Operations

Installation

```
sudo tools/worker install
```

This copies the systemd unit files (`vte-worker@.service` and `vte-worker@.socket`) to `/etc/systemd/system/`, enables the socket, and starts it. Both files are systemd templates that take the relative path from `/var/www/html` as the instance parameter (e.g. `vte-worker@vte-agentic`), which allows running multiple Workers for different VTE installations on the same machine.

If your installation differs from `/var/www/html/PATH`, you must edit the templates manually or with `systemctl edit [--full]` for both `vte-worker@.socket` and `vte-worker@.service`, before installing.

Commands

Command	Effect
`tools/worker install`	Install systemd units, enable and start the socket
`tools/worker restart`	Send a restart signal to the running Worker
`tools/worker status`	Print uptime, number of active consumers, queued jobs

Signals

Signal	Effect
`SIGTERM` / `SIGINT`	Graceful shutdown: stop accepting new connections, wait for all running Consumers to finish, then exit.
`SIGHUP` / `SIGUSR1`	Reload.

Configuration

Setting	Location
Maximum concurrent Consumers	`Zygote.php — CONSUMERS_MAX`

Socket path	`systemd/vte-worker@.socket – ListenStream`
	
	`config.inc.php – $worker_socket_URL`

Log file	`logs/worker.log`

Web servers config

All clients must hold a streaming connection to the worker to receive events in **real time**. This means your web server pipeline must handle

- **concurrently**
- **without buffering or compression**
- **at least 1 connection per user across all vte installs** for browsers that support the SharedWorker
- **plus margin for heartbeats**

this translates to reverse proxies, single virtualhosts, apache proxy module, php fpm pools, ecc...

To simplify targeting the streaming connections only without interfering with other requests flows, `sse.php` can be choosen.

PHP FPM pool

`/etc/php/8.3/fpm/pool.d/sse.conf`

```
[sse]
user = www-data
group = www-data

; align with your web server
listen = /run/php/php8.3-fpm-sse.sock
listen.owner = www-data
listen.group = www-data
listen.mode = 0660

pm = dynamic

; at least 1 per user + 10 for heartbeats
; 30 users for vte1, 20 users for vte2 -> 60 childrens
pm.max_children = 60
```

```
pm.start_servers = 8
pm.min_spare_servers = 4
pm.max_spare_servers = 12
pm.max_requests = 500

; enable to verify (domain.com/status?full)
; pm.status_path = /status
```

Apache + mod_proxy

`/etc/apache2/sites-available/domain.conf`

```
<VirtualHost *:80>
  # ...

  <FilesMatch \.php$>
    # example pre-existing config
    SetHandler "proxy:unix:/run/php/php8.3-fpm.sock|fcgi://main"
  </FilesMatch>

  <Location /sse.php>
    # /run/php/php8.3-fpm-sse.sock - different pool, must be configured in php-fpm
    # fcgi://sse - "sse" is an arbitrary name, must not conflict with other apache pools
    SetHandler "proxy:unix:/run/php/php8.3-fpm-sse.sock|fcgi://sse"
    SetEnv no-gzip 1
    SetEnv dont-vary 1
  </Location>

  <Proxy "fcgi://sse">
    ProxySet flushpackets=on timeout=600
  </Proxy>

  # ...
</VirtualHost>
```

NGINX gateway

`/etc/nginx/conf.d/domain.conf`

```
server {
  # ...

  location ~ \.php(?:$|/) {
    include fastcgi_params;
    # ...
    fastcgi_pass_header X-Accel-Buffering;  # forward header to client (to reverse proxy)
  }

  # ...
}
```

NGINX reverse proxy

`/etc/nginx/conf.d/proxy_hosts/domain.conf` (can differ)

```
server {
  # ...

  location /sse.php {
    proxy_buffering     off;
    proxy_read_timeout  3600;
    proxy_send_timeout  60;
    # ...
    proxy_pass       http://...;
  }

  # ...
}
```

Troubleshooting

Enable Logging

Logging is disabled by default. To enable it, set the static flag before starting the Worker:

```
\Vtenext\Services\Worker\Worker::$enableLog = true;
```

Logs are written to `logs/worker.log`. The log format includes a timestamp, the process role, and the message.

In the browser, put the console verbosity to debug.

Common Issues

Symptom	Likely Cause
Connection refused	The Worker is not running or the socket path is incorrect. Check `$worker_socket_URL` in `config.inc.php` and verify the socket file exists.
Consumer not starting	Process limit reached (`RLIMIT_NPROC`). The Worker attempts to raise it to 1000 + `CONSUMERS_MAX` at startup.
Job queued but never runs	All 10 Consumer slots are occupied by long-running tasks. Check `tools/worker status` for current usage.
CRM doesn't load anymore	All PHP-FPM pool slots are saturated, check with the `/status` endpoint

SharedWorker

When the SharedWorker is active, debugging can done differently on each browser:

- **Firefox** — visit about:debugging#workers (copy-paste) and press **Debug** then you can inspect everything and put breakpoints. Logs are also shown in the first tab that spawned the SW (*yes 3 times, it's a browser bug as of June 2026*) and in the browser console in multi-process mode (Ctrl+Shift+J).
- **Chrome** — visit chrome://inspect/#workers (copy-paste) and press **Inspect**, same story. No logs are shown in individual tabs.

Technical limitations

Browsers connections cap

Web connections from browser tabs rely on Server-Sent Events (SSE), which keep a long-lived HTTP connection open to the server. Browsers enforce a hard limit of **6 concurrent connections per domain** (HTTP/1.1). The `SharedWorker.js` script multiplexes all tabs through a single SSE connection per browser, bypassing the limit entirely.

- When SharedWorker is not available (older or unsupported browsers), each tab opens its own SSE connection and the 6-connection limit per domain applies.
- The Worker's `CONSUMERS_MAX` limit (10 concurrent Consumer processes) is a separate server-side concern — it caps how many jobs run in parallel, not how many connections can remain open.

Disconnection detection

TCP provides no built-in notification when a peer disconnects, but even so some web servers components don't forward such events. To detect that a browser has closed the connection, the `WebConnector` writes a newline at every read tick (default every 10 seconds) and clients must keep sending heartbeats (default every 30 seconds) otherwise the worker will reap them.

Xdebug and `set_time_limit`

When the Xdebug extension is loaded, `set_time_limit(0)` (which normally removes the execution time limit) does not work reliably. Xdebug overrides PHP's internal timer and enforces its own `xdebug.max_nesting_level` and related constraints, which can cause long-running Consumer methods to be terminated prematurely on development environments where Xdebug is active. If the Worker behaves unexpectedly during development, disable Xdebug or set `xdebug.mode=off` in the PHP configuration.

Revision #5
Created 2026-09-17 14:12:04 UTC by Diego
Updated 2026-09-18 13:29:50 UTC by ddalmaso