

Fine tuning 2026.10.26

DEBRiS optimization guide

Based on server-info-DEBRiS-20261026-130457 (Ubuntu 24.04, nginx, PHP 8.4-FPM, MariaDB 10.11, Redis, Docker). Each change names the file, the line to find, what to put instead, and why.

1. What the report shows

DEBRiS is the smallest server so far: a Celeron N3060 with 2 cores and 2 threads (no hyper-threading), 1.8 GB RAM, a 2 GB swap file, and a 28 GB eMMC chip as the root disk. The data lives on two USB hard disks (/home/www-data with 293 GB and /home/app-data, where Docker keeps its data). It has run 5 days 19 hours with no out-of-memory kills and no service errors. Right now 952 MB is used, 866 MB is available and 318 MB of swap is in use.

Two things make this machine different from the others:

- RAM is tiny, so one greedy program can slow everything down.
- The root disk is eMMC flash. It is fast enough, but it has a limited number of write cycles. Swap, the system journal, MariaDB's logs and temporary tables, nginx logs and Redis snapshots are all written to it, so tuning here also means writing less.

The nginx configuration is already well adapted to a small host (worker_connections 1024, keepalive 30, gzip level 2 with pre-compressed files, HTTP/3 off), so most work is in PHP, memory and Docker.

#	Area	What the report says	Priority
1	PHP-FPM	5 workers x memory_limit 512M can ask for 2.5 GB on a 1.8 GB machine (141% of RAM); process.priority -19	High
2	Swap and eMMC	Swap file sits on the eMMC; swappiness 60; dirty-page limits are percentages of RAM; journald uses 48 MB	High

3	Docker	n8n (313 MB), its Postgres and Navidrome have no memory limit; live-restore is off; logs are not rotated	High
4	OPcache	Tracing JIT with a 100 MB buffer on a 1.8 GB machine; 100 000 file slots and only 8 MB interned strings	Medium
5	Background services	multipathd, snapd and fwupd use about 90 MB together	Medium
6	MariaDB	Almost default configuration; 29% of temp tables go to disk (the eMMC); key_buffer_size 128M is unused	Low
7	Redis	Healthy (95.8% hit rate) but has no maxmemory	Low

Already good: nginx, MariaDB (63 MB of data, 0.065% buffer pool misses, 3 slow queries in 2.8 days, never more than 3 connections), Redis (1.5 MB, on a socket), all Docker ports on 127.0.0.1, the Docker data directory on the hard disk instead of the eMMC.

2. Before you start

Back up each file before editing and test before you reload:

```
sudo cp -a /etc/php/8.4/fpm/pool.d/www.conf /etc/php/8.4/fpm/pool.d/www.conf.bak-$(date +%F)
sudo php-fpm8.4 -t
sudo nginx -t
```

Safest order: swap, services and journal first, then PHP-FPM, OPcache, MariaDB, Redis, Docker. Take a baseline so you can compare later:

```
free -m
cat /proc/pressure/memory
awk '{printf "%d MB written to eMMC since boot\n", $7/2048}' /sys/block/mmcblk1/stat
```

The second line shows how much time programs spent waiting for memory (PSI). The third shows how much has been written to the eMMC since the last boot; the health script reports the same number per day.

To see the chip's own wear estimate (not every eMMC reports it), try either of these:

```
cat /sys/block/mmcblk1/device/life_time  
sudo apt install mmc-utils && sudo mmc extcsd read /dev/mmcblk1 | grep -i 'life time'
```

A value of 0x01 means 0 to 10% of the rated life is used; 0x0A means 90 to 100%.

3. Swap, journal, services and kernel

3.1 zram: compressed swap in RAM

Your swap is a 2 GB file on the eMMC. Swapping to flash is slow and wears the chip. zram stores swapped pages compressed in RAM instead (typically 2 to 3 times smaller), which is far faster and writes nothing to flash.

```
sudo apt install zram-tools
```

Edit /etc/default/zramswap so it reads:

```
ALGO=zstd  
PERCENT=50  
PRIORITY=100
```

PERCENT=50 allows up to about 900 MB of zram. PRIORITY=100 makes the kernel fill zram first and use /swap.img (priority -2) only as overflow. Apply and check:

```
sudo systemctl restart zramswap  
swapon --show  
zramctl
```

After a week, if swap.img shows almost no use, you can remove it to stop all swap writes to the eMMC: `sudo swapoff /swap.img` and put a `#` in front of the `/swap.img` line in `/etc/fstab`. Keep it for now as a safety net.

