugang deines Containers zur Verfügung. Alle laufen als dein Container-User (kein sudo nötig, kein Root-Zugriff).

webunit-supervisor — Hintergrund-Prozesse

Verwaltung von Symfony-Messenger-Workern, Queue-Workern, Node-Daemons.

```
webunit-supervisor reload      # ~/.webunit/supervisor.d/*.conf einlesen
webunit-supervisor status      # Status aller Programme
webunit-supervisor restart <name> # Ein Programm neu starten
webunit-supervisor tail <name>  # Log-Zeilen (stdout)
webunit-supervisor tail-err <name> # Log-Zeilen (stderr)
webunit-supervisor validate     # Syntax-Check ohne Reload
webunit-supervisor help
```

Details siehe Hintergrund-Prozesse (Supervisor).

webunit-ssh — SSH-Auth-Verwaltung

```
webunit-ssh status             # Aktueller PW-Auth-Modus + Key-Anzahl
webunit-ssh list-keys          # Aktive SSH-Keys auflisten
webunit-ssh add-key '<pubkey>'  # SSH-Public-Key hinzufuegen
webunit-ssh remove-key <fingerprint> # SSH-Key loeschen (fp aus list-keys)
webunit-ssh password-auth on|off # Passwort-Login erlauben / sperren
webunit-ssh help
```

Beispiel-Ausgabe:

```
== SSH-Auth-Konfiguration ==
Passwort-Login:  DEAKTIVIERT (nur Key)
Aktive Keys:      3
Verwaltet in:     /etc/webunit/sshpipecpub.key (managed – nicht editieren)
Key hinzufuegen:  webunit-ssh add-key '<public-key>'
```

Wichtig zu `~/.ssh/authorized_keys`: Diese Datei wird von unserer Infrastruktur IGNORIERT. Eintraege dort haben keine Wirkung. Die Datei enthaelt bewusst nur einen Hinweis-Text und ist read-only — sshd im Container liest ausschliesslich `/etc/webunit/sshpipecpub.key`, was vom Admin-System zentral verwaltet wird.

Keys hinzufuegen: `webunit-ssh add-key '<public-key>'` (siehe unten) oder im Admin-Panel unter „SSH-Keys“.

Keys auflisten:

```
$ webunit-ssh list-keys
== Aktive SSH-Keys (3) ==
SHA256:HkktzVstpsuXhuWUej/qTQ24CL+6CB8YkUMhnjZ+ow
  Label:    Vom Container 05.09.2026 10:20 – eddsa-key-20260905
  Angelegt: 2026-09-05 10:20:20   Laeuft ab: -

SHA256:...
...

Loeschen: webunit-ssh remove-key <fingerprint>
```

Key loeschen (Fingerprint aus `list-keys` kopieren):

```
$ webunit-ssh remove-key SHA256:HkktzVstpsuXhuWUej/qTQ24CL+6CB8YkUMhnjZ+ow
Loesche Key SHA256:... – dies ist SOFORT wirksam.
Fortfahren? [j/N] j
Key geloescht – SHA256:... (Vom Container 05.09.2026 10:20 – eddsa-key-20260905)
In wenigen Sekunden verweigert sshpipec Logins mit diesem Key.
```

Nach Loeschung: ein spaeteres `webunit-ssh add-key` mit demselben Public-Key stellt den Key wieder her (gleicher DB-Eintrag, `restored`) statt neu anzulegen. Doppel-Adds sind idempotent (`already_known`).

Passwort-Login-Toggle: `webunit-ssh password-auth on|off` oder im Admin-Panel.

webunit-password — Passwoerter selbst rotieren

```
webunit-password ssh          SSH-/SFTP-Passwort neu setzen
webunit-password db <label>  Datenbank-Passwort neu setzen
webunit-password help
```

Detaillierte Anleitung inkl. Sicherheits-Hinweisen: siehe [Passwoerter selbst aendern](#).

webunit-opcache-reset — OPcache leeren (Deployer-freundlich)

Leert den PHP-OPcache im Apache-Prozess ohne Restart, ohne Downtime. Zusaetzlich werden der `realpath`-Cache und (falls aktiv) APCu geleert — zusammen sind das exakt die Caches, die nach einem Symlink-Umbiegen (`current -> releases/N`) noch auf die alte Release zeigen wuerden.

```
webunit-opcache-reset
```

