StorageDiagLV Workbench

User Guide — Silas QLog Analyzer

Version 1.4.08062 · 2026-08-06

Unofficial tool — not a QNAP product.

The primary, always-current guide lives inside the application.

OVERVIEW AND HOW TO USE THIS GUIDE

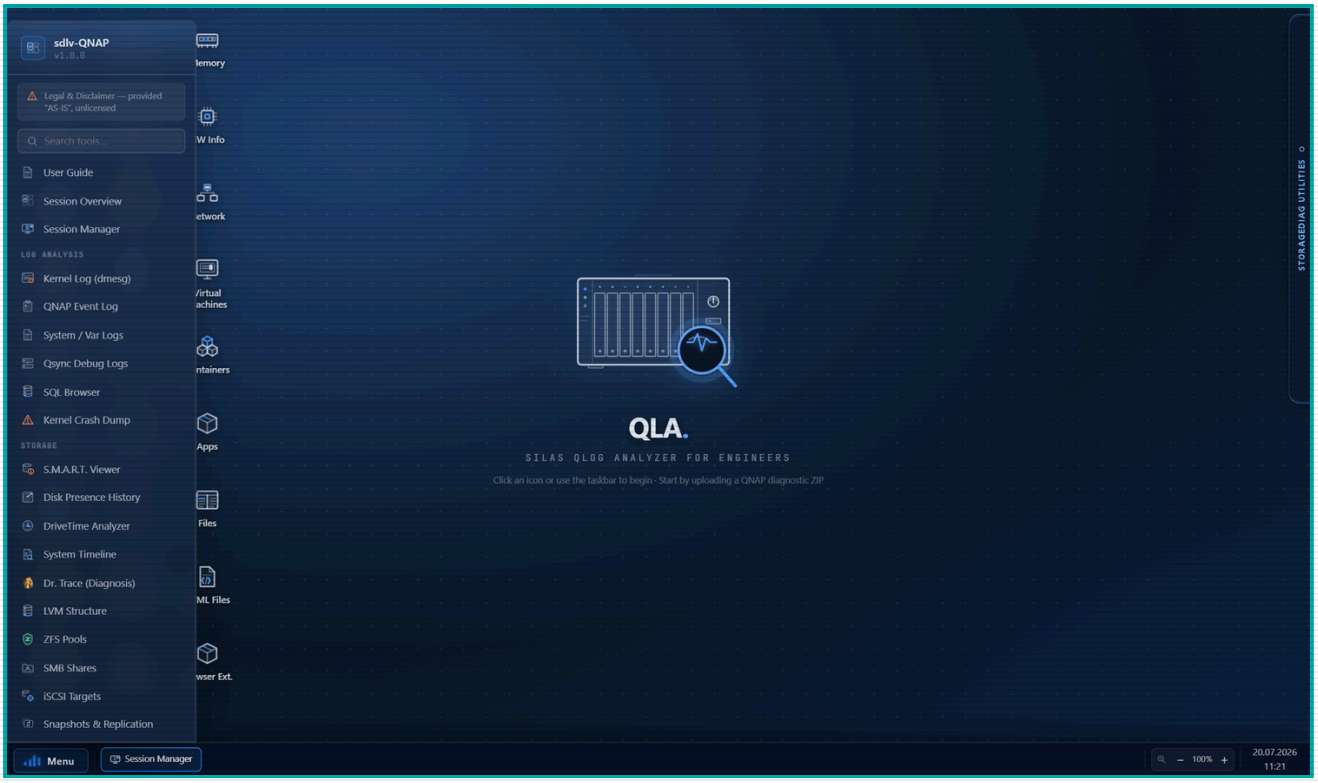
> **Important — what this is.** StorageDiagLV is **not an official QNAP product** and is for **analysis only**. It sends **nothing to the QNAP cloud** and uses **no AI** unless an administrator explicitly configures one. Its findings are pattern-based **leads to verify** — never act destructively on its word alone. Full details, components, privacy and licence: 0099. About, privacy and licence.

What StorageDiagLV is

StorageDiagLV is a web tool that turns a QNAP NAS [illegible] into a readable, prioritized health picture. You upload the bundle a customer or a support case provides; the tool extracts it, recognizes the logs inside, and opens a set of windowed viewers that colour-code what matters, flag known problems, and correlate events across the whole system into a plain-language verdict.

It is built for [illegible] — people who need to read a NAS's logs quickly and reach a defensible conclusion, without hand-grepping multi-megabyte kernel dumps.

> Beyond this passive analysis of an uploaded bundle, StorageDiagLV can also run **live diagnostics** against a NAS (I/O benchmark, memory test, network throughput, etc.). Those live tools are covered in their own chapters later; this guide starts with the everyday workflow: read a bundle.



The workspace at a glance — the tools menu on the left (grouped Log Analysis · Storage · …), the desktop icons, and the utilities rail down the right edge. Everything starts by uploading a diagnostic bundle.

Prefer to watch first? The video walkthrough

There is a [redacted] of the whole workflow — a real support case followed from the uploaded bundle, through the verdict, to a memory test on the customer's NAS (and the twist the logs were hiding). It is narrated in six languages.

- ▶ **Watch it in the browser** — with a language switcher: English · Polski · Français · Čeština · Italiano · 繁體中文.
- Or download the video file: English · Polski · Français · Čeština · Italiano · 繁體中文

The video is a tour, not a reference — after watching it, the chapters below tell you what each window actually shows and how to read it.

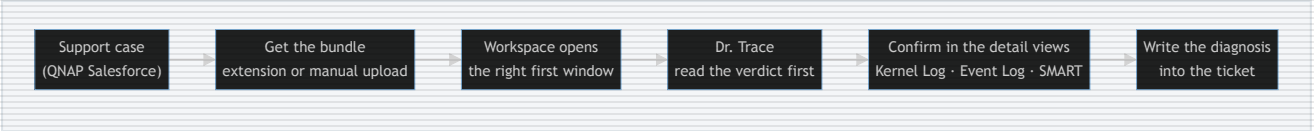
Prefer to be walked through it? The guided how-to

If you would rather learn by doing than by reading, the [redacted] takes you through the real windows a step at a time. Each step opens the window it is about, puts a spotlight on the control being discussed, and gives you one sentence about it. Back and Next move through the tour, and the panel can be dragged out of the way.

@@TOUR:case|Walk me through diagnosing a case@@ @@TOUR:disks|Walk me through judging a disk@@ @@TOUR:memory|Walk me through a suspected memory fault@@

It is also on the desktop and in the Start menu as . The chapters below stay the reference: the tour shows you where things are, the chapters explain how to read what you find.

The everyday workflow



1. – pull it straight from a Salesforce case with the browser extension, or upload it by hand. (Chapters 0002. Integrate Salesforce with Browser Extension, 0003. Uploading logs.)
2. – the desktop, windows, and sessions. (0004. The workspace.)
3. – open ; it correlates the kernel and event logs into ranked incidents with a likely cause and a "how to recover" path. (0007. Reading Dr. Trace (the incident timeline).)
4. – drill into the , , and to verify the verdict with the raw evidence. (Chapters 0005, 0006, 0008.)
5. – write up what happened and what to do. (0009. A worked example — diagnosing a case end to end.)

Key concepts (read these once)

- – one uploaded bundle. It is private, lives on the server, and expires automatically. One session can hold several device dumps (e.g. both NAS units of an HA pair).
- – how serious a line is. It comes from two places at once (the level the kernel/app stamped on the line, and the category of any matched pattern) and the tool always shows the of the two. The scale, worst to least: .
- – a saved signature for a known problem, tagged critical/warning/info. Patterns add meaning: a matched line gets a coloured pill and, when documented, a 📖 link to a Knowledge-Base article. You manage them in the .
- – the opposite of a pattern: it marks a benign, expected line so it stops competing for your attention. Ignore rules only subtract noise; they never change a real match or a line you marked by hand.
- – two controls that shrink a huge log: keeps only the lines that stand out, collapses repeats into ×N groups.

Recommended reading order

For a newcomer: read [redacted] in order to get set up, then read the log chapters in this order, which builds from simplest to most powerful:

1. 0005. Reading the QNAP Event Log – the friendliest log (one severity source, no ignore/dedup).
2. 0006. Reading the Kernel Log (dmesg) – the core skill: severity, patterns, ignore, Focus, dedup.
3. 0007. Reading Dr. Trace (the incident timeline) – the two logs correlated into a verdict.
4. 0008. Reading S.M.A.R.T. disk health – judging individual drives.
5. 0009. A worked example – diagnosing a case end to end – everything together on a real case.

[redacted] (read as needed for a specific subsystem): 0011. Storage layout – LVM, RAID and DRBD · 0012. ZFS pools (QuTS hero) · 0013. Memory, OOM and crash dumps · 0014. Network interfaces · 0015. Hardware info – sensors, DMI, USB and apps · 0016. Pattern Editor and Regex Helper · 0017. Live diagnostics and Doctor utilities · 0018. Disk Presence History · 0019. Virtual Machines · 0020. Container Station · 0021. Snapshots and Replication · 0022. System Timeline · 0023. iSCSI · 0024. Shares (SMB) · 0025. Process Explorer · 0026. SQL Browser · 0027. HBS 3 – backup jobs and logs · 0028. Regex Cookbook · 0029. Troubleshooting · 0030. Glossary · 0031. User management and roles · 0032. DriveTime Analyzer · 0033. memhound – the RAM tester · 0034. System and Var Logs · 0035. Manual Disk Analyzer · 0036. Qsync and app-log bundles · 0037. File Browser and HTML Reports. And 0099. About, privacy and licence – what the tool is, privacy, components, and the licence.

> **Tip.** In day-to-day work you'll do it the other way round – **verdict first (Dr. Trace), detail second**. This reading order is for learning the tool; the worked example in chapter 0009 shows the faster working order.

Watch it

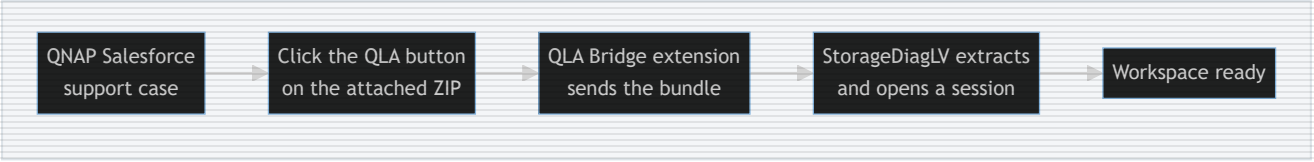
> In-app demo: Moving through this guide inside the app.

> In-app demo: The Workbench is a QPKG – it lands on the QTS desktop and in the App Center like any other app.

INTEGRATE SALESFORCE WITH THE BROWSER EXTENSION

The QLA Bridge browser extension links StorageDiagLV to QNAP's Salesforce support console. Once installed, a 🔬 QLA button appears next to every diagnostic archive attached to a case — one click uploads that bundle straight into StorageDiagLV, so you skip the download-then-re-upload round trip.

> **You install the extension once per browser.** After that it works on every case.



Get your personalized extension package

Log in to StorageDiagLV, click the Extensions icon in the toolbar, and download your personalized `.zip`. The package is tied to your analyzer URL, so the extension already knows where to send bundles.

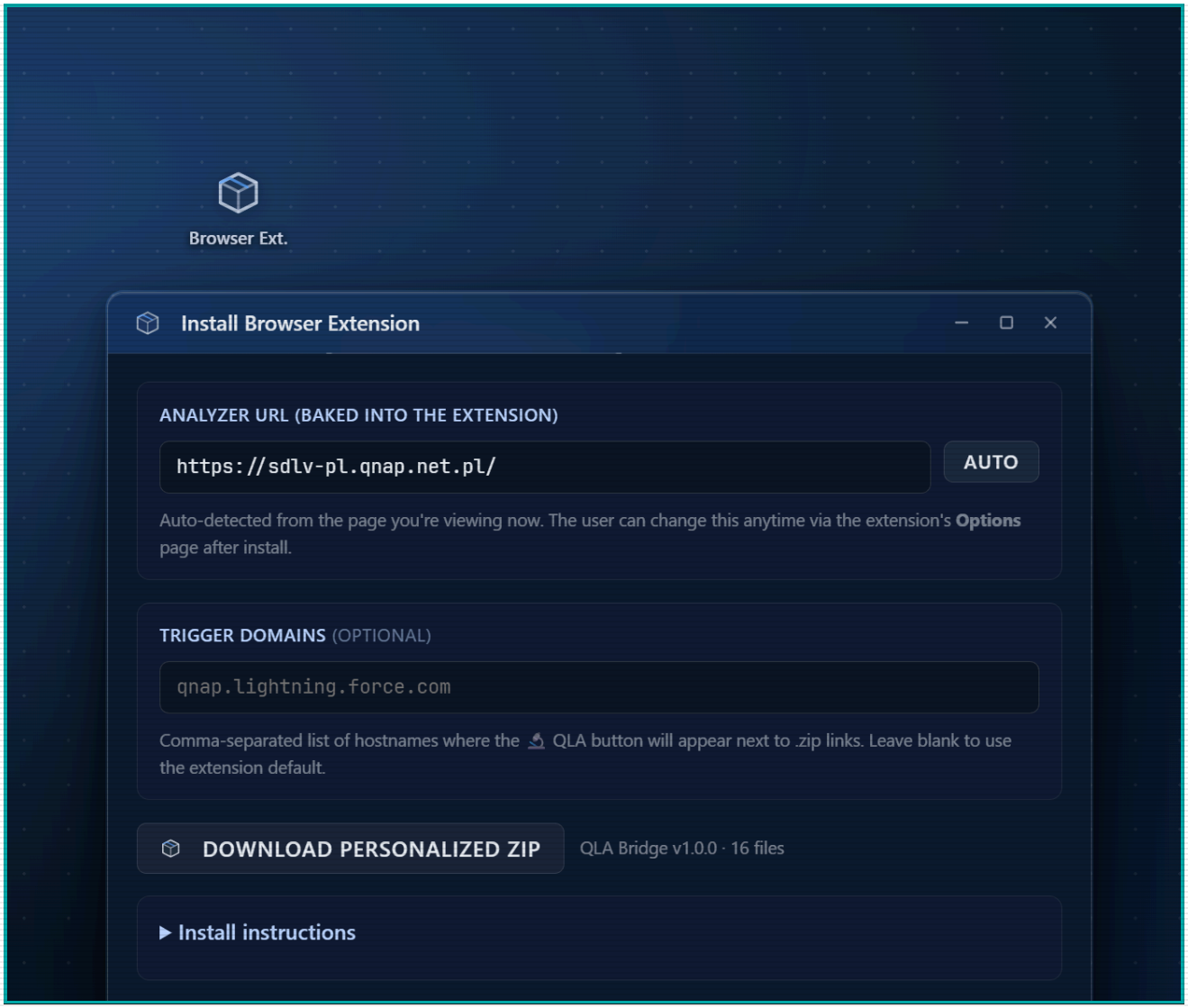


Figure — Downloading the personalized QLA Bridge package from StorageDiagLV.

Install (Chrome / Edge) — recommended

1. [illegible] (e.g. `C:\SDLV`).
   Chrome loads the extension from this folder every time it starts — do not move or delete it.