3.2 sysctl settings

Create /etc/sysctl.d/99-debris.conf:

```
vm.swappiness = 100
vm.page-cluster = 0
vm.dirty_background_bytes = 33554432
vm.dirty_bytes = 134217728
net.core.netdev_max_backlog = 2048
net.ipv4.tcp_max_syn_backlog = 1024
```

Line	Why
vm.swappiness = 100	With fast compressed swap it is cheaper to swap out an idle program page than to throw away the file cache that nginx and Nextcloud reuse. This is common practice with zram. If you skip zram, use 20 instead.
vm.page-cluster = 0	Reads back one page at a time from swap instead of eight; right for zram.
vm.dirty_background_bytes and vm.dirty_bytes	The current dirty_ratio 20 and dirty_background_ratio 10 are percentages of RAM (360 MB and 180 MB here). On slow USB disks that lets a big upload pile up unwritten data and then stall. These byte values start writeback at 32 MB and cap the backlog at 128 MB. Setting the _bytes versions replaces the _ratio ones.
the two network lines	tcp_max_syn_backlog is only 128 and netdev_max_backlog 1000; raising them only helps with bursts of connections.

Apply with `sudo sysctl --system` and check with `cat /proc/sys/vm/swappiness`.

3.3 Limit the system journal

systemd-journald uses 48 MB of RAM and writes to the eMMC. Edit `/etc/systemd/journald.conf`, in the [Journal] section, and set (remove the # in front of each line if present):

```
SystemMaxUse=100M
SystemMaxFileSize=16M
MaxRetentionSec=2week
```

Apply and trim what is there now:

```
sudo systemctl restart systemd-journald
sudo journalctl --vacuum-size=100M
journalctl --disk-usage
```

3.4 Stop services you do not use

multipathd (27 MB), snapd (27 MB) and fwupd (33 MB) are running. First check that you do not need them:

```
sudo multipath -ll
snap list
```

If multipath -ll prints nothing, you have no multipath disks; switch multipathd off. If snap list shows only base snaps (bare, core, snapd) and nothing you use, snapd can go too. fwupd only delivers firmware updates, which a headless server seldom needs.

```
sudo systemctl disable --now multipathd.service multipathd.socket
sudo systemctl disable --now fwupd-refresh.timer
sudo systemctl disable --now snapd.service snapd.socket snapd.seeded.service
```

Skip the last line if snap list shows something you rely on. Together this frees roughly 90 MB, 5% of your RAM. Also make sure the weekly TRIM job for the eMMC is active: systemctl status fstrim.timer.

4. PHP-FPM pool

File: /etc/php/8.4/fpm/pool.d/www.conf (pool [www], serving Nextcloud at ej.onni.me).

Find	Change to	Why
process.priority = -19	process.priority = 0	A negative nice value lets PHP workers beat nginx and MariaDB for the CPU. On 2 threads that can starve the database that PHP is waiting for.
pm.max_children = 5	pm.max_children = 3	Hard cap on simultaneous PHP requests. Your workers use 63 to 79 MB each. With 5 workers at memory_limit 512M the worst case is 2.5 GB, more than the machine has; with 3 it is 1.5 GB (85% of RAM) and normal use is about 240 MB. Three workers are plenty for 2 threads and one Nextcloud.
pm.start_servers = 2	pm.start_servers = 1	Workers created at start; it must lie between min_spare and max_spare.
pm.max_spare_servers = 3	pm.max_spare_servers = 2	Above this many idle workers FPM kills the extras.

Leave pm.min_spare_servers = 1, pm.max_requests = 500, clear_env = no and the 512M memory_limit in php.ini (512M is what Nextcloud recommends). Add these lines at the end of the

file:

```
pm.status_path = /fpm-status-www
slowlog = /var/log/php8.4-fpm-slow.log
request_slowlog_timeout = 10s
```

`pm.status_path` lets the health script read live numbers straight from the socket (nothing is published through nginx). The slowlog writes a PHP stack trace for any request that runs longer than 10 seconds.

One decision for you: three timeouts disagree. The pool stops every PHP request after 300 seconds (`request_terminate_timeout`), nginx waits up to 3600 s (`fastcgi_read_timeout`) and `php.ini` allows 21600 s (`max_execution_time`). The 300 wins. Normal uploads use short chunks and are not affected, but downloading a multi-gigabyte file through the web page can take longer than 5 minutes and would be cut. If you do that, set `request_terminate_timeout = 3600`; otherwise keep 300, which protects your three workers from stuck requests.