Beispiel-Ausgabe:

```
OPcache geleert.
Vorher: 4231 Scripts im Cache, 84318720 Bytes belegt.
```

Warum es dieses CLI gibt: Die uebliche Empfehlung `cachetool.phar opcache:reset -- fcgi=/run/php-fpm.sock` funktioniert bei uns NICHT — unser Image faehrt Apache mit `mod_php` (kein PHP-FPM, kein `.sock`). Fuer denselben Zweck gibt es genau dieses CLI.

Wenn du in einem Deployer-Task ohne SSH-Sitzung arbeiten willst, bleibt auch Cachetool im `--web`-Adapter moeglich:

```
php cachetool.phar opcache:reset \
  --web=curl --web-path=./public --web-url=https://deine.domain
```

Cachetool legt dazu kurz eine Adapter-PHP-Datei ins Public, ruft sie per HTTP an und loescht sie wieder. Das erwischt denselben `mod_php`-OPcache. Vorteil `webunit-opcache-reset`: kein Public-Endpoint, kein Aufräumen, zusaetzlich `realpath+apcu`.

pma-link — PMA-Tageslink

pma-link

Gibt eine URL aus, die dich einmalig fuer 12 Stunden zu phpMyAdmin auf deiner Node lotst. Die URL ist ein signierter Tageslink — ohne den ist das PMA-Login-Formular von aussen nicht mal erreichbar.

Beispiel:

PMA-Zugang (12h gueltig ab jetzt, Tages-Grenze 04:00 Berlin):

```
https://pma.<node>.webunit.cloud/pma-open?daykey=...&cid=...&node=...
```

Link im Browser oeffnen — Admin validiert und leitet weiter zu PMA.

Wichtig: die URL ist nur einen "Arbeitstag" gueltig. Tages-Grenze ist 04:00 Berlin — falls du morgens frueh anfaengst, laeuft dein Cookie ab 04:00 ab und du brauchst einen neuen Link. Nachts arbeitende Kunden haben von 04:00 an denselben Link 24 Stunden minus 12 Cookie-Timeout.

Wer die URL kennt, kommt zu PMA — nicht in oeffentlichen Chats posten.

Was du KEIN CLI dafuer brauchst

- Passwort/Zugangsdaten anzeigen: Admin-Panel oder Access-Link
- Passwort rotieren: Admin-Panel (Subscription → SSH → "Passwort neu wuerfeln" oder Datenbanken → "Rotieren")
- DB anlegen/aendern: Admin-Panel
- PHP-Werte tunen: Admin-Panel (Custom-INI-Textblock pro Hostname)
- MariaDB-Werte tunen: Admin-Panel (Custom-Config-Textblock pro DB)

Bash-Zusatzausstattung im Container

Beim ersten Login findest du in deinem Home:

- `~/.bashrc` mit Aliases (`ll`, `www` fuer `cd /var/www`)
- `~/www` → Symlink auf `/var/www` (Docroot)

- `~/logs` → Symlink auf `/var/log/apache2` — pro vhost gibt es hier `<domain>-access.log` und `<domain>-error.log`. **Persistent** ueber Container-Recreate hinweg und im taeglichen Backup enthalten.
- `~/.webunit/` — deine eigenen Supervisor-Configs (`supervisor.d/`) und Programm-Logs (`logs/`). Auch persistent + im Backup.
- `~/.ssh/authorized_keys` — **wird von unserer Infrastruktur IGNORIERT**. Die Datei enthaelt einen Hinweis-Text und ist read-only. sshd liest ausschliesslich `/etc/webunit/sshpiper_pub.key` (zentral verwaltet). Zum Key-Hinzufuegen: `webunit-ssh add-key '<public-key>'` oder Admin-Panel.
- `~/.ssh/id_rsa`, `~/.ssh/config`, `~/.ssh/known_hosts` — voll unter deiner Kontrolle, fuer OUTGOING SSH (Deploy, git clone via SSH, rsync). Nur `authorized_keys` ist speziell.