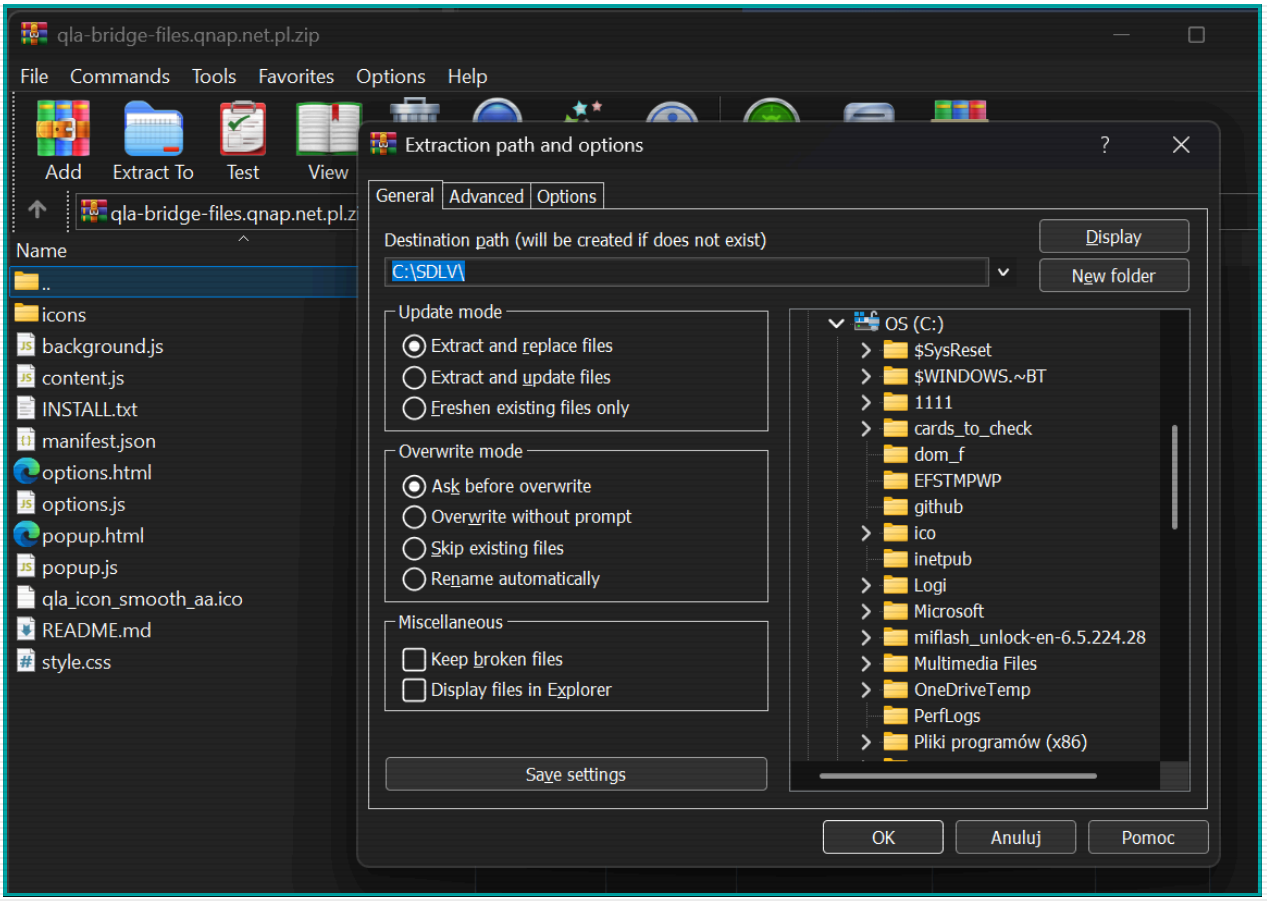


Figure — Extract the package to a folder you will keep.

2. Open `chrome://extensions` (or `edge://extensions`).
3. Turn on (top-right toggle).
4. Click and select the extracted folder.
5. Done — the icon appears in your toolbar.

> **Tip.** Pin the icon (puzzle-piece menu → pin) so it's always visible while you work cases.

Install (Safari — macOS 13.3+, Safari 16.4+)

the way Chrome and Edge can. On Apple's platform a web extension must be wrapped in a small macOS app, built once with Apple's command-line converter (part of the free). There is no way around this — and no pre-built app is shipped, so you build it yourself once, or simply use Chrome/Edge.

You therefore have two options:

- The extension and the QLA button work identically there, with no build step. If Safari is not a hard requirement, use this.

- (needs a Mac + the free Xcode Command Line Tools — you do not need the full Xcode IDE for the build itself, only the tools):
  1. Install the tools: `xcode-select --install`
  2. Convert the extracted extension folder into an app project:

     ```
     xcrun safari-web-extension-converter /path/to/extracted-extension \
       --app-name "QLA Bridge" --bundle-identifier com.qla.bridge
     ```

  3. Build and launch the generated app once (this registers the extension with Safari).
  4. In , enable . For a local (unsigned) build, first enable (it resets on each Safari launch).
  5. On a case page, grant the extension access to the QNAP Salesforce domain ("Always Allow on This Website").

> **The extension source is Safari-ready** — it uses the standard cross-browser WebExtensions API, so the converter runs cleanly. The full step-by-step build is in `extension/SAFARI.md`.

Using it on a case — the 🔬 QLA button

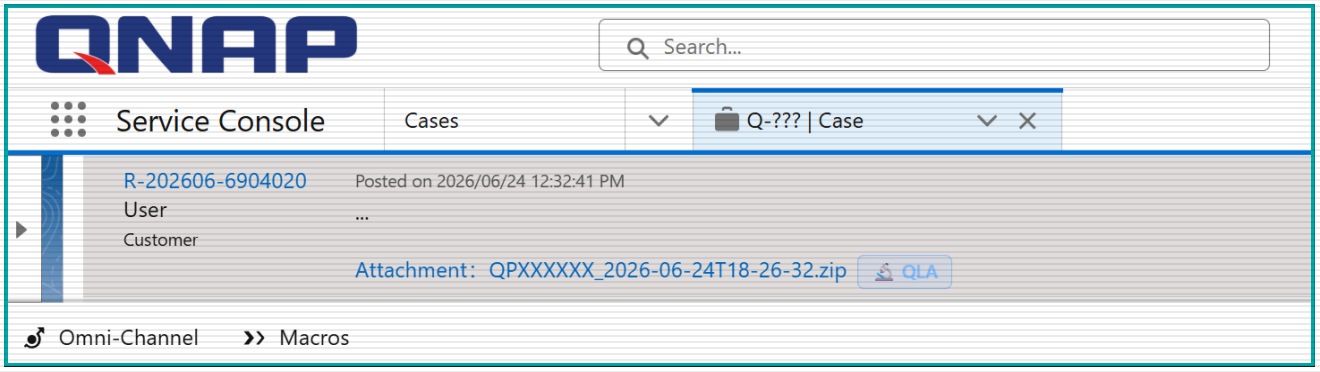


Figure — A QNAP Salesforce case: the 🔬 QLA button next to the diagnostic `.zip` attachment.

Once the extension is installed, open — — a QNAP Salesforce case that has a diagnostic bundle attached. The extension scans the page and adds a 🔬 button next to every diagnostic archive it recognizes (`.zip`, `.tar.gz`, `.tgz`, plus named app-log bundles).

> **Don't see the button? Reload the page.** Salesforce loads case content dynamically; the button appears once the attachment is on screen.

> Reloading the case page after opening it is the reliable trigger.

Click 🔬 [illegible]. The extension then:

1. downloads the bundle from Salesforce using your existing login,
2. uploads it directly to your StorageDiagLV instance, and
3. opens the analyzer in a new browser tab with the session ready — you land in the right first viewer.

No manual download, no re-upload. From here, continue with 0003. Uploading logs.

> **Options.** Click the extension's toolbar icon → **Options** to change the analyzer URL, the trigger domains (which sites show the button), or the maximum file size.

Watch it

> In-app demo: Installing the QLA Bridge extension, personalised for your Salesforce console.

> In-app demo: Pointing the bridge at your own Workbench instead of the public demo, per trigger domain.

> In-app demo: The public demo server — the whole passive analysis, with the live tools switched off.

UPLOADING LOGS

There are two ways to get a diagnostic bundle into StorageDiagLV.

Where the bundle comes from

A "diagnostic bundle" is the ZIP that QNAP's own [illegible] generates. To make one on the NAS:

1. Open the NAS web interface.
2. [illegible].
3. Click [illegible].
4. Wait for it to finish (it can take several minutes).
5. Download the ZIP.

That ZIP is what you upload here — or what the 🔬 [illegible] button pulls straight from a Salesforce case.

A. Directly from Salesforce (via the extension)

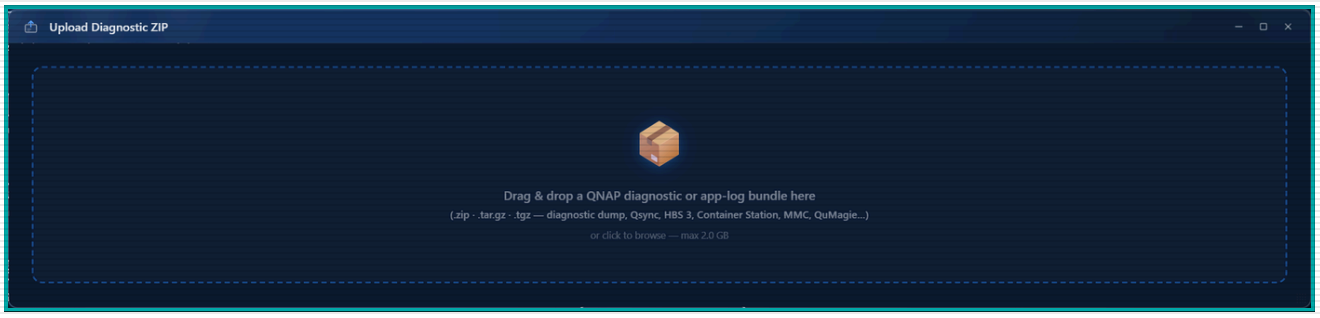
With the [illegible] extension installed (see 0002. Integrate Salesforce with Browser Extension), open a QNAP support case that has a diagnostic bundle attached and click the extension icon. The bundle is sent to your analyzer and a session opens automatically — no manual download required.

B. Manual upload

Drag a bundle onto the [illegible] window, or click the drop zone to browse for it.

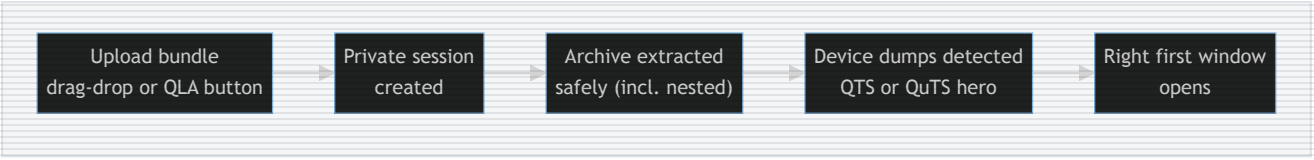
[illegible] `.zip`, `.tar.gz`, `.tgz`. These cover a standard QNAP diagnostic dump as well as per-app bundles (Qsync, HBS 3, Container Station, MMC, QuMagie…). Other extensions are rejected.

A progress bar shows the percentage transferred. The maximum size is computed on the server from free disk space and PHP limits; the browser does a courtesy pre-check before sending.



The Upload window — drag a bundle onto the drop zone, or click to browse. Accepted formats: `.zip`, `.tar.gz`, `.tgz`.

What happens after upload



1. **A private session is created** – each upload gets its own random session folder with a lifetime stamp. When the session expires it is cleaned up automatically; the success message tells you when.
2. **The archive is safely extracted** – including any archives nested inside it (with protection against path-traversal and decompression-bomb tricks).
3. **Device dumps are detected** – the extractor finds every kernel log and identifies one or more device dumps, detecting whether each is a QTS or QuTS hero system (this decides, for example, whether you get the ZFS or the LVM/DRBD viewers).
4. **The workspace opens the right first window** – the Overview tab for a normal diagnostic dump, or the app's dedicated viewer for an app-log bundle.

> In-app demo: Loading a bundle end to end – upload, extraction and parsing, then the workspace surfacing a machine-check (MCE) event caused by mixed RAM.

> **Note – one upload can contain several device dumps.** A bundle captured from an HA pair or an expansion chain may hold more than one NAS. The taskbar session indicator shows the count; use the **Session Manager** to switch between them (see 0004. The workspace).

> **Tip – sessions are shareable.** You can hand a session to a colleague on another machine with a resume link (Session Manager → share). The data stays server-side; the link just re-attaches them to the same extracted session until it expires.

Watch it

> In-app demo: Picking a diagnostic ZIP and handing it to the uploader.

> In-app demo: The Session Manager – resuming, clearing or sharing a bundle you already uploaded.

> In-app demo: Resuming an earlier bundle, sharing it, or starting again with a new ZIP.

THE WORKSPACE

StorageDiagLV presents a [illegible] : a surface with icons, a [illegible], and a [illegible]. Every analysis tool opens in its own window.

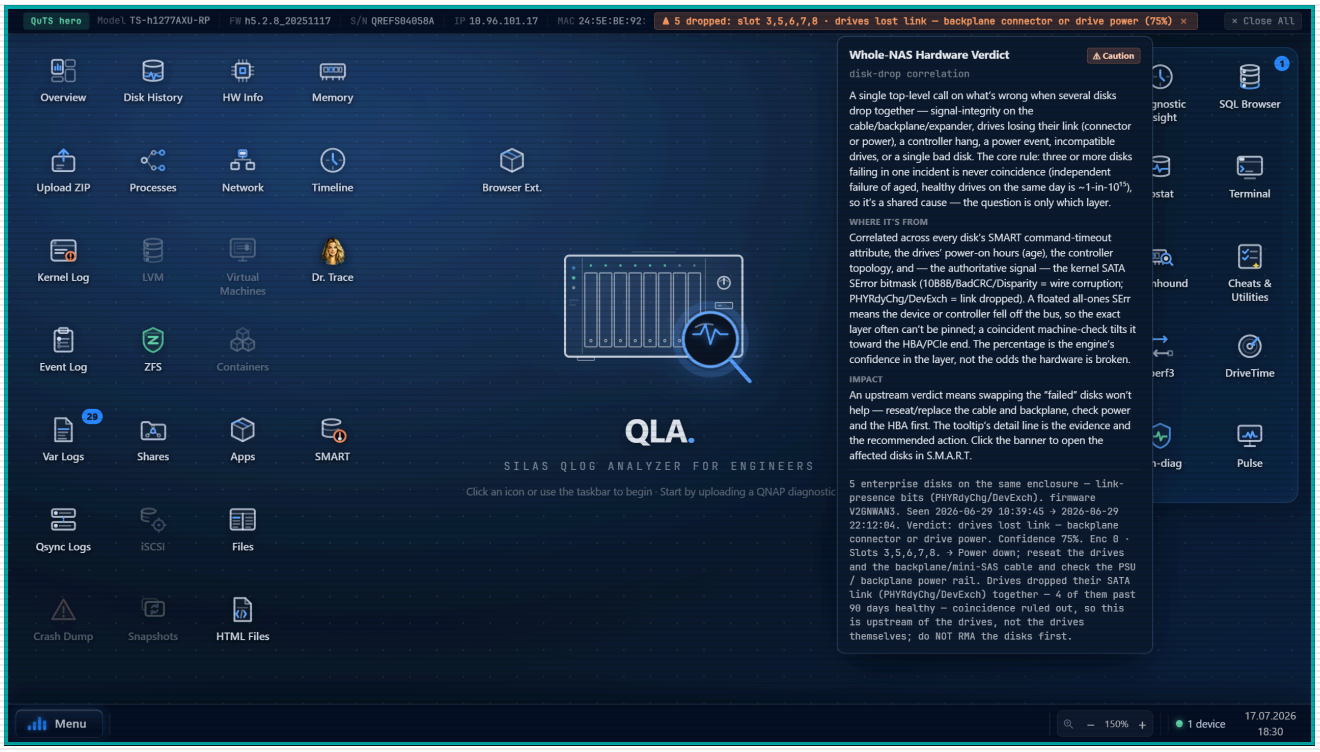


Figure — The StorageDiagLV desktop: the icon grid, taskbar, and a Dr. Trace whole-NAS verdict tooltip.

Opening tools

Open a tool from the [illegible] or by double-clicking its [illegible] — both do the same thing. Type in the Start menu's search box to filter tools by name, then press [illegible] to open the first match.

> **Greyed-out tools.** A tool is disabled (and won't open) when it doesn't apply to the current session — for example a ZFS viewer on a QTS (non-ZFS) dump, or a viewer whose data isn't present in the bundle. This is normal: the workspace only offers what the uploaded data supports.

