Background Job Processing

VeloxFactory processes background work through a Redis-backed queue managed by Laravel Horizon: thumbnail generation, mail dispatch, and everything the Job Scheduler fires. Horizon runs as a single supervised process and manages its own worker pool internally, with dynamic scaling, real-time monitoring and a built-in dashboard.

Prerequisites

Horizon has two hard requirements beyond the base VeloxFactory stack: a running Redis instance and the `php-redis` PHP extension. Neither is optional, Horizon will refuse to start without both.

Redis

Redis can be installed natively or run as a Docker container. Both work equally well; the choice depends on your infrastructure preferences.

Native installation:

```
apt install redis-server
systemctl enable redis-server
systemctl start redis-server
```

Docker container:

```
docker run -d \
  --name
  --restart
  -p
  redis:alpine
```

The container binds to `127.0.0.1` only, Redis is not exposed to the network, which is the correct default for a single-server deployment.

Whichever method you choose, verify connectivity before proceeding:

```
redis-cli                                                                ping
#                                       Expected:                        PONG
```

PHP Redis Extension

VeloxFactory is configured to use `phpredis` as the Redis client. The extension must be installed and active for PHP:

```
apt                                    install                                    php-redis
systemctl   restart   php8.2-fpm       #   adjust   version   to   match   your   PHP   installat
```

Confirm the extension is loaded:

```
php                    -m                    |                    grep                    redis
#                                       Expected:                                       redis
```

.env Configuration

With Redis running and the extension installed, update your `.env` to activate Redis as the queue driver:

```
QUEUE_CONNECTION=redis

REDIS_CLIENT=phpredis
REDIS_HOST=127.0.0.1
REDIS_PASSWORD=null
REDIS_PORT=6379
```

Web Server

VeloxFactory requires a web server that routes all requests through Laravel's `public/index.php` entry point. Both Apache2 and nginx are supported. The Horizon dashboard at `/horizon` and the Reverb WebSocket endpoint require no special routing rules, they are handled by Laravel and PHP-FPM like any other request, with one exception: Reverb needs a WebSocket proxy pass.

Apache2

Enable `mod_rewrite` before configuring the vhost:

```
a2enmod rewrite proxy proxy_http proxy_wstunnel
systemctl restart apache2
```

Virtual host configuration:

```
<VirtualHost *:80>

    </Directory>

    #

    CustomLog ${APA
</VirtualHost>
```

Laravel's bundled `.htaccess` in `public/` handles the rewrite rules, no additional configuration needed for URL routing.

nginx

```
server {
```

```
    }

                                                                                                 f

    }

                                                                                               #

    }

                                                                                      location

    }
}
```

> ℹ **Adjust the PHP-FPM socket path to match your PHP version.** On systems with multiple PHP versions installed, the socket is typically at `/var/run/php/php8.2-fpm.sock`. Verify with `ls /var/run/php/`.

Queues

VeloxFactory uses three queues with distinct priorities:

Queue	Jobs	Notes
`high`	Thumbnail generation	Dispatched on-demand when a report is rendered. Processed with highest priority, workers on this queue are never blocked by maintenance routines.
`default`	Mail dispatch, the scheduler tick	Mails sent in background mode are queued here, together with the minute tick that looks for due schedules. Both are short jobs.
`scheduler`	Runs of the Job Scheduler	Scheduled renders and the purge run. These are the long jobs of the instance, minutes rather than seconds, so they get a queue of their own.

> **i The three queues run in separate worker pools.** A purge run on the `scheduler` queue cannot delay thumbnail generation on `high` or a mail on `default`, they never compete for the same worker.

Mail Dispatch

A mailing sent with `sendAsync` creates its mail task immediately and hands the send to the `default` queue. The job carries only the ID of the task, never the attachments, so nothing large ever travels through Redis: the files are read from the history disk at send time.

A mail that runs into the dispatch rate limit of its mailer is not failed and not lost. The job is released back onto the queue with a delay until the next window opens, and the task stays *pending* in the meantime. Several workers may try to send at the same time, the limiter counts atomically in Redis, so the configured caps hold regardless of how many workers are running.

Scheduled Work

Every recurring job of the instance is a record in the Job Scheduler, with its own crontab expression. Laravel's own schedule holds exactly one entry: once a minute it queues `DispatchDueScheduledJobsJob`, which reads the due schedules from the database and queues one run per schedule on the `scheduler` queue.

That includes the purge run, which deletes print tasks, mail tasks, history records, orphaned files and the scheduler's own run log in one pass and in a fixed order. The *Schedule your jobs* chapter covers the retentions and the rest of the configuration.

A scheduled run is attempted once and is never retried: a second attempt would print a second label and send a second mail. Two runs of the same schedule never overlap either. What happened

is on the run record, with its trace id.

Horizon Supervisors

Horizon manages workers through internal supervisors, process groups, each responsible for one queue. The system-level Supervisor (Supervisord) only ever manages the single Horizon master process; Horizon itself handles everything below that.