Zusaetzlich vorinstalliert: `composer`, `wp` (WP-CLI), `node`, `npm`, `git`, `curl`, `mariadb`, `msmtp` (Mail via `/usr/sbin/sendmail`), `nano`, `zstd`.

Fehlendes Tool? Kurz per Support melden — wir installieren pauschal ins Image, alle Kunden bekommen es.

Passwoerter selbst aendern

Passwoerter selbst aendern

Du kannst SSH-/SFTP- und Datenbank-Passwoerter jederzeit selbst rotieren — direkt aus deinem Container per CLI. Kein Ticket, keine Wartezeit.

Der Befehl ist `webunit-password`. Rotation ist **sofort wirksam** und das neue Passwort wird **einmalig** angezeigt — kopieren und speichern.

SSH-/SFTP-Passwort

```
$ webunit-password ssh
```

Das aktuelle SSH-Passwort wird sofort unguelteig. Bestehende Sitzungen bleiben offen, aber neue Logins brauchen das neue Passwort.

Wichtig: vorher eine zweite SSH-Sitzung mit dem neuen Passwort aufmachen und pruefen, dass Login klappt, BEVOR diese Sitzung geschlossen wird. So kommst du nicht versehentlich aus.

Fortfahren? [j/N] j

Neues SSH-Passwort:

8f3aK2q7mZpxN4Rtv9LcJhBw

Jetzt in einer zweiten Sitzung testen. Erst wenn Login mit dem neuen Passwort klappt, diese Sitzung schliessen.

Wichtig: deine aktuelle SSH-Sitzung laeuft nach der Rotation weiter, SSH prueft das Passwort nur beim Login. Bevor du die Sitzung schliesst, oeffne eine **zweite** SSH-Verbindung mit dem neuen Passwort und pruefe, dass Login klappt. Erst dann die erste Sitzung beenden. Sonst kannst du dich aussperren.

Datenbank-Passwort

Das Label ist der Datenbank-Name aus dem Admin-Panel (meist `main` oder ein Kunden-Name — im Zweifel: Datenbank-Kachel in der Subscription-Ansicht).

```
$ webunit-password db main
```

Das aktuelle Datenbank-Passwort fuer 'main' wird sofort unguelteig.

Deine Anwendung wird die Datenbank nicht mehr erreichen, bis du das neue Passwort in deiner Config eingetragen und die Anwendung neu geladen hast.

Fortfahren? [j/N] j

Neues Datenbank-Passwort (Label: main):

Q7pX2mkV9fLcNhZa8Bwry3Ju

Trage das Passwort in die Config-Datei deiner Anwendung ein und starte die Anwendung neu.

Wichtig: das alte Passwort ist mit der Rotation **weg** — keine Uebergangsphase. Ab dem Moment schlaegt jeder DB-Zugriff mit dem alten Passwort fehl. Direkt danach das neue Passwort in die Konfigurations- Datei deiner Anwendung eintragen und die Anwendung neu laden.

Wo genau das Passwort in deiner Anwendung eingetragen wird, weisst du selbst am besten — je nach Anwendung eine Config-Datei, eine Environment-Variable oder ein Secret-Manager. Nach dem Eintragen die Anwendung neu starten oder ihre Konfiguration neu einlesen lassen.

Was passiert im Hintergrund

Die CLI ruft unseren Node-Dienst auf, der:

1. ein neues Zufalls-Passwort erzeugt (24 Zeichen),
2. es lokal setzt (SSH: `chpasswd`; DB: `ALTER USER ... IDENTIFIED BY`),
3. es an unser Admin-Panel meldet, damit es dauerhaft ist und beim naechsten Sync nicht ueberschrieben wird,
4. es dir einmalig ausgibt.

Das neue Passwort landet nirgends in Logs. Wenn du es verlierst, musst du erneut rotieren.

Alternative: SSH-Key statt Passwort

Sicherer als Passwort-Rotation ist ein SSH-Key. Public-Key einmalig in `~/.ssh/authorized_keys` deines Containers ablegen, danach kannst du den Passwort-Login im Admin-Panel komplett abschalten. Keys ueberleben Container-Recreate automatisch (werden von uns synchronisiert).