Windows

Each tool is a draggable, resizable window with the usual [illegible] buttons and a matching [illegible].

Action	How
Move	Drag the title bar. Windows [illegible] to screen edges and to each other; hold [illegible] to move freely.
Maximize	Double-click the title bar, or the maximize button.

Action	How
Minimize / restore	The taskbar button toggles it: buried → front, on-top → minimized, hidden → restored.
Close front window	
Zoom the whole UI	+ / − / 0 (reset)

Re-opening a tool that's already open just brings it to the front — you never get duplicates.

Themes & preferences

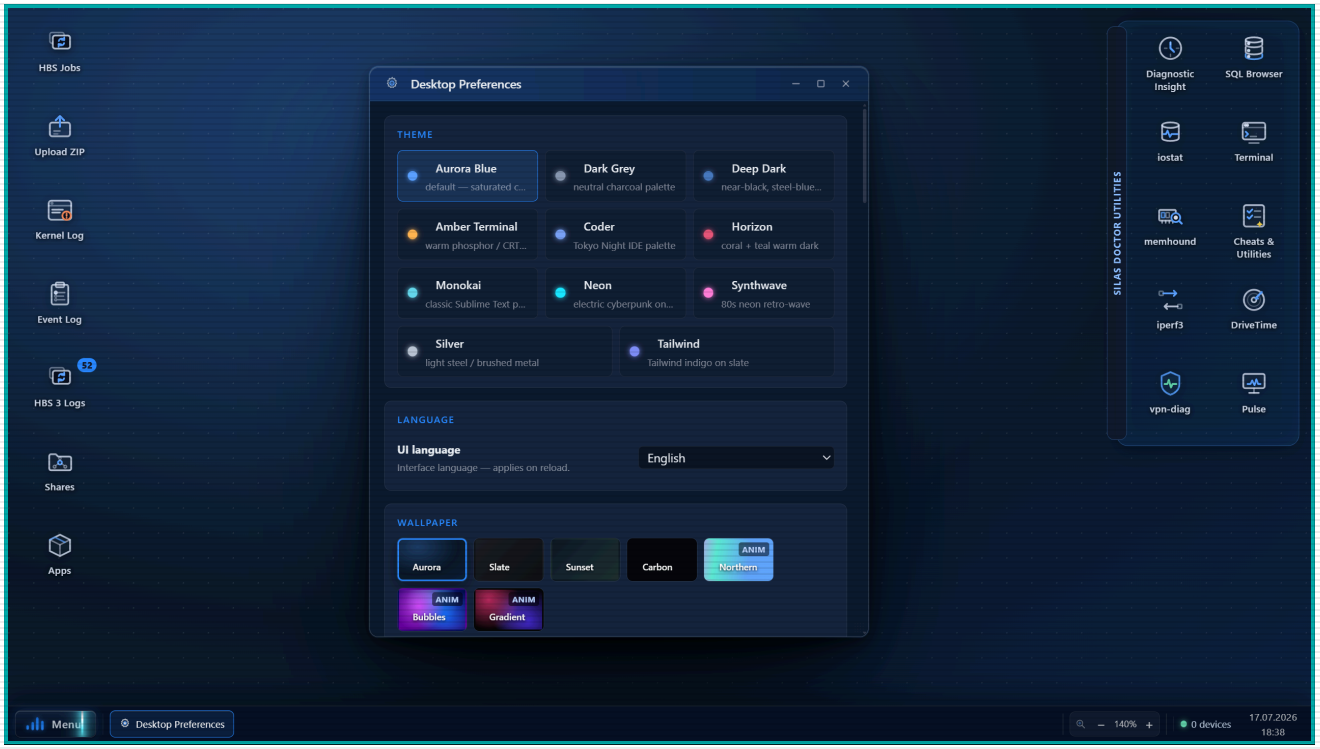


Figure — Desktop Preferences: colour theme, UI language, and a static or animated wallpaper.

Right-click the desktop (or) to change the (Default, Grey, High-Contrast, Amber, Coder, Horizon, Monokai, Neon, Synthwave, Silver, Tailwind), the , the , the font family/scale, blur effects, and the screensaver.

Sessions

Your uploaded data lives in a . The taskbar shows how many device dumps the active session holds; hover it for the session ID and device list, click it to open the .

From the Session Manager you can:

- between saved sessions (e.g. the two NAS units from one HA bundle).

- Share a session to another machine via a resume link (the data stays on the server).
- Close the current session — this closes every window and returns you to the upload screen.

> **Note.** When a session becomes active, the workspace jumps to the most relevant first window for that bundle type and dims the icons that have no data to show — so what's lit up is what's worth opening.

Watch it

> In-app demo: Session Overview — a verdict per module, read before opening any single viewer.

> In-app demo: The same Overview scoring the NAS it is installed on, with no bundle uploaded at all.

> In-app demo: Memory, swap, OOM kills, CPU and disk temperature and fan RPM on one panel.

READING THE QNAP EVENT LOG

The QNAP Event Log viewer shows the NAS application-level event log (`log_tool`) — the same System/Connection events you see in QTS/QuTS, but analyzed alongside your patterns. It is the human-facing counterpart to the Kernel Log: where the kernel log is what the hardware/kernel said, the event log is what QNAP's software recorded (volumes, shares, logins, apps, snapshots…).

> **How to open it:** Start menu → **Event Log**. It opens on the **newest** events (last page). Or from here: @@OPEN:Logtool|Open the Event Log@@

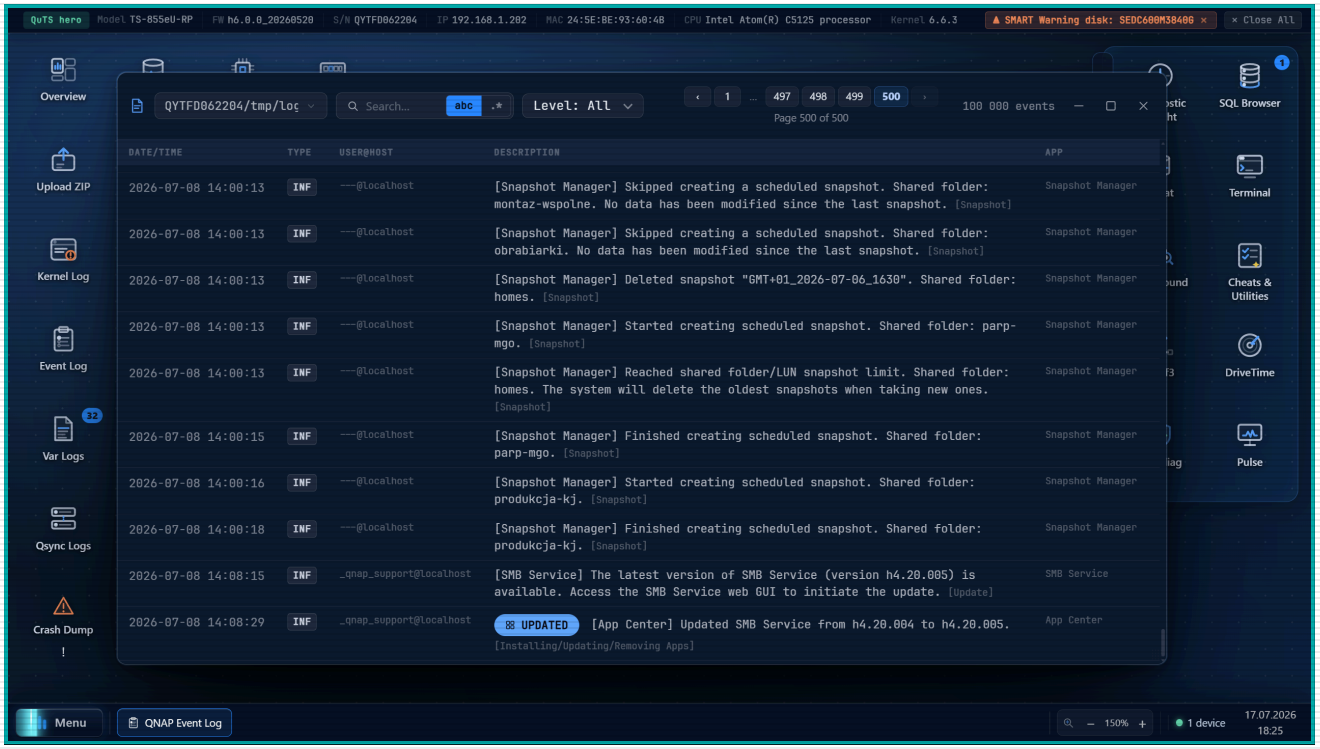


Figure — The QNAP Event Log: parsed system events (Snapshot Manager, SMB Service, App Center) with severity, source and app columns.

Columns

Column	Meaning
Date / Time	When the event was recorded
Type	Severity — `INF` (info), `WRN` (warning), `CRT` (critical)
User@Host	The user and source IP/host that triggered it
Description	The event text, plus any pattern badges and its `[category]`
App	The QNAP app/component that raised it

Colour — simpler than the Kernel Log

The row colour comes [illegible] : `CRT` is red, `WRN` amber, `INF` neutral.

> **Important — this differs from the Kernel Log.** In the Kernel Log a matched detection pattern can raise a line's colour. In the Event Log it does **not**: a matched pattern shows as a badge/pill on the Description, but the row colour always reflects QNAP's own event Type. So read the Type column as the authority on severity here.

Pattern badges

Detection patterns whose [illegible] is `logtool` or `both` are matched against the [illegible] text (only the rows on the current page are matched). A match can appear as:

- a [illegible] at the start of the description (for "event-action" patterns that carry a verb label, e.g. `Created` / `Removed` / `Degraded`), or
- an [illegible] for ordinary patterns.

These tell you *"this event matches a known signature"* — click a 📖 badge for the Knowledge-Base write-up.

Finding events: Search + Level

- [illegible] — filters as you type; toggle [illegible] for PCRE. Search matches the [illegible] only (not user, host, or app).
- [illegible] — `All`, `≤ Warning`, or `Info only`. Note `Info only` shows *only* INFO rows (not "info and below").

> **Note — no Ignore or Dedup here.** Unlike the Kernel Log and Dr. Trace, the Event Log does not apply ignore rules and does not collapse repeats into ×N groups. Every matching event is shown as its own row. To see repeated events correlated and de-duplicated, use **Dr. Trace**, which merges this log with the kernel log (next-but-one chapter).

How to read it

1. Open the Event Log; it lands on the newest events.
2. Set [illegible] to drop the routine INFO chatter.
3. Scan the [illegible] column — reds and ambers first — and read the [illegible] + [illegible] to see what subsystem is unhappy (volume, pool, share, HA, app…).
4. Note the [illegible] on security-relevant events (logins, permission changes).

5. Cross-reference the timestamp with the [illegible] and [illegible] : an app-level "volume degraded" event usually has a matching kernel/disk story underneath it.

Next: 0006. Reading the Kernel Log (dmesg).

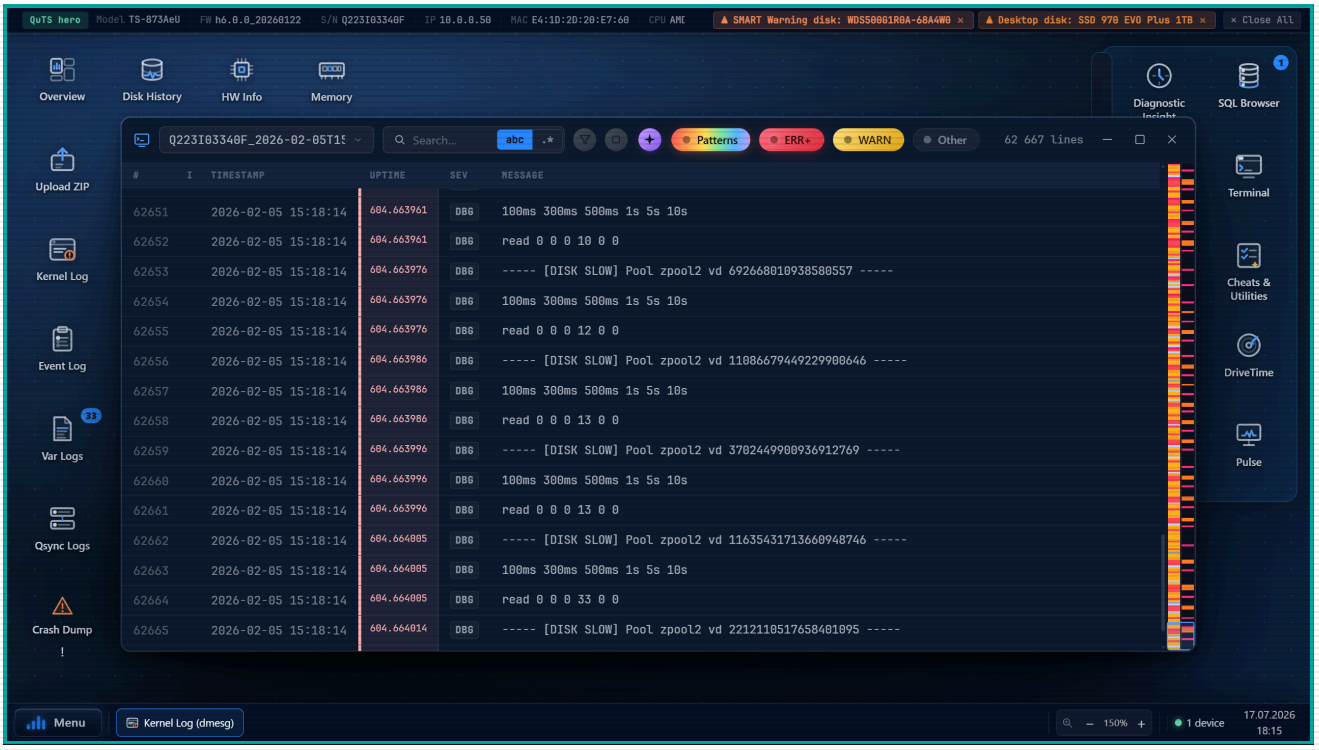
Watch it

> In-app demo: The QNAP Event Log with each entry carrying its category icon.