Supervisor	Queue	Balancing	Notes
`supervisor-rendering`	`high`	Auto	Scales worker processes dynamically based on queue depth, up to 10 in production. Job timeout: 120 seconds.
`supervisor-default`	`default`	Simple	Fixed worker count, up to three processes in production. Runs with lower CPU priority (`nice 10`), mail dispatch should not compete with rendering for system resources. Job timeout: 300 seconds.
`supervisor-scheduler`	`scheduler`	Simple	Up to two processes, on its own queue connection `redis-scheduler` and with lower CPU priority (`nice 10`). Job timeout: 1800 seconds, long enough for a purge over a large history disk or a batch of scheduled renders.

> ℹ **A connection's `retry_after` has to stay above the job timeout of every supervisor using it.** A job that runs longer than `retry_after` is considered lost and handed to a second worker while the first is still working on it, which for a scheduled render means a duplicate print and a duplicate mail. The `redis` connection allows 360 seconds against a timeout of 300, `redis-scheduler` allows 1860 against 1800. Raise `retry_after` first whenever you raise a timeout.

System Supervisor Configuration

Supervisord keeps three VeloxFactory processes alive and restarts them automatically on failure: the Horizon master, the Reverb WebSocket server, and the scheduler.

```
; Horizon - manages all VeloxFactory queue workers internally
[program:veloxfactory-horizon]
directory=/var/www/veloxfactory
command=/usr/bin/php artisan horizon
user=www-data
process_name=%(program_name)s
numprocs=1
autostart=true
autorestart=true
startretries=10
stopasgroup=true
killasgroup=true
stopsignal=TERM
startsecs=3
stopwaitsecs=1860
redirect_stderr=true
stdout_logfile=/var/log/supervisor/veloxfactory-horizon.log
environment=HOME="/home/www-data",PATH="/usr/local/bin:/usr/bin:/bin"

; Reverb - WebSocket server for real-time events
[program:veloxfactory-reverb]
directory=/var/www/veloxfactory
command=/usr/bin/php artisan reverb:start
user=www-data
autostart=true
autorestart=true
startretries=10
stopasgroup=true
killasgroup=true
stopsignal=INT
startsecs=3
stopwaitsecs=60
redirect_stderr=true
stdout_logfile=/var/log/supervisor/veloxfactory-reverb.log
```

```
environment=HOME="/home/www-data",PATH="/usr/local/bin:/usr/bin:/bin"

;      Scheduler      -      runs      Laravel's      task      schedule      every      minute
[program:veloxfactory-schedule]
directory=/var/www/veloxfactory
command=/usr/bin/php                              artisan                              schedule:work
user=www-data
autostart=true
autorestart=true
startretries=10
stopasgroup=true
killasgroup=true
stopsignal=INT
startsecs=3
stopwaitsecs=60
redirect_stderr=true
stdout_logfile=/var/log/supervisor/veloxfactory-schedule.log
environment=HOME="/home/www-data",PATH="/usr/local/bin:/usr/bin:/bin"
```

i `stopsignal=TERM` is required for Horizon. SIGTERM triggers a graceful shutdown, Horizon finishes any in-flight jobs before stopping its workers. Using SIGKILL or SIGINT instead will interrupt running jobs mid-execution and may leave Report History Records in an incomplete state.

The scheduler process drives every recurring job of the instance. Without `schedule:work` running (or an equivalent cron entry calling `schedule:run` every minute), no schedule fires: no scheduled render, no cleanup, and the disk keeps filling. A cron entry is the alternative on hosts without Supervisord:
`* * * * * cd /var/www/veloxfactory && php artisan schedule:run >> /dev/null 2>&1`

After updating the configuration:

```
supervisorctl                                                                    reread
supervisorctl                                                                    update
supervisorctl                                                                    status
```

The Dashboard

The Horizon dashboard is available at `/horizon`. It is restricted to users with the `global:admin` permission, the same permission required to manage users and system-wide settings.

The dashboard provides a real-time view of the entire queue system: pending and completed jobs, throughput metrics, worker counts per supervisor, and a full log of failed jobs with their stack traces. Failed jobs can be retried directly from the dashboard without any CLI access.

For scheduled work the Job Scheduler's own run log is the better place to look: it holds one record per run with its trace id, its result and the history record it produced. Horizon shows the job, the run log shows what the job did.

Graceful Shutdown & Deployments

When Horizon is stopped, for deployments, configuration changes, or server maintenance, it finishes any jobs currently in flight before exiting. The Supervisord configuration allows up to 1860 seconds for this drain, matching the longest job timeout of any supervisor, so even a purge run over a large history disk finishes cleanly.

```
# Stop Horizon gracefully (in-flight jobs will finish first)
supervisorctl stop veloxfactory-horizon

# Restart after a deployment
supervisorctl restart veloxfactory-horizon
```

Never force-kill the Horizon process during a deployment. Always use `supervisorctl stop` or `supervisorctl restart`, both send SIGTERM and wait for the drain period.

A deployment window shorter than the drain period is worth planning around the schedules: a restart during the nightly purge waits for it, and a schedule whose slot falls into the downtime is dropped rather than fired late.

Revision #7
Created 2026-06-15 19:30:51 UTC by Benjamin Fischer
Updated 2026-09-21 10:19:25 UTC by Benjamin Fischer