Apply:

```
sudo php-fpm8.4 -t && sudo systemctl reload php8.4-fpm
```

After a few days check max children reached in the health script. If it is above 0, raise `pm.max_children` by 1.

5. OPcache

File: `/etc/php/8.4/fpm/conf.d/10-opcache.ini`. Replace its contents with:

```
zend_extension=opcache.so
opcache.enable=1
opcache.enable_cli=0
opcache.memory_consumption=128
opcache.interned_strings_buffer=16
opcache.max_accelerated_files=16229
opcache.save_comments=1
opcache.validate_timestamps=1
opcache.revalidate_freq=60
opcache.jit=disable
opcache.jit_buffer_size=0
```

- `opcache.jit=tracing` with a 100 MB buffer to disabled. The tracing JIT helps CPU-bound code, while Nextcloud waits on the database and disk, and the N3060 is slow at compiling. It also reserves 100 MB of your 1.8 GB. This is my recommendation, not a measurement: to test it, switch it back on and compare the CPU load.
- `opcache.max_accelerated_files` 100000 to 16229. PHP rounds up to a prime from this list: 3907, 7963, 16229, 32531, 65407, 130987. 100000 became 130987 slots for a single Nextcloud that needs about 10 000 to 15 000. Count yours: `find /var/www/ej.onni.me -name '*.php' | wc -l`, and if it is above about 13000 use 32531.
- `opcache.interned_strings_buffer` 8 to 16. Nextcloud's admin page warns when this buffer is nearly full, and 16 MB is cheap.
- `memory_consumption` stays at 128 MB; the health script shows the real use and warns above 85%.
- `validate_timestamps=1` (PHP's default) with `revalidate_freq=60` lets Nextcloud updates take effect within a minute.

Apply with `sudo systemctl restart php8.4-fpm`.

6. MariaDB (host service, version 10.11)

Your database is tiny: ej (Nextcloud) is 59.5 MB, of which `oc_filecache` is 29 MB, plus 3.7 MB of system data. MariaDB itself uses 75 MB. In 2.8 days it answered 160 707 queries, read from disk only 0.065% of the time, and never had more than 3 connections. The 128 MB buffer pool holds all the data and only 43 MB of it is in use, so leave `innodb_buffer_pool_size` as it is.

File: `/etc/mysql/mariadb.conf.d/50-server.cnf`, section `[mysqld]`. Your current lines there are `bind-address`, `slow_query_log`, `long_query_time = 2`, `expire_logs_days`, `character-set-server`, `collation-server` and `innodb_log_file_size = 256M`. Add these lines below `innodb_log_file_size`:

```
tmp_table_size = 32M
max_heap_table_size = 32M
key_buffer_size = 8M
aria_pagecache_buffer_size = 16M
max_connections = 30
thread_cache_size = 8
```

Line	Why
------	-----

tmp_table_size and max_heap_table_size = 32M	1 303 of 4 562 temporary tables (29%) were written to disk because they exceeded 16M, and on this host that disk is the eMMC. The two settings must be raised together because the smaller one is the limit. Memory is taken only when a query really needs it. Tables with BLOB or TEXT columns always go to disk, so the ratio will fall but not reach 0.
key_buffer_size = 8M	Only MyISAM tables use it; all yours are InnoDB. Default is 128 MB.
aria_pagecache_buffer_size = 16M	MariaDB's own system tables use Aria; the 128 MB default is far more than they need.
max_connections = 30	You never used more than 3. A lower cap limits the worst-case memory of runaway connections.
thread_cache_size = 8	The default of 151 keeps up to 151 idle threads ready; you create threads rarely.

Apply the three dynamic ones live, then restart once so the file is proven to work:

```
sudo mariadb -e "SET GLOBAL tmp_table_size = 33554432; SET GLOBAL max_heap_table_size =
33554432; SET GLOBAL key_buffer_size = 8388608;"
sudo systemctl restart mariadb
sudo mariadb -e "SELECT @@tmp_table_size/1048576 AS tmp_mb, @@key_buffer_size/1048576 AS
key_mb, @@max_connections;"
```

Nextcloud is unavailable for a few seconds during the restart.

Optional decision, a trade-off: innodb_flush_log_at_trx_commit is 1, so every commit forces a write and sync to the eMMC. Setting it to 2 writes at each commit but syncs only about once per second, which means fewer flash writes and faster commits. The cost: if the whole machine loses power or the kernel crashes, you can lose the last second of transactions (a MariaDB crash alone loses nothing). For one personal Nextcloud many people accept this; keep 1 if you prefer maximum safety.