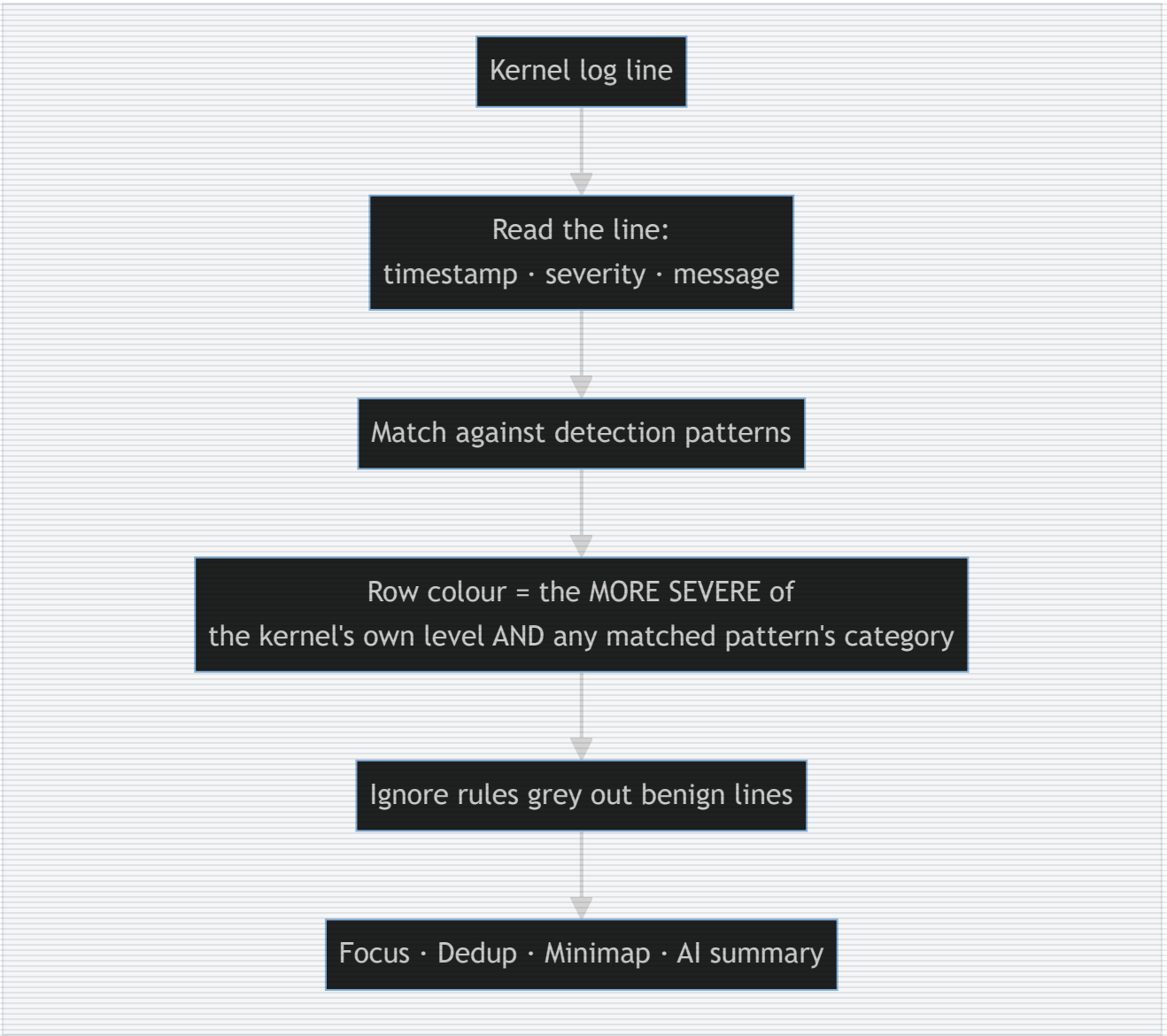
READING THE KERNEL LOG (DMESG)

The Dmesg Log window shows the NAS kernel ring buffer (`klogd_dump.log`) as a readable, colour-coded table. It is where most hardware and storage faults surface first. This chapter explains how to read it — what the colours mean, what the badges tell you, and how to cut a huge log down to the few lines that matter.

> **How to open it:** Start menu → **Kernel Log**, or from here: @@OPEN:Dmesg|Open the Kernel Log@@ The window opens on the **last page** (the newest lines — what is happening now), not the top.



How the analysis works



The colour rule in one line — the row always takes the worst of the two severity signals:

```
row colour  =  min( min(severity, 5),  catTint )
```

where is the kernel's own syslog level (lower = worse) and is for a critical-pattern match, for warning, or when nothing matched — so the lower number wins, and a pattern can only raise a line's severity, never hide it.

1. Anatomy of a row

Each line has six columns:

Column	Meaning
#	Line number in the file
⊘	Ignore marker (shown only on lines matched by an ignore rule)
time	Wall-clock timestamp (reconstructed from the boot anchor for early boot lines)
monotonic	Seconds since boot ([4848286.74])
level	Severity badge — the level the kernel itself stamped on the line
message	The kernel message, with pattern pills and decode panels appended

---

2. Colour = severity, and it comes from two places

The row's colour is its . Two independent things decide it, and the log always shows the of the two:

1. — every kernel line carries a syslog level:

Badge	Level	Meaning
EMERG / ALERT / CRIT	0–2	system is/near unusable —
ERR	3	an error occurred
WARN	4	warning
NOTE	5	notice
INFO	6	informational (the default when a line has no level)
DBG	7	debug

2. — a pattern is a saved signature for a known problem, tagged , , or

(see chapter Pattern Editor).

> **Important** — a match can only make a line more severe, never less. An `ERR` line stays red even if no pattern matches it. A pattern match can raise an otherwise-quiet line (e.g. a RAID failure the kernel logged at `INFO`) up to critical, but it can never hide a high-severity line. If you see a strongly-coloured row with no pill, its colour came from the kernel's own level — read it anyway.

---

3. Pills, badges and decodes on the message

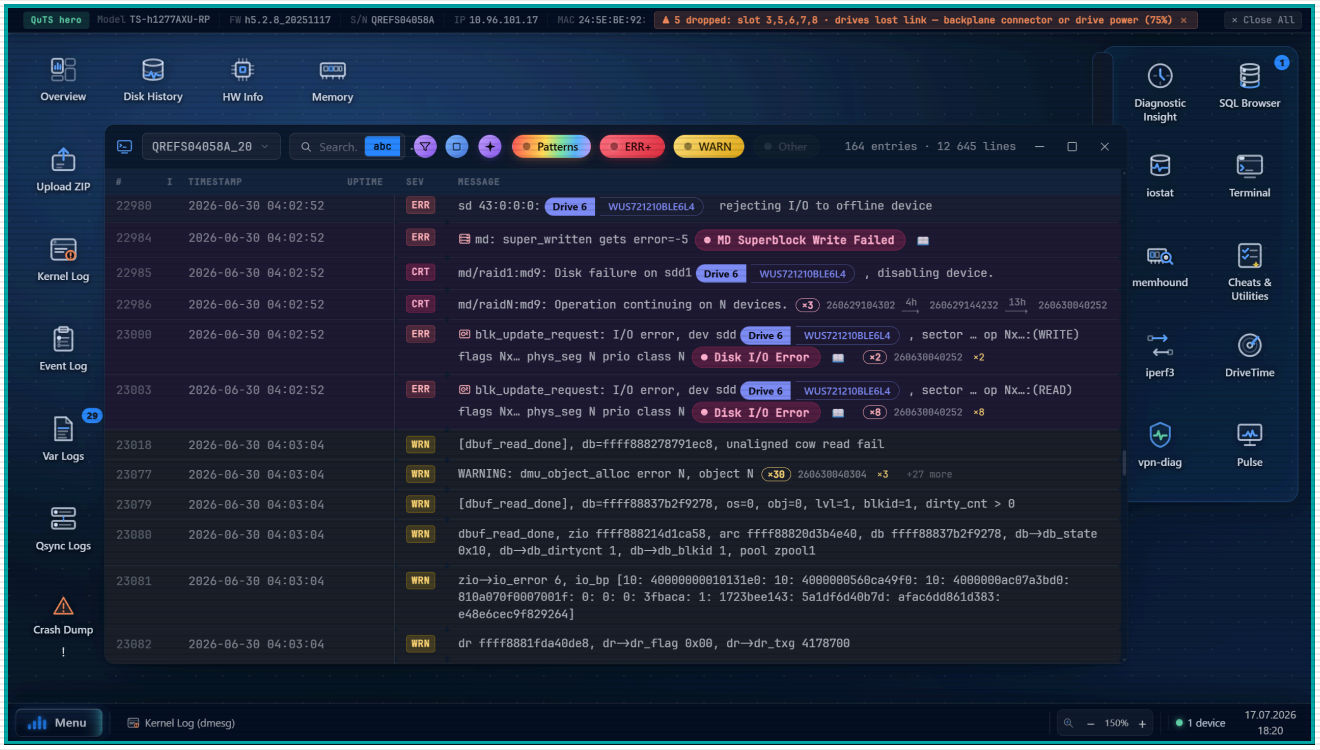
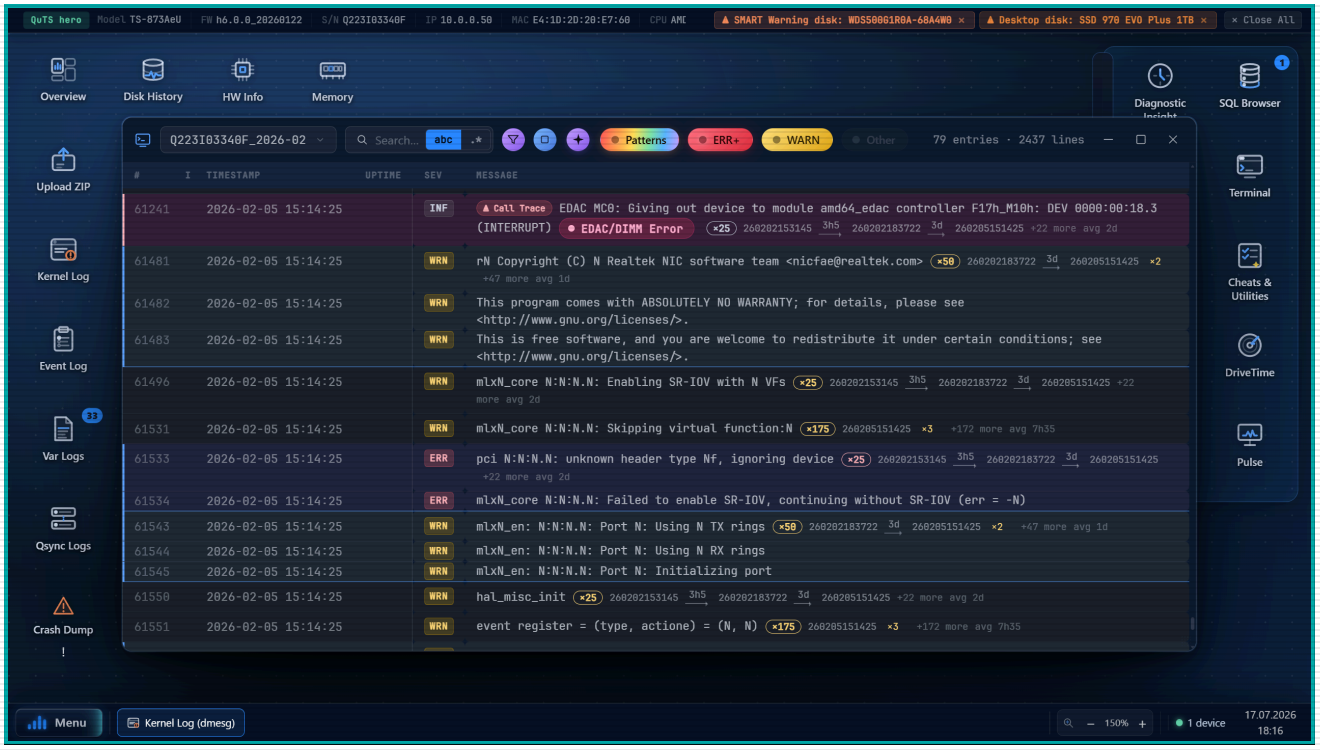


Figure — A RAID member being kicked out of md9, with inline drive-bay + serial badges and pattern-matched "Disk I/O Error" tags on the collapsed events.

- **Pattern pills** — a coloured pill naming each detection pattern the line matched. One line can carry several. The pill colour follows the pattern's category.
- **KB article** — the matched problem has a Knowledge-Base article (cause + workaround). Click it to open the write-up.
- **Decoded value** — for a few pattern types the viewer decodes the raw hex for you (e.g. an MCE machine-check bank status, or an NVMe `log_info` code) into plain language.
- **Drive-bay chip** — device names like `sdb` or `dm-3` are resolved to a physical drive-bay chip where the session has the enclosure map; click a drive chip to jump to its SMART page.



4. Ignore rules – hiding expected noise

Some lines are benign and always present (link-up chatter, `crng init done`, USB re-enumeration). An greys those out so they stop competing for your attention.

Ignore rules only – they never change a pattern match or a line you marked by hand:

- In the normal table an ignored line is (its severity colour is suppressed) but still shown.
- In , , the and the , ignored lines are .

> **Tip – create ignore rules from the log itself.** Right-click a benign line → **Add to ignore**. The viewer builds a sensible pattern for you (numbers, interface names, addresses generalised) so one rule covers all its variants. Manage and scope rules later in the **Pattern Editor**.

---

5. Cutting a big log down: Focus, Dedup, Search

A raw kernel log is 4–20 MB. Three controls make it tractable.

Focus

hides everything routine and keeps only the lines that stand out. A line survives Focus if it is , , or . Everything else – including ignored lines and routine `NOTICE` / `INFO` / `DEBUG` – is dropped. Use Focus first on any log

you have never seen: it turns thousands of lines into the handful worth reading.

> **Tip — mark before you Focus.** Right-click any line → **Mark line** to pin it. Marked lines always survive Focus even if they are otherwise routine, so you can keep context lines next to a fault.

Dedup

Dedup collapses repeated lines and repeated blocks into a single row with a count. Groups are ordered by their most recent occurrence, so the bottom of the list is what is happening now. Dedup is how you tell a one-off glitch (×1) from a storm (×5000) at a glance.

Search

The search box filters as you type. Toggle for pattern search (PCRE). Combine with the severity ceiling (e.g. "≤ WARN") to answer questions like "every error mentioning `ata`".

---

6. The minimap

The vertical strip on the right is a , top = start, bottom = end. Each band is coloured by the worst severity in that slice, so a red band near the bottom means a recent critical event. Click a band to jump straight there. Ignored lines never paint the minimap, so it shows real signal, not noise.

---

7. A practical reading routine

1. — it lands on the newest page.
2. — where are the red/orange bands? Recent (bottom) matters most.
3. — collapse the log to warnings-or-worse + known patterns.
4. — see which of those are one-offs vs. storms (×N).
5. — critical → error → warning. Open any 📖 for the known-issue write-up.
6. — **Add to ignore** the confirmed-benign noise; **Convert to pattern** anything worth catching automatically next time; **Mark** the lines that tell the story.
7. — open to see the same events correlated with QNAP events and a plain-language verdict (next chapter).

> **Best practice.** Read the newest boot first. Disks and HBAs move between NAS units, so the most recent boot is the authoritative picture of the current hardware; older boots can describe a different machine.

Next: 0007. Reading Dr. Trace (the incident timeline) — how StorageDiagLV correlates these kernel lines with QNAP events into a single verdict.

Watch it

> In-app demo: Kernel Log — 44 757 lines cut down with the minimap, Focus and Dedup until the machine-check is the thing left on screen.
> In-app demo: Close-up on the Focus and Dedup toggles, with the row count changing as each one goes on.

> In-app demo: Close-up on the minimap chips — dropping the routine band so the heat map shows only signal.

> In-app demo: Close-up on the machine-check banner and the link through to the verdict.

> In-app demo: A kernel log pasted straight into the window — parsed, severity-coded and pattern-matched with no bundle involved.

> In-app demo: A kernel Oops with its register dump, and the taint flags spelled out — F and O are normal on a QNAP.

READING DR. TRACE (THE INCIDENT TIMELINE)

Dr. Trace is the biography of the NAS. It merges the two logs you read separately — the kernel log (`klogd_dump.log`, tagged `KRN`) and the event log (`log_tool`, tagged `EVT`) — onto a single absolute time axis, then correlates them into scored incidents with a plain-language verdict and a "how to recover" path. When you don't know where to start on a case, start here.

> **How to open it:** Start menu → Dr. Trace. Or from here: @@OPEN:Lifeline|Open Dr. Trace@@



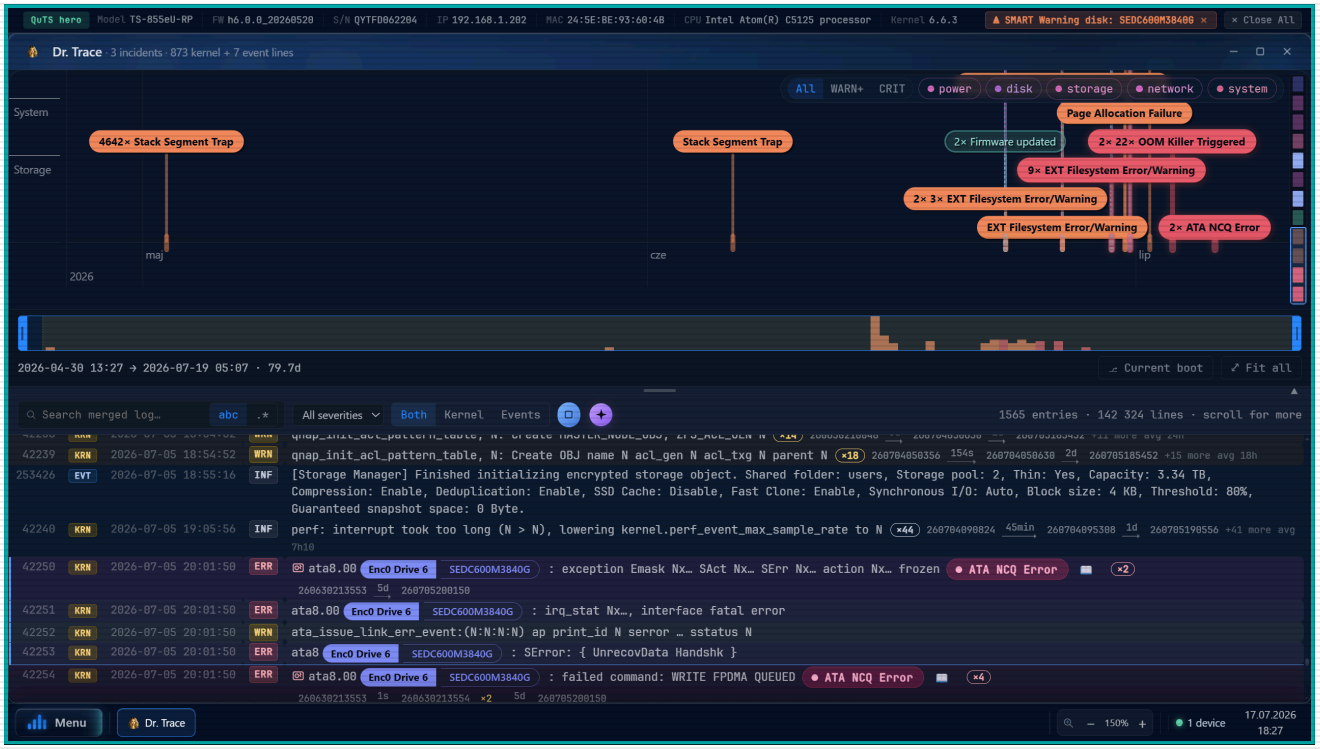


Figure — Dr. Trace correlates kernel call traces and event-log lines on a dual-lane timeline — here recurring ATA NCQ errors on Enc0 Drive 6.

The three areas

1. The incident timeline (top)

One block per — a group of correlated signals, scored into a ranked verdict and coloured by its likely cause. Incidents are grouped into (disk, memory, network, power…). Under each verdict lane you also get:

- — the raw evidence behind the verdict.
- — factual milestones: boots, NIC link up/down with the reason, per-drive SMART/disk faults, and KB-flagged kernel events (e.g. a ZFS panic).
- — a `fault → recovered` bar (e.g. `PSU_FAIL → PSU_OK`) when the fault later cleared.
- `×N` — many identical incidents collapsed into one, with the count.
- — catastrophic faults (CPU/MCE, DIMM, PCIe, backplane) are drawn in red/pink/purple.

2. The scrubber

A full-life trim bar under the timeline with a severity "waveform" of the whole history and a shaded band showing where kernel coverage actually exists. Because QNAP's kernel log concatenates many boots, the timeline usually reaches back . Use to jump to the latest boot, or fit the whole life. Moving the scrubber drives both the timeline and the merged log below.

3. The merged log (bottom)

The same lines as the Kernel + Event logs, interleaved in true chronological order and following whatever window the timeline shows ("respecting time"). Rows are tinted by the their syslog severity and any matched pattern (same rule as the Kernel Log); benign lines matched by lose their tint and grey out. Toggle to collapse repeats into `×N` groups across the whole history (kernel and event lines are never merged together; call traces stay intact).

The verdict popup (click any incident)

Clicking an incident opens its diagnosis and scrolls the merged log to its exact lines. The popup shows, top to bottom:

1. — the localized verdict.
2. — primary cause · "✓ recovered {date}" · `×N` · time period · a badge (= inside the current boot; = older history, where the snapshot's topology may not match).

3. — one coloured segment per candidate cause, sized by its probability.
4. (when relevant) — an MCE→RAM warning with a link; it always says verify before replacing anything.
5. — opens the Knowledge-Base article for the diagnosed rule (when one exists).
6. — the ranked causes, each with a ; the ⓘ opens root-cause help.
7. — the raw log lines behind the verdict.
8. — change the (7d / 30d / 1y / all) and the , then . Settings you've computed before return instantly from cache ("cached · computed … ago").

How to read it

1. and look at the loudest, most-recent blocks on the timeline.
2. → read the verdict split and the evidence lines. That's the machine's best correlated guess with its proof.
3. for the known-issue workaround, or the memory advisory's for suspected RAM.
4. around the incident (it's already scrolled there) to see the kernel and event sides of the same moment together.
5. — treat (older-than-this-boot) verdicts as leads to confirm, not facts, because the disk/topology map came from the current snapshot.
6. If a verdict looks too narrow or too wide, widen/narrow the and .

> **Note.** The merged stream is complete — it pages the full history, nothing is silently sampled away. The one exception is a very low-RAM NAS, which shows only the newest slice and says so.

Next: 0008. Reading S.M.A.R.T. disk health.

Watch it

> In-app demo: Dr. Trace — narrowing eleven incidents to the current boot and reading the CPU machine-check verdict. In-app demo: Close-up

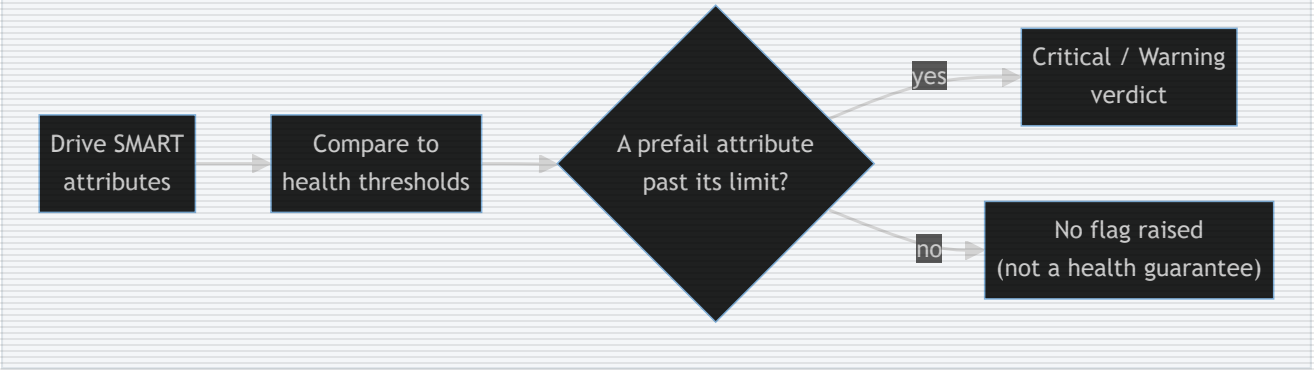
on the scrubber controls — pinning the view to the current boot, then collapsing repeats.

In-app demo: Twenty-one segmentation faults, an adapter that dropped and a fan failure, ranked on one axis.

READING S.M.A.R.T. DISK HEALTH

The [illegible] viewer shows one card per physical drive, parsed from that slot's `tmp/smart/smart_<enc>_<slot>.info` file. It turns the raw attribute dump into a per-disk verdict plus the context you need to judge whether a disk belongs in this NAS at all.

> **How to open it:** Start menu → **SMART**. You can also jump here by clicking a drive chip in the Kernel Log or Dr. Trace. Or from here: `@@OPEN:Smart|Open SMART@@`



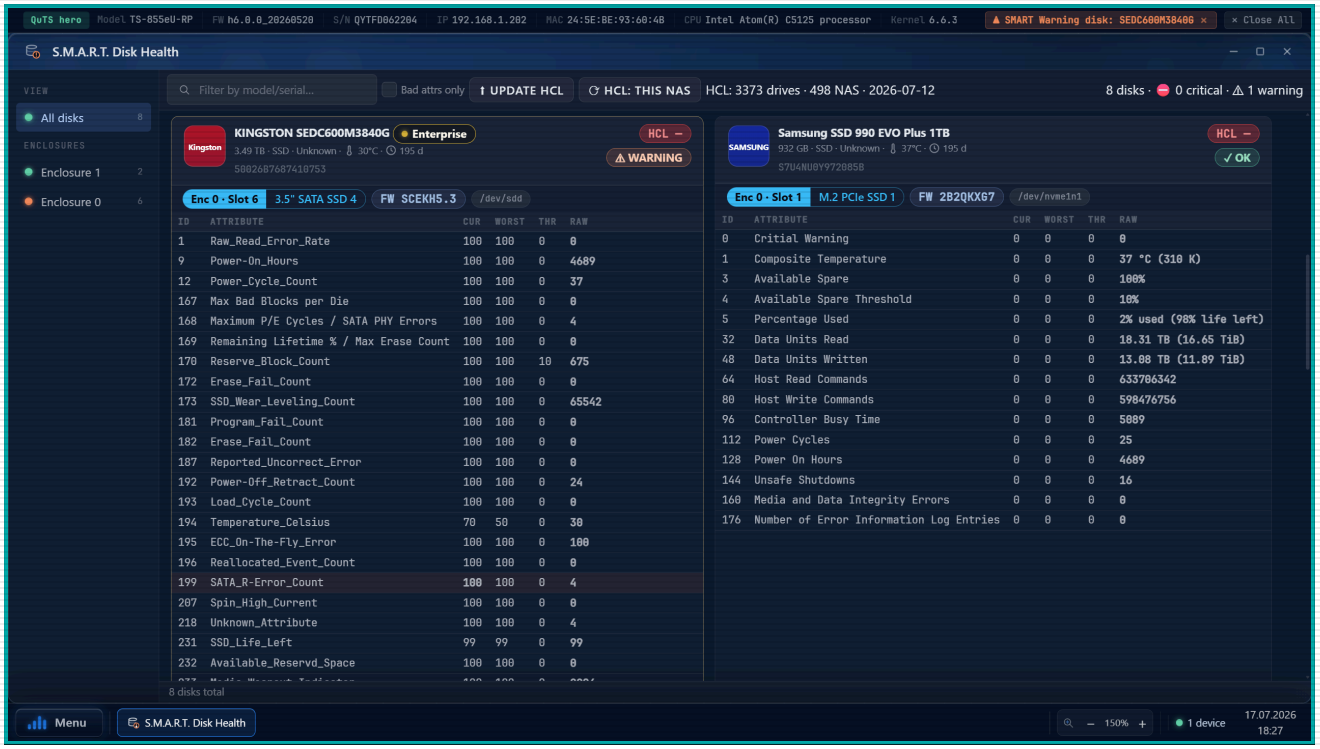


Figure — S.M.A.R.T. Disk Health: a SATA enterprise SSD flagged Warning next to a healthy NVMe SSD, with full attribute tables and HCL lookup.

Read a card top-down

1. The header verdict — OK / WARNING / CRITICAL

This is the result: the worst level found across all attributes. means at least one attribute crossed a hard limit the drive's own vendor threshold was breached (normalized fell below — the drive saying "I am pre-failing").

2. The attribute table

Columns: . Two things decide health, and you should read both:

- — Reallocated Sectors (5), Reported Uncorrect (187), Current Pending (197), Offline Uncorrectable (198), UDMA CRC Errors (199), Reallocation Events (196).
- — if , that's the drive vendor's own about-to-fail signal, independent of the counters above.

turns amber at 55 °C and red at 65 °C. Use to hide the healthy rows. Hover any row for a plain-language description, likely cause, impact, and a bar showing where Cur/Worst sit relative to the threshold.

> **Important — don't panic at giant Raw numbers.** A few attributes pack several values into one field and are shown **pre-decoded** for you: Command Timeout (188) is really three counters (`total / >5s / >7.5s`), Power-On Hours (9) and Head-Flying Hours (240) can carry milliseconds in the upper bytes. Read the decoded value, not the packed blob.

3. NVMe cards read differently

Same table, different meaning — the same ID number means different things on NVMe. The rows that matter: (endurance consumed / life left), (spare below threshold = critical), (shown in °C), and the (any bit set = fault). SATA-style reallocation IDs do not apply.

4. Context chips (not SMART health)

Around the verdict you'll see chips that are context, not the health result:

- — NAS / Enterprise / Desktop / Surveillance / Archive / ⛔ (from model patterns).
- — ✓ compatible with this NAS · ✗ known NAS but disk not listed · ? NAS model unknown · ~ inferred · – not found. (`?` / `–` mean "can't determine", which is the same as "incompatible".)

- , (with a ⚠ when the power-on counter looks reset/wrapped), and seal % for helium HDDs.

> A drive can be "OK" on SMART yet still flagged as a consumer/blacklisted model or off the compatibility list. Both matter for a QNAP support verdict.

The whole-device verdict banner

When one or more disks in the bundle are – a failing SMART state, or a drive class that doesn't suit the role (consumer, archive-SMR, or blacklisted in a RAID) – a red appears (e.g. "SMART Critical disks ×2: ST10000NM0086-2AA101…"). Click it for a that explains, in plain language, what it means, (raised by the SMART viewer after it classifies each disk), the , and – most useful – the (e.g. "ST10000NM0086-2AA101 (Enc 0 Slot 13), … (Enc 0 Slot 14)"). That's your jump-list to the disks that need attention.

Snapshot vs. trend – SMART here, DriveTime for history

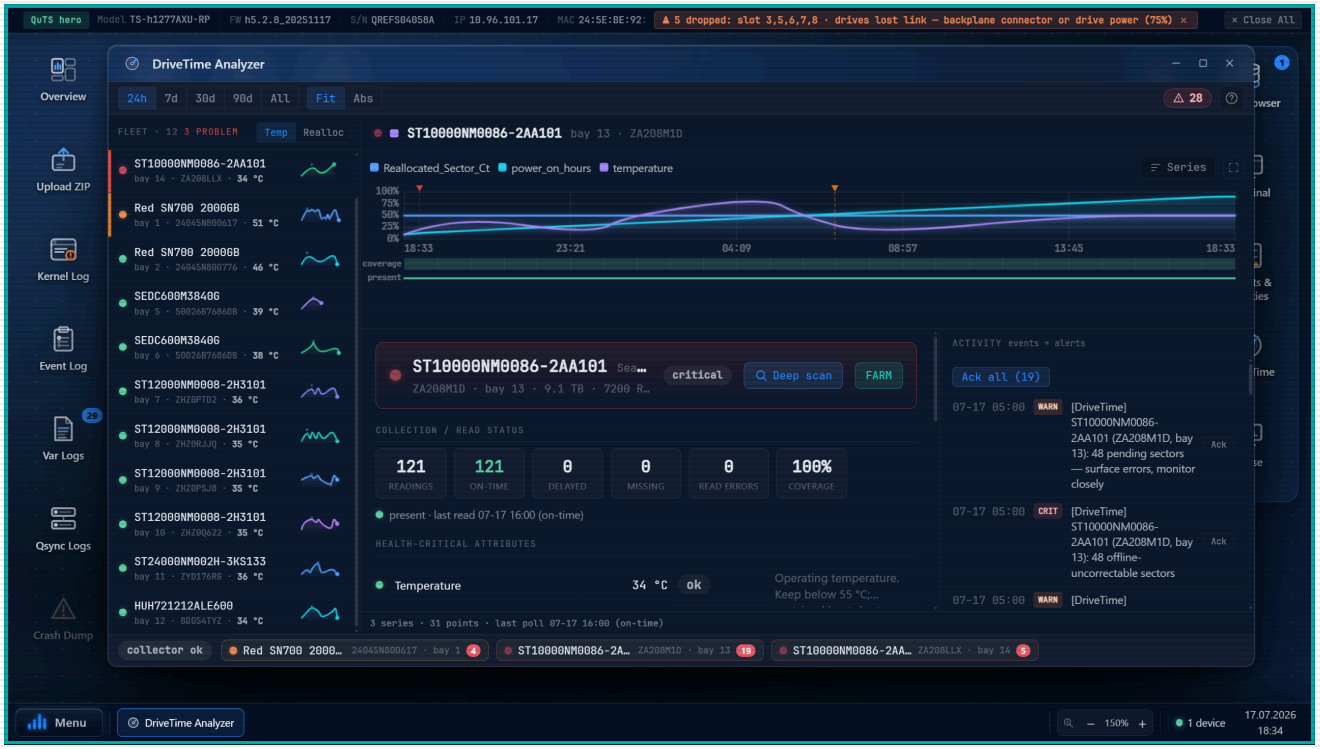


Figure – DriveTime plots each drive's SMART attributes over time – here a critical Seagate disk with pending and offline-uncorrectable sectors in the fleet alert feed.