The slow query log is DEBRiS-slow.log in the data directory (3 queries so far). The MariaDB error log goes to the journal: use journalctl -u mariadb -n 50.

7. Redis (host)

Redis is working well: 1.5 MB of memory and a 95.8% hit rate (575 558 hits, 25 055 misses). Only a limit is missing. In /etc/redis/redis.conf add:

```
maxmemory 32mb
maxmemory-policy volatile-lru
```

32 MB is more than twenty times current use and keeps a runaway cache from eating RAM that PHP and MariaDB need. volatile-lru deletes only keys that expire, which protects Nextcloud's file locks. Apply live, then the file edit keeps it after a restart:

```
redis-cli -s /run/redis/redis-server.sock CONFIG SET maxmemory 32mb
redis-cli -s /run/redis/redis-server.sock CONFIG SET maxmemory-policy volatile-lru
```

Optional: Redis periodically saves a snapshot to /var/lib/redis on the eMMC. If you only use it for Nextcloud's cache and locks (nothing there needs to survive a restart), add save "" to redis.conf to turn the snapshots off. The effect is small: a few kilobytes at a time.

8. Docker

8.1 Daemon settings and log rotation

File: /etc/docker/daemon.json. It currently contains only the data-root. Keep that line, because it points Docker at your hard disk, and add the others:

```
{
  "data-root": "/home/app-data/.docker",
  "live-restore": true,
  "log-driver": "json-file",
  "log-opts": { "max-size": "10m", "max-file": "3" }
}
```

- live-restore is false now, so every Docker restart or package upgrade stops all 3 containers; with true they keep running.
- Without max-size each container log grows without limit, on your hard disk.

Apply with `sudo systemctl restart docker`. This one restart still stops the containers (live-restore was off); they start again by themselves. The log limits only apply to containers created afterwards, which the next step does.

8.2 Memory limits, per container

The compose files are /home/app-data/n8n.ej.onni.me/docker-compose.yml and /home/app-data/music.ej.onni.me/docker-compose.yml. Add the options under each service entry, at the same indent as restart: (spaces only, no tabs). Find the entry by its container_name.

Option	What it does
mem_limit: 512m	Hard ceiling for the container's RAM. Above it the kernel stops the biggest process in the container (shown as OOMKilled) and the restart policy starts it again. A limit is a ceiling, not a reservation.
memswap_limit: 512m	RAM plus swap in total. The same value as mem_limit means this container may not use swap, so a runaway job fails fast instead of slowing the whole machine.
cpus: 1.5	Caps the container at 1.5 of your 2 threads, leaving room for nginx and MariaDB.
oom_score_adj: 500	If the whole host runs out of memory, the kernel picks this container before MariaDB, nginx or PHP.

Suggested values (usage from your docker stats):

Compose file	container_name	Used now	mem_limit	Extra
n8n	n8n.ej.onni.me	313 MB	512m	memswap_limit: 512m, cpus: 1.5, oom_score_adj: 500, NODE_OPTIONS (below)
n8n	db.n8n.ej.onni.me (Postgres 17)	40 MB	256m	command line (below)
music	music.ej.onni.me (Navidrome)	29 MB	256m	memswap_limit: 256m, cpus: 1

Example for the n8n entry in /home/app-data/n8n.ej.onni.me/docker-compose.yml. Only the marked lines are new; the file uses the list style (- KEY=VALUE) under environment:, so add the line in the same style:

```
n8n:
  image: n8nio/n8n
  container_name: n8n.ej.onni.me
  restart: always
  mem_limit: 512m           # new
  memswap_limit: 512m      # new
  cpus: 1.5                 # new
  oom_score_adj: 500        # new
  ports:
    - "127.0.0.1:5678:5678"
  environment:
    - DB_TYPE=postgresdb
    # ... your other lines stay as they are ...
```

```
- NODE_OPTIONS=--max-old-space-size=384 # new
```

NODE_OPTIONS=--max-old-space-size=384 tells n8n's JavaScript engine to start cleaning up memory when it reaches 384 MB, a little below the container limit, so it slows down instead of being killed. For the Postgres entry add mem_limit: 256m and, at the same indent, this line, which shrinks Postgres's default cache to fit the small RAM:

```
command: postgres -c shared_buffers=64MB -c max_connections=20
```

For the Navidrome entry in /home/app-data/music.ej.onni.me/docker-compose.yml add mem_limit: 256m, memswap_limit: 256m and cpus: 1. Navidrome reads your whole music folder every 6 hours (ND_SCANSCHEDULE: 6h); if the library rarely changes, 24h means fewer disk scans. Transcoding on the fly is the expensive part for the N3060, so use direct play in your music apps where you can.