This card is a from the uploaded bundle. To see whether a value is climbing (e.g. a reallocated-sector count creeping up week over week), use the fleet view – it runs the

but over the on-NAS collector's history, so a slowly-degrading disk shows up there before it trips a hard limit here.

> **FARM is not in the bundle.** An uploaded diagnostic carries only the SMART `*.info` snapshot. Deeper drive telemetry — FARM (per-head / reliability metrics) and vendor deep-scan data — can only be read from a live drive (the on-NAS collector or a deep-scan), never reconstructed from logs.

How to read it

1. Scan the across all cards — any CRITICAL/WARNING first.
2. On a flagged disk, open the table and check the and whether any . That's your "is it dying" answer.
3. Check and, on SSD/NVMe, .
4. Read the — is it even a supported NAS disk? A blacklisted or non-HCL disk explains a lot of "weird" behaviour on its own.
5. For "is it getting worse?", switch to for the trend.
6. Correlate a bad disk back to / the — reset/timeout/reallocation events there usually name the same slot.

Next: hardware, storage-topology and live-diagnostic chapters (to come).

Watch it

> In-app demo: S.M.A.R.T. — eight disks with no failing attribute, checked against the NAS compatibility list. In-app demo: Close-up on the compatibility-list controls and the coverage they report.

A WORKED EXAMPLE — DIAGNOSING A CASE END TO END

This chapter puts the reading skills together on a realistic support case. The numbers and messages are illustrative — the point is the method: which window to open, in what order, and what to look for. Once this routine is second nature, most cases take a few minutes.

> **Scenario.** A customer reports that a volume "keeps going read-only" and the NAS "freezes for a while, then recovers." They attached a diagnostic bundle to their QNAP support case.

Step 1 — Get the bundle into StorageDiagLV

Open the Salesforce case, click the extension icon, and let it push the attached bundle into StorageDiagLV. (No extension? Download the bundle and drag it onto the window instead — see 0003. Uploading logs.) The workspace opens with the session ready.

> **Best practice.** If the bundle contains more than one device dump (HA pair, expansion chain), check the taskbar session indicator and confirm you are looking at the NAS the customer is reporting on before you start. Use the **Session Manager** to switch.

Step 2 — Read the verdict first (Dr. Trace)

Open . Look at the timeline's most recent, loudest blocks and click the top incident.

In our scenario the verdict popup reads something like:

> **Likely cause:** Disk — Drive 5 (uncorrectable/pending sectors) **72%** · SATA link (bus resets) 24% · other 4% Badges: `approx` · `×14` · period "last 6 weeks"

Read it like this:

- The says the tool is fairly confident this is a , with SATA link resets as a secondary story (often the symptom of a dying disk, not a separate fault).
- `×14` means this pattern has recurred fourteen times — consistent with "keeps going read-only."
- `approx` means the incident is older than the current boot, so treat the bay mapping as a strong lead to confirm, not gospel.

- Follow if it's lit — that opens the Knowledge-Base article for this exact signature (cause + workaround).

Dr. Trace has already scrolled the merged log to the incident's own lines. Skim the evidence rows (they decided the verdict); the ones are context.

> **Why start here.** Dr. Trace has done the correlation for you — it lined up the kernel side and the QNAP-event side of the same moments and ranked the causes. Everything after this is confirmation.

Step 3 — Confirm in the Kernel Log

Open the to see the raw kernel evidence for bay 5.

1. Turn on to drop the routine noise.
2. Turn on — you want to know if the disk errors are a storm, not a one-off.
3. Search for the drive (e.g. `sdf` or the bay's device name; the drive-slot chip in Dr. Trace tells you which). You are looking for the classic dying-disk trio:
   - and messages,
   - clustered around them (the bus recovering after the disk stalls),
   - a filesystem or RAID reaction — , , or similar.

If those appear together, coloured red, and Dedup shows a high ×N, the kernel agrees with Dr. Trace: the disk is failing and the link resets are collateral.

> **Read the newest boot first.** Disks move between NAS units, so the most recent boot is the authoritative picture of the current hardware.

Step 4 — Judge the disk itself (SMART)

Open and find the bay-5 card. Confirm the physical evidence:

- Header verdict or .
- In the table, nonzero , , or — and/or a row where (the drive's own "pre-fail" flag).
- Note the and chips: if it's a Desktop or ⛔ Blacklisted model, or not on the compatibility list, that is part of the story — a non-NAS-rated disk failing this way is expected.

> **Snapshot vs. trend.** The SMART card is a single moment. If you want to show the count climbing over weeks (great for a warranty conversation), switch to the **DriveTime** trend view — same thresholds, but over time.

Step 5 — Cross-check the QNAP side (Event Log)

Open the [illegible] and set [illegible]. You should find the application-level echoes of the same story: "[Pool 1] … degraded", "Volume … changed to read-only", timestamps lining up with the kernel events. This is what the customer saw, in QNAP's own words — useful for writing the reply.

Step 6 — Write the diagnosis

You now have a defensible chain of evidence from three independent angles. A good write-up states the [illegible], the [illegible], and the [illegible]:

> **Diagnosis.** The disk in **Bay 5** is failing (uncorrectable + pending sectors, confirmed by SMART; CRITICAL). Its stalls trigger SATA link resets, which make the pool drop to read-only — matching the reported "freeze then recover." Recurred ~14× over six weeks.
>
> **Action.** Replace the Bay-5 disk with a compatibility-listed NAS/enterprise model; after rebuild, confirm the link resets stop. (Note if the failed disk was a non-HCL / desktop model.)
>
> **Evidence.** Dr. Trace incident (72% disk); Kernel Log medium-error + reset storm on the bay-5 device; SMART CRITICAL on that drive.

The routine, distilled

1. [illegible] (extension or upload) → confirm the right device dump.
2. [illegible] → read the top verdict; follow "How to recover".
3. [illegible] (Focus + Dedup) → confirm the raw evidence on the named entity.
4. [illegible] → confirm the hardware (and check disk-type / HCL).
5. [illegible] → the customer-facing echo, for the reply.
6. [illegible] conclusion + evidence + action.

> **When Dr. Trace has no strong verdict**, invert the process: scan the **Kernel Log** minimap for red bands, **Focus** the log, and read the worst clusters directly — then use SMART / Event Log to corroborate whatever you find.

MCE DECODER — READING MACHINE-CHECK EXCEPTIONS

A is the CPU reporting a hardware fault it detected itself — bad RAM, a failing CPU cache, or a problem on the bus between them. The processor records a compact status word for each fault; unreadable on its own. The turns that word into a plain-language verdict: how serious it is, and which part of the machine looks responsible.

> Two ways in. **Paste** kernel-log or mcelog text into the decoder and read it anywhere — no live NAS needed. Or, on a live NAS, press **Scan this NAS** to read the current evidence straight off the machine (this live scan needs the **sdlvd** daemon and an **operator** or **admin** role).

What an MCE tells you — and what it doesn't

An MCE is the hardware raising its hand to say "something went wrong down here". It tells you the kind of fault and roughly where. It does , by itself, prove which component is bad. That distinction is the whole discipline of reading these: the decode gives you a , not a receipt.

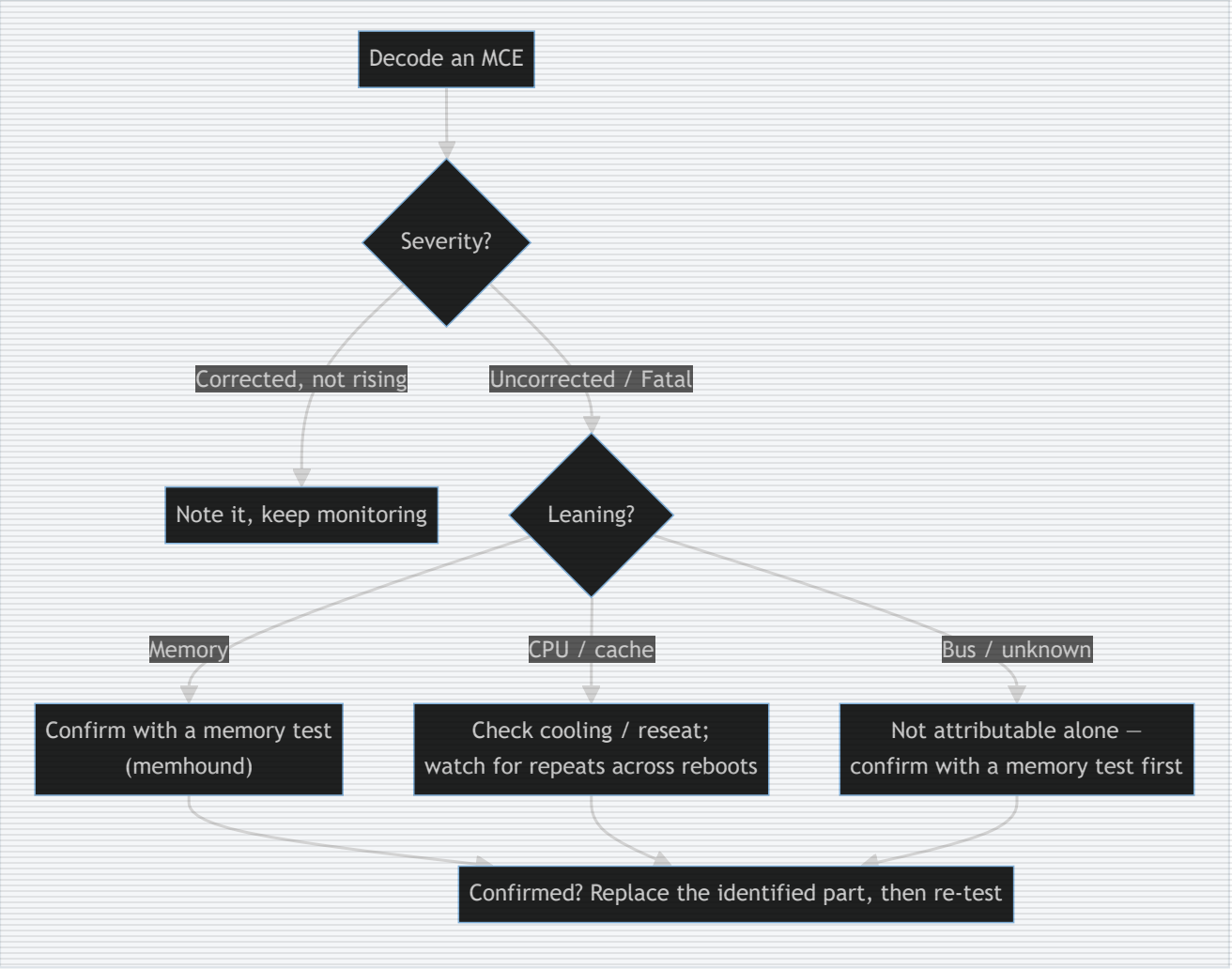
The three severities

Every decoded event is one of three levels — read this first.

Severity	Meaning	What to do
	The error happened but the hardware fixed it (usually ECC memory catching a flipped bit).	One is normal background noise. A rising count over days is the real signal.
	The hardware could not fix it. Data may have been affected.	Take it seriously; confirm the subsystem and act.
	The error corrupted the CPU's own working state; the system usually could not continue.	Something failed hard — this often is why a NAS rebooted or froze.

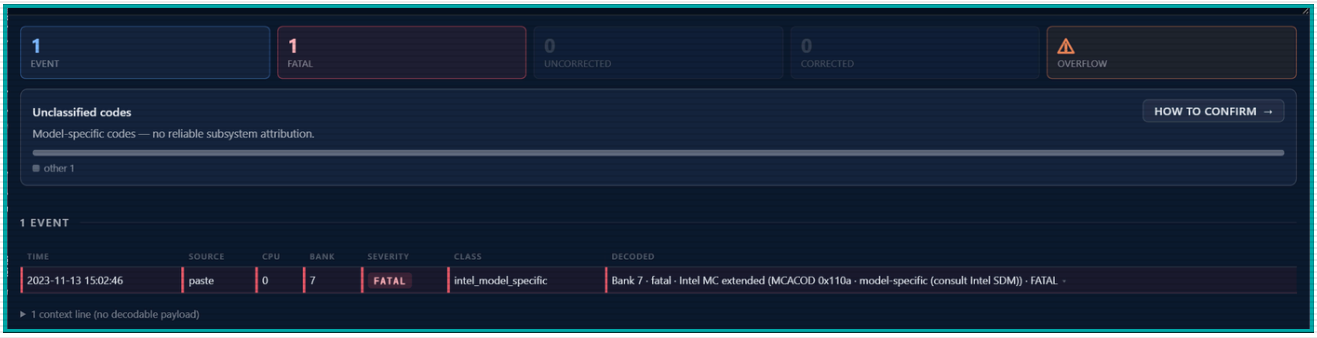
Decode is a suspicion — confirm before you replace

The decoder points at a subsystem: , , or the between them. Treat that as a lead to confirm, never as a shopping list — heat, a marginal power supply, or a loose DIMM can all imitate one another. The button opens the step-by-step procedure, and that procedure is the authoritative word.



What the window shows

- **[illegible]** – counts of fatal / uncorrected / corrected events, any firmware error records, and an **[illegible]** note when errors arrived faster than they could be logged (the true count is higher than shown).
- **[illegible]** – the one-sentence subsystem guess, with the **[illegible]** link.
- **[illegible]** – identical faults folded with a count, so a log repeating the same error a thousand times reads as one row. Below it, the full event list; open any row for the decoded detail.
- **[illegible]** – on any event found while the NAS was starting up. That fault happened *before* this boot, and very often it is the reason the machine restarted.



A decoded paste — the severity strip (here), the leaning line with its link, and the grouped event table. This example is a model-specific code, so the decoder honestly reports "no reliable subsystem attribution" rather than guessing a part.

Where the evidence comes from

QTS already runs the industry-standard collector in the background, so the evidence is usually waiting to be read:

- — MCEs the running system logged as they happened.
- the NAS keeps for exactly this purpose (empty on a healthy machine).

A — a box reporting zero machine checks is good news.

The background badge

On a live NAS you don't have to go looking. The dashboard quietly checks for machine-check activity in the background and, if any appears, raises a — amber for corrected-only, red when fatal or uncorrected events are present. Click it to open the decoder; that also marks the current evidence as seen, so the same set never nags you again. New evidence raises a fresh badge.

Typical workflows

- Scan it (or paste its kernel log). A event carrying the tag is the smoking gun for the restart. Cross-check the crash dump in 0013. Memory, OOM and crash dumps and the timeline in 0007. Reading Dr. Trace (the incident timeline).
- Paste the customer's kernel log. Repeated memory events that climb over time point at a DIMM starting to fail — confirm with a memory test in 0033. memhound — the RAM tester, then reseat or replace on the indicated channel.
- Paste just those — the decoder handles single lines, full mcelog blocks, and firmware error records equally.

Pitfalls

- ECC correcting one bit is the system working as designed. Volume and trend matter, not a lone hit.
- Some CPUs report a memory error without saying which channel; the decoder says so rather than guess.
- — RAM, CPU, or mainboard. Confirm with a memory test before attributing it.

- Ignore the boot-time "Uncorrected" and "Ethernet controller" lines. They mention MCE but report no error; the decoder filters them out on purpose.

> **See also:** 0008. Reading S.M.A.R.T. disk health (a machine check is about CPU/RAM/bus, **not** disks — SMART is the separate disk-health story) and 0017. Live diagnostics and Doctor utilities for the other live tools.

Demo clips

Short screen recordings of this window at work.

> In-app demo: Eight machine checks that only ever fire at boot — a firmware artifact, not a failing part.

> In-app demo: A machine-check line in the kernel log expanded in place into its decoded bank, status and address.

STORAGE LAYOUT — LVM, RAID & DRBD

On a NAS the storage stack is built from Linux software RAID + LVM (QuTS hero uses ZFS instead — see 0012. ZFS pools). The window draws that whole stack in one place, bottom to top.

> **How to open it:** Start menu / desktop → LVM. Or from here: @@OPEN:Lvm|Open LVM@@

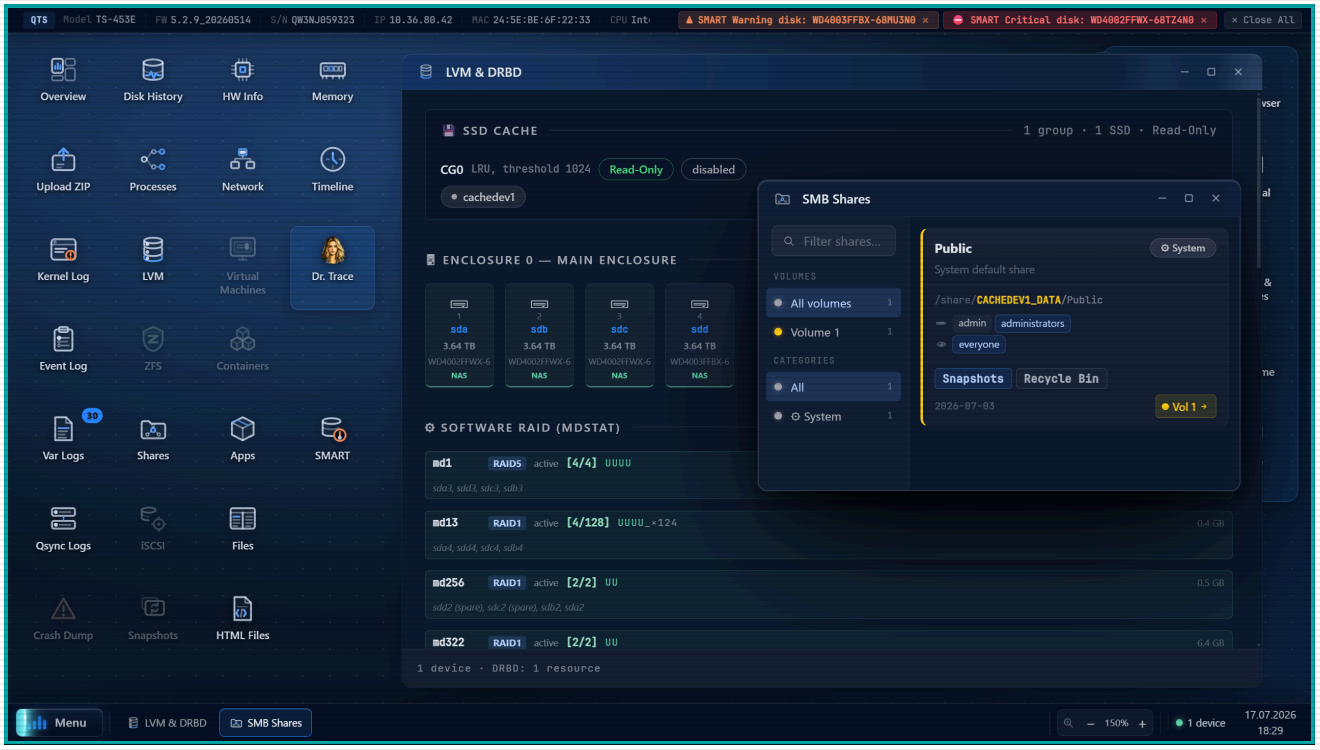
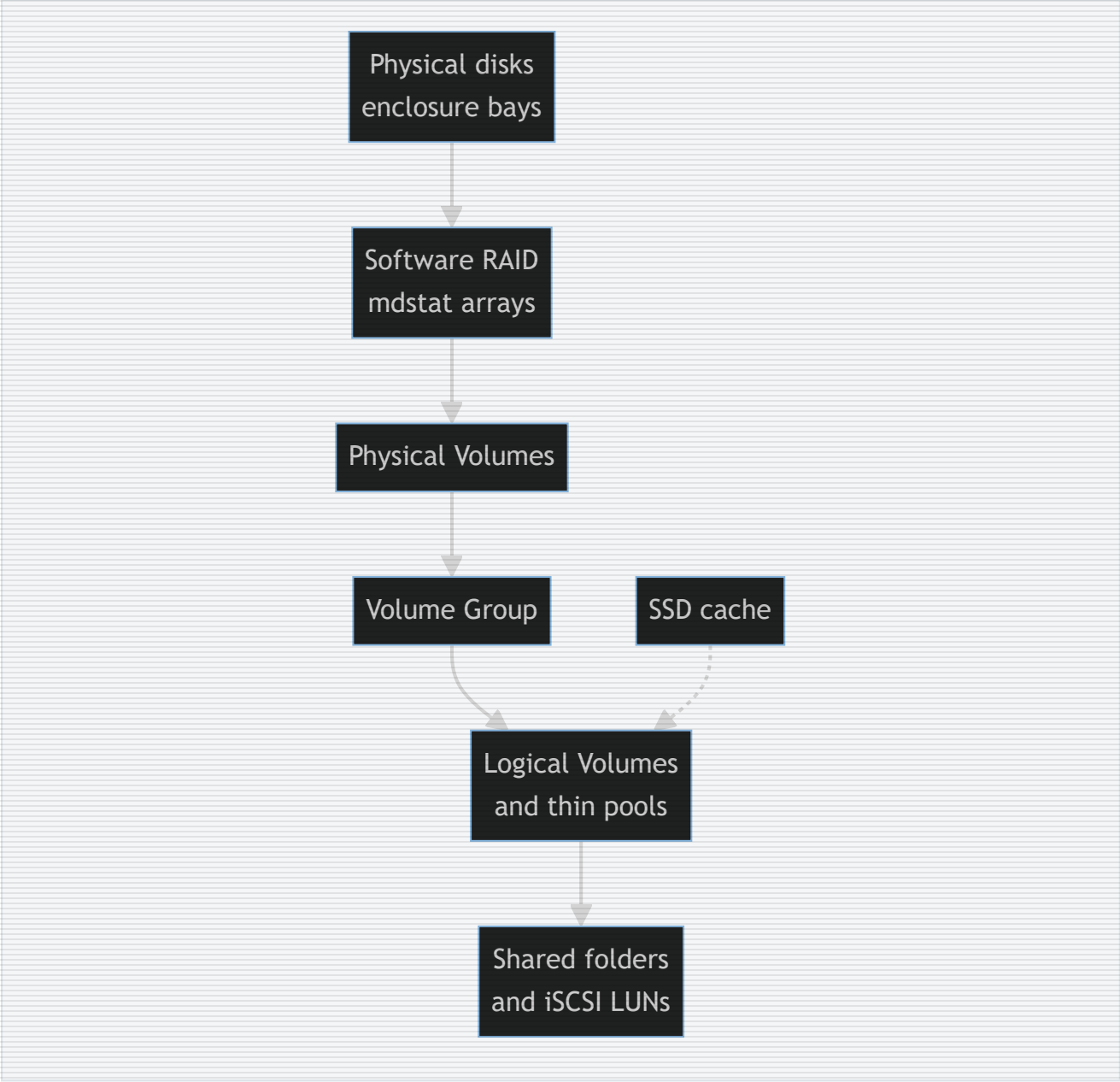


Figure — The LVM & DRBD viewer: SSD cache banner, enclosure bays, and the software-RAID (mdstat) arrays.



What you see, bottom to top

- One row per array (`md0` , `md1` , …) with its RAID level, how many member drives are , a row of per-slot boxes, its size, and a while an array is resyncing.
  - Slot boxes: , , . A healthy array shows all-up; a single red box means a degraded array.
  - QNAP's system arrays that mirror across every bay (`md9` , `md13`) are recognised, so a 4-bay NAS doesn't paint 124 empty slots — they collapse to a compact `_×N` chip and count as healthy.
- Each VG shows its Physical Volumes (with a or tag when a PV sits on top of one) and its Logical Volumes as coloured by type — thin-pool (orange), thick (green),

thin (blue), plus cache/linear/striped variants. Pools sort first, then by size.

- (HA pairs). Each resource card shows the two hosts with their device, disk and address, and the configured sync rate.

SSD cache

If the NAS has an , a banner at the top shows it: the (Read-Only or), the eviction algorithm (e.g. LRU), whether it's , and per-cache-device stats – , , and .

> **Watch the dirty % on a Write-Back cache.** "Dirty" is write data that lives only on the SSD and hasn't been flushed to the array yet. A high dirty % on a **Write-Back** cache means a lot of not-yet-persisted data is riding on that SSD – if the cache SSD fails before it flushes, that data is lost. (Read-Only caches don't carry this risk.)

Each drive bay also carries its – erprise / / Desktop… – the same disk-type classification shown in SMART, so you can spot a consumer disk sitting in a RAID at a glance.

How health is shown

- is the headline signal: any missing member turns that slot red and marks the array degraded; a rebuild shows a live progress bar.
- The separate table (per-drive superblock: online/offline, events, last-update) is read from the HTML report, so its numbers can differ slightly from the live `/proc/mdstat` array rows – treat it as the "what each disk's superblock thinks" cross-check.

Reader gotchas (worth knowing)

- It reflects the last-committed metadata QNAP saved; a very recent, uncommitted change may not appear. Sizes are derived from physical extents.
- – an LV that is part thick, part thin may show a single type. Read the bars as a layout guide, not a strict audit.
- The card labels the first host as "primary" purely by position; the configuration files do not carry the real Primary/Secondary state or connection status. A Secondary/Secondary or disconnected pair still shows host 1 as "primary". The sync , not a measured live speed.

- Thin pool fill % is not shown. Thin pools appear by type and allocated size only; the data/metadata usage percentages are not in the LVM backup, so this window can't tell you a thin pool is nearly full. (The percentages you do see belong to the separate banner.)

> **Best practice.** Use this window to confirm the shape of the array (level, members, degraded/rebuild) and where each volume sits. To judge whether a specific disk is failing, cross-check it in SMART (0008. Reading S.M.A.R.T. disk health) and the Kernel Log.

Watch it

> In-app demo: LVM and DRBD — a write-back SSD cache running at 99.89 % with dirty blocks still to flush.

ZFS POOLS (QUTS HERO)

On a QUTS hero NAS the storage is ZFS. The ZFS Pools window draws each pool's health, physical layout, cache/log devices, and datasets. (A QTS/LVM bundle shows a short note pointing you to the 0011. Storage layout – LVM, RAID and DRBD viewer instead.)

> **How to open it:** Start menu / desktop → ZFS. Or from here: @@OPEN:Zfs|Open ZFS Pools@@

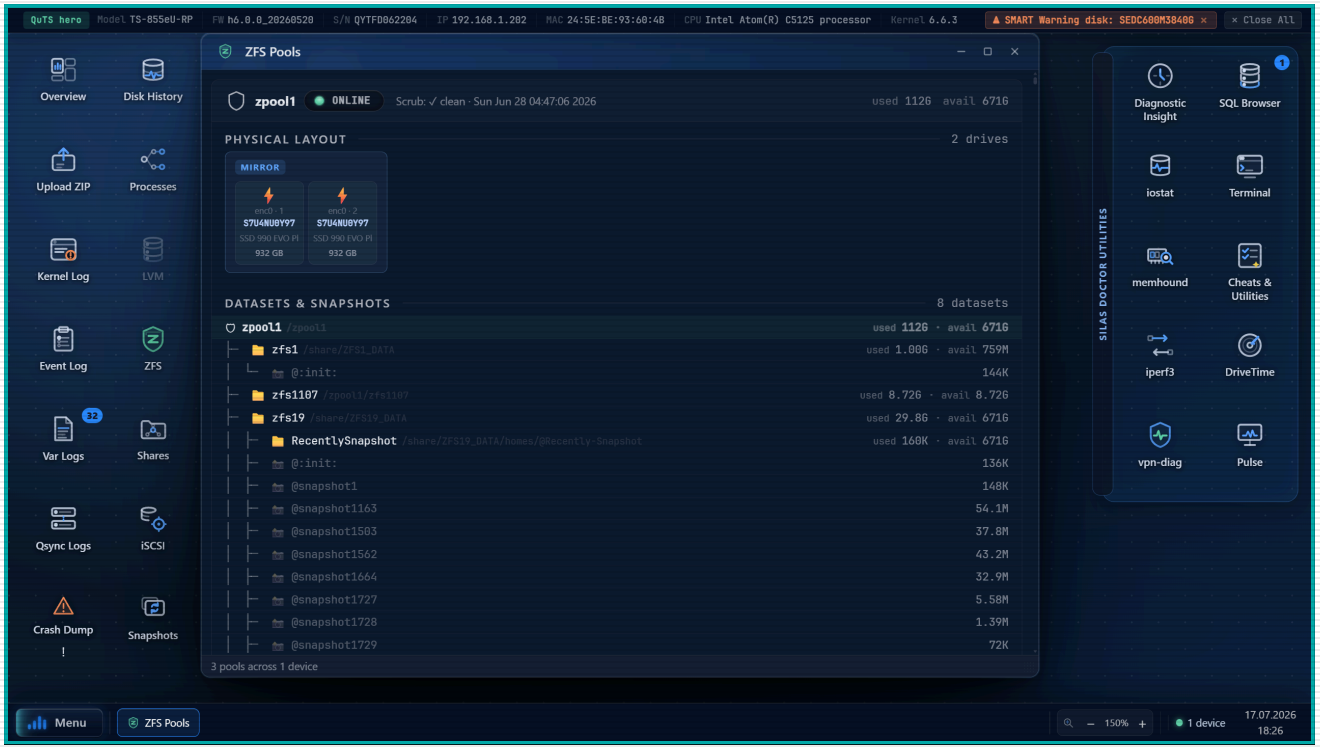
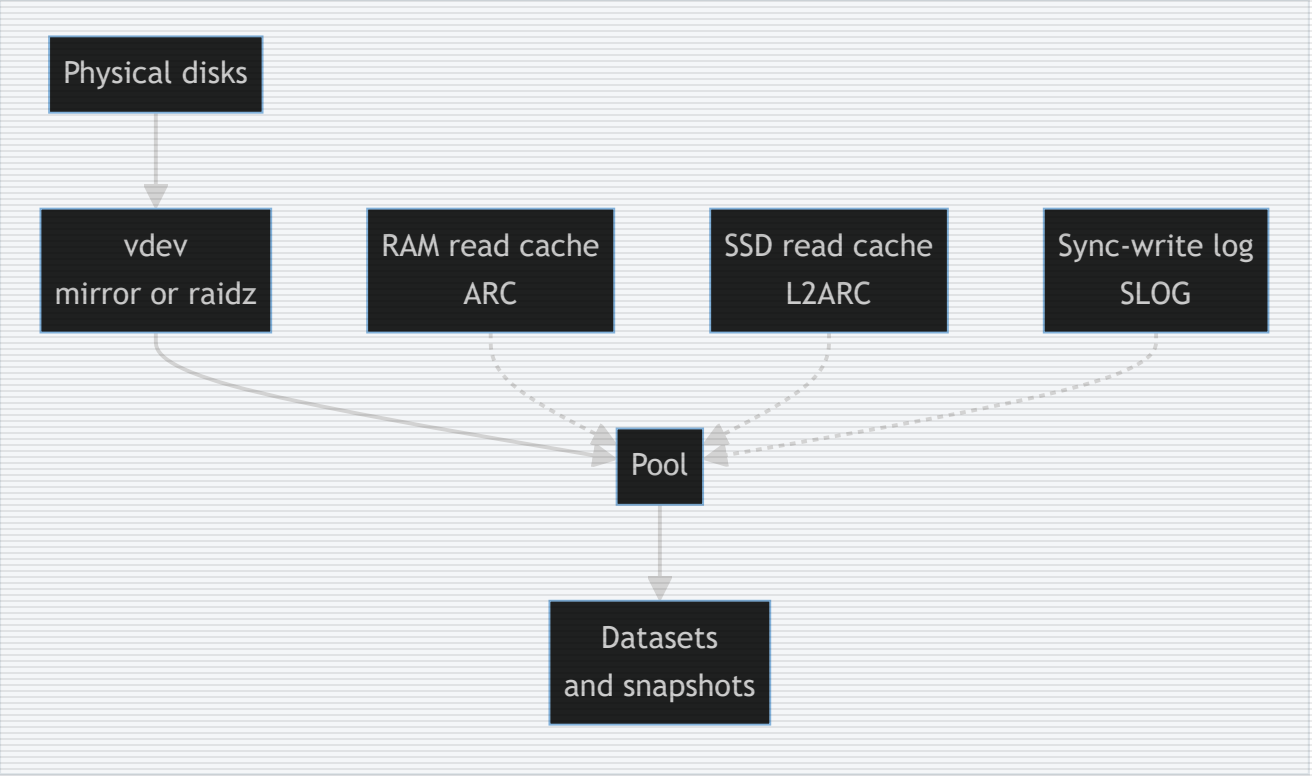


Figure – The ZFS Pools window: an online mirror pool with its physical drive layout and dataset/snapshot tree.