Optional, to keep n8n's database small and write less to the hard disk: add - EXECUTIONS_DATA_MAX_AGE=168 under environment: to keep 7 days of execution history instead of the default 14, and - EXECUTIONS_DATA_SAVE_ON_SUCCESS=none to stop saving successful runs. The cost of the second one is that you can no longer inspect past successful runs (failed runs are still saved).

Apply each stack in its own folder, one at a time (the app restarts for a few seconds):

```
cd /home/app-data/n8n.ej.onni.me && docker compose config -q && docker compose up -d
cd /home/app-data/music.ej.onni.me && docker compose config -q && docker compose up -d
docker inspect n8n.ej.onni.me --format 'mem={{.HostConfig.Memory}}
swap={{.HostConfig.MemorySwap}}'
docker stats --no-stream
```

docker compose config -q prints nothing when the YAML is valid. For mem_limit: 512m the inspect line should show mem=536870912 and swap=536870912. After a few days check whether any limit was hit:

```
docker inspect -f '{{.Name}} oom={{.State.OOMKilled}} restarts={{.RestartCount}}' $(docker ps
-aq)
```

OOMKilled true, or a restart count that keeps rising, means that limit is too low; raise it by half.

9. nginx and Nextcloud

nginx needs one small change that helps the eMMC. In /etc/nginx/nginx.conf, in the http block, change the line access_log /var/log/nginx/_access.log; to:

```
access_log /var/log/nginx/_access.log combined buffer=32k flush=5m;
```

Without `buffer=` nginx writes every request to the flash as a separate small write; with it, requests are collected and written in blocks, as your site logs already do. Test with `sudo nginx -t`, then `sudo systemctl reload nginx`. HTTP/3 stays off, which is right for this CPU.

Optional, to ease the N3060: make Nextcloud create smaller previews. These commands only change settings (adjust the path if your Nextcloud root differs):

```
sudo -u www-data php /var/www/ej.onni.me/occ config:system:set preview_max_x --value=1024 --type=integer
sudo -u www-data php /var/www/ej.onni.me/occ config:system:set preview_max_y --value=1024 --type=integer
sudo -u www-data php /var/www/ej.onni.me/occ config:app:set preview jpeg_quality --value=60
```

Preview generation is one of Nextcloud's most CPU-hungry jobs; 1024-pixel previews at quality 60 cost a fraction of the default size and take less space on disk. The cost is slightly softer thumbnails in the web interface.

10. The adapted health script

`php_health_debris.sh` is the PODiUM script adapted for DEBRiS. It reads the `www` pool's socket and settings from `www.conf`, warns when `max_children` times `memory_limit` exceeds RAM, shows memory pressure (PSI) and the swap devices (so you can see `zram`), warns about `n8n` and `Navidrome` without a memory limit, and adds two eMMC rows: the chip's wear estimate (when the kernel reports it) and how many megabytes are written to the eMMC per day since boot, with a warning above 3000 MB/day.

Install (same way as on the other servers):

```
sudo apt install libfcgi-bin
sudo install -m 755 php_health_debris.sh /opt/scripts/analyze_php.sh
sudo /opt/scripts/analyze_php.sh
```

Before your edits expect a FAIL for the `www` status page (until `pm.status_path` is added) and WARN rows for worst-case PHP RAM (5 x 512M is 141% of RAM), `process.priority` and the containers without `mem_limit`. After the edits these should turn OK. The eMMC writes per day row is a before and after measure for sections 3 and 6: run the script once now, make the changes, and compare after a day of normal use.

11. Rollback and what I did not see

Every file has a .bak copy from section 2. Restore it and reload the service. For sysctl delete /etc/sysctl.d/99-debris.conf and reboot; to remove zram, sudo apt remove zram-tools; to bring the services back, systemctl enable --now with the same unit names.

Not in your report, so unchecked: Nextcloud's config.php, the firewall (check sudo ufw status verbose once), the contents of /etc/fstab, the eMMC wear value, and how much the VS Code server process on this machine uses while you are connected. The numbers (3 workers, 512m for n8n, 32M temp tables) are sensible starting points for this hardware, not measured optima. Re-run the script after a day of normal use, and again after a week.

Revision #1

Created 2026-10-26 13:46:20 UTC by meson

Updated 2026-10-26 13:46:40 UTC by meson