What you see

- — a green / amber / red LED for (FAULTED, UNAVAIL…), with the literal state text.
- — appears only when the pool reports something other than "No known data errors".
- — reformatted from the pool's `scan:` line into a plain result: `clean`, `N errors`, or `no scrub run`, with the date.
- — the vdevs as colour-coded cards (raidz, mirror, cache, log, spare), each disk showing its error counters. Disks are matched back to their bay, model and serial, and are .
- — L2ARC (read cache) and SLOG (write-log) devices are surfaced separately, with warnings that matter:
  - a → `single point of failure for sync writes`;
  - a → `critical, sync writes may fail`;
  - an → `warning, no data loss, but read cache lost`.
- — an indented tree with used/available per node and a 📸 marker on snapshots.

Where the data comes from

Everything in this window is read – specifically the `zpool status` and `zfs list` text QNAP already froze into

the HTML report. Nothing is run on the NAS.

Reader gotchas

- **live** A connected NAS can only list pool names (to recognise a QuTS hero box) and stream live throughput into the window. There is **zpool status** — live pool health, scrub state and error counters are available from an uploaded bundle. This is a real limitation, not a missing figure.
- may leave the RAID-type label and per-disk dots blank; the pool name, health, capacity and scrub text still show. For the full picture always use this window, which renders everything correctly.

> **Best practice.** Read the pool lamp and scrub result first, then the per-disk READ/WRITE/CKSUM counters — a disk with nonzero CKSUM is corrupting data and is your prime suspect. Click it through to SMART to confirm the hardware.

Watch it

> In-app demo: The ZFS viewer on a QTS bundle: no pools, and a pointer to the LVM viewer instead.

MEMORY, OOM & CRASH DUMPS

The Memory window answers three questions about a NAS's RAM, and the separate Crash Dump window is the black-box recorder for an actual kernel panic.

> **How to open them:** Start menu / desktop → **Memory** and **Crash Dump**.

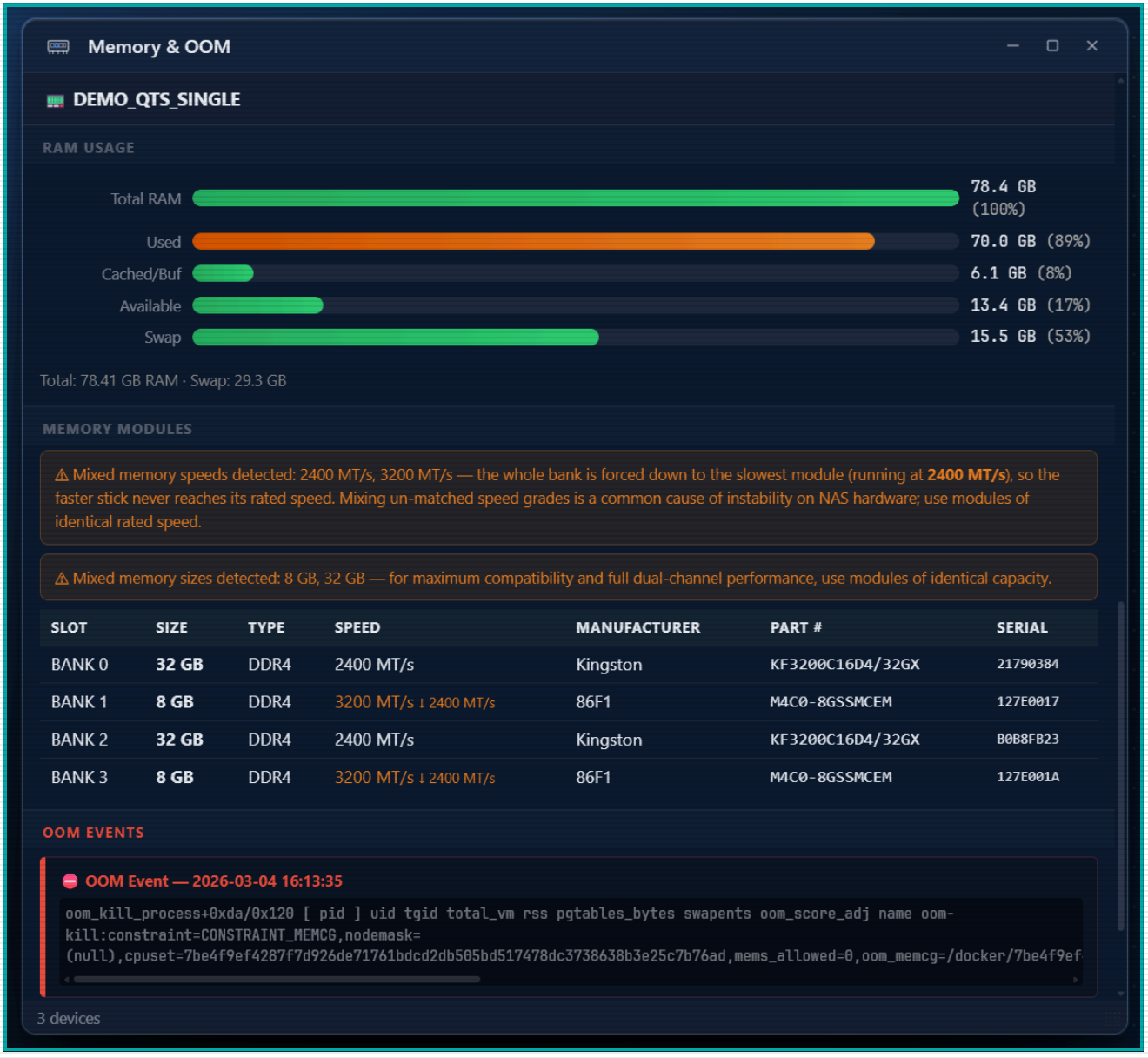
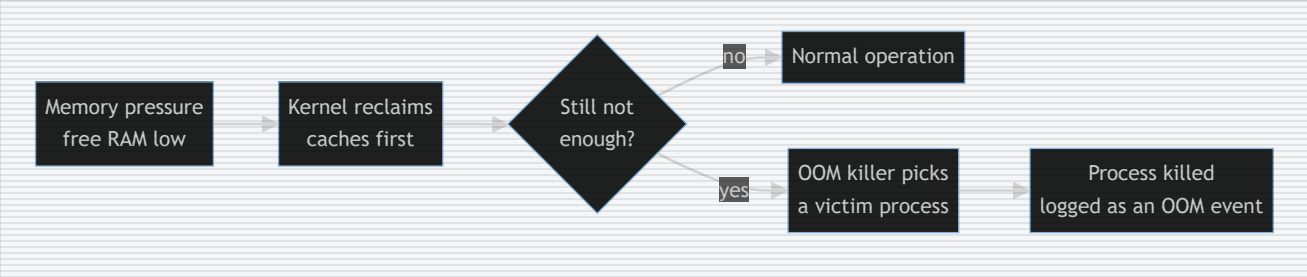


Figure – Memory & OOM: RAM/Swap bars, the DIMM table with mixed-speed/size warnings, and the OOM events list (here a Docker-cgroup OOM).



1. Is it running out of RAM?

The Memory window reads the same `/proc/meminfo` numbers Linux uses and draws bars for , and . The bar turns and , so a memory squeeze is visible at a glance.

> **Note.** There is no separate "high memory" alert beyond the bar colour — a red Used or Swap bar is the signal. The real problem indicators are the OOM list and the DIMM checks below.

2. Did the system kill programs to survive? (OOM)

When Linux runs out of memory it force-kills processes — an ("Out Of Memory") event. The tool scans the kernel log for these (`Out of memory`, `Killed process`, the OOM killer) and lists each one with its date and captured lines.

- Any event turns the section .
- OOM events are one of the clearest signs the NAS was over-committed, or that a program leaked memory.

> **Container OOM vs host OOM.** An OOM line that mentions `CONSTRAINT_MEMCG` or `oom_memcg=/docker/…` means a **container hit its own memory limit** (a cgroup) — not the whole NAS running out of RAM. That points at one container to fix (see 0020. Container Station), not the host itself.

3. Is the physical RAM installed correctly?

The window lists the per slot — size, speed, type, manufacturer, part # and serial — and actively compares the sticks, because mismatched memory is a classic cause of random NAS crashes:

- Mixing speed grades drags the whole bank down to the slowest stick and is a common instability cause; a stick forced below its rated speed is marked.
- is flagged.
- get a softer, informational "not validated as a kit" note — a prime suspect if MCE errors or random instability

appear elsewhere.

Crash Dump (pstore) – why the box rebooted

If the NAS suffered a , the crash log survives the reboot in a special memory area (`console-ramoops-0` , the "pstore" dump). The window shows those saved kernel messages line-by-line with their uptime timestamp, and highlights any lines matching known-problem – this is where you look to see why the box rebooted itself.

> **Best practice.** Put it together: the RAM bars show current pressure, the OOM list shows whether pressure already caused damage, the DIMM mismatch warnings show whether the hardware itself is a suspect, and the crash dump is the recording of an actual panic. A mixed-DIMM warning next to MCE events in the Kernel Log / Dr. Trace is a strong RAM lead – confirm with **memhound** or **memtest86** (see 0099. About, privacy and licence on memhound's confidence).

Watch it

> In-app demo: Memory and OOM – RAM at 91 % with the installed modules listed underneath.

> In-app demo: The kernel crash dump from pstore, flagged as carrying both a kernel warning and a machine-check.

NETWORK INTERFACES

The Network Interfaces window shows each NAS port's current status and addressing. The history of link drops (and the tool's guess at why) lives in [illegible] – the two views complement each other.

> **How to open it:** Start menu / desktop → **Network**. Or from here: @@OPEN:Network|Open Network Interfaces@@

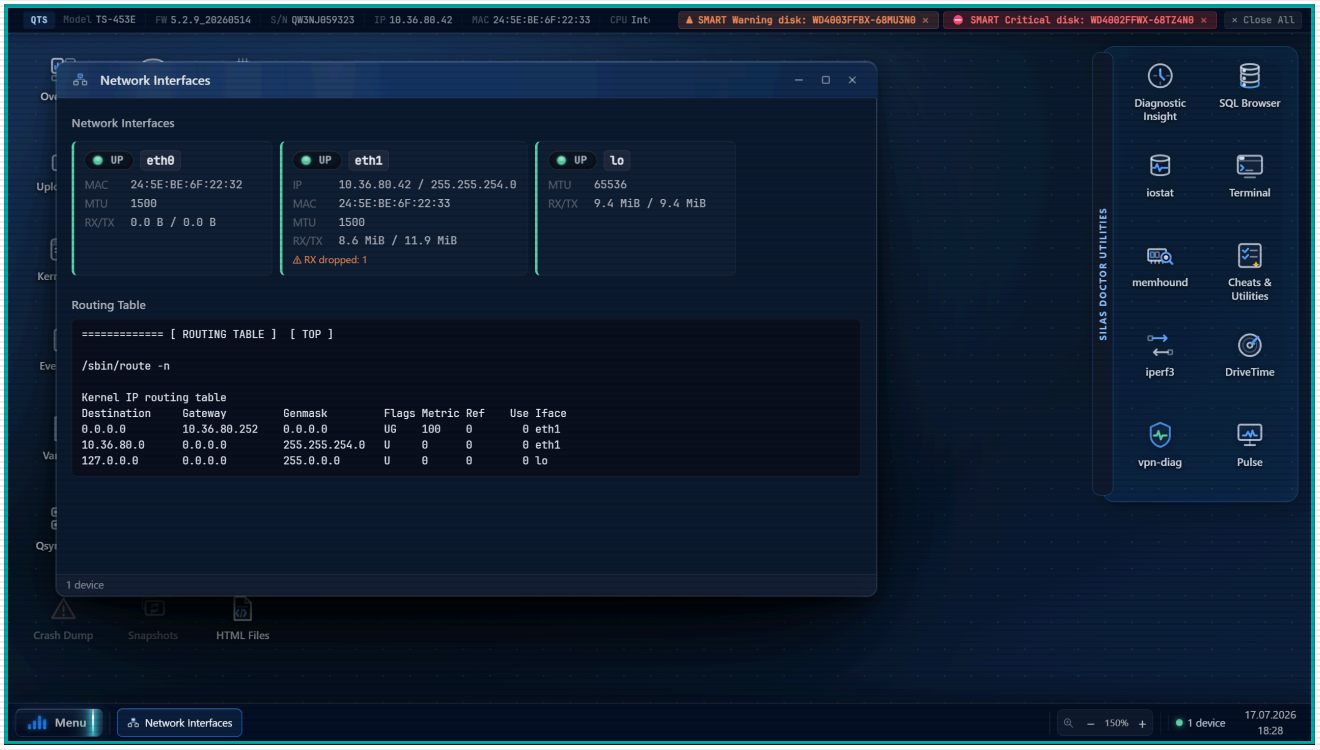


Figure – The Network Interfaces window: per-NIC status cards (note the RX-dropped warning on eth1) plus the parsed routing table.

What the Network window shows

Per NAS in the bundle, one card per network port:

- a green / grey link lamp,
- [illegible],
- total traffic,
- a line if the port has been dropping received or transmitted packets.

Below the cards it prints the device's and its list.

All of this is read (the `ifconfig` and routing sections embedded in the HTML report, plus `nic_model.conf`), so it reflects the moment the diagnostic was captured – not a live query.

> **No bond / LACP topology.** There is no special handling for bonded or LACP links — a bond appears simply as another port card if the report lists it. LACP problems show up instead as log patterns (e.g. "LACP No Partner Response") in the Kernel Log.

Link changes over time & the "why" — in Dr. Trace

Because a network driver rarely states why a link dropped, Dr. Trace infers it from two clues drawn from the kernel log:

- → suspected cable / connector / switch-port problem.
- → an unstable cable or SFP/transceiver — reseat or replace it.

Separately, QNAP's own event log turns `[Network] Interface … disconnected/connected` lines into "Network: up/down" entries on the timeline.

> **Best practice.** Use the Network window for current per-port status and addressing; use **Dr. Trace** (0007. Reading Dr. Trace (the incident timeline)) for the history of link drops and the diagnostic guess at their cause. A "renegotiated below best speed" note plus RX-dropped counters is a classic bad-cable signature.

Watch it

> In-app demo: Network Interfaces — every port with its address, MAC, MTU and traffic counters.

HARDWARE INFO — SENSORS, DMI, USB & INSTALLED APPS

The HW Info window pulls everything about the NAS's physical body out of the bundle and lays it out in one place (a merged window with a left rail: System / Hardware / Sensors / USB). Installed Apps is a separate window listing the QPKGs on the NAS.

> **How to open them:** Start menu / desktop → **HW Info** and **Apps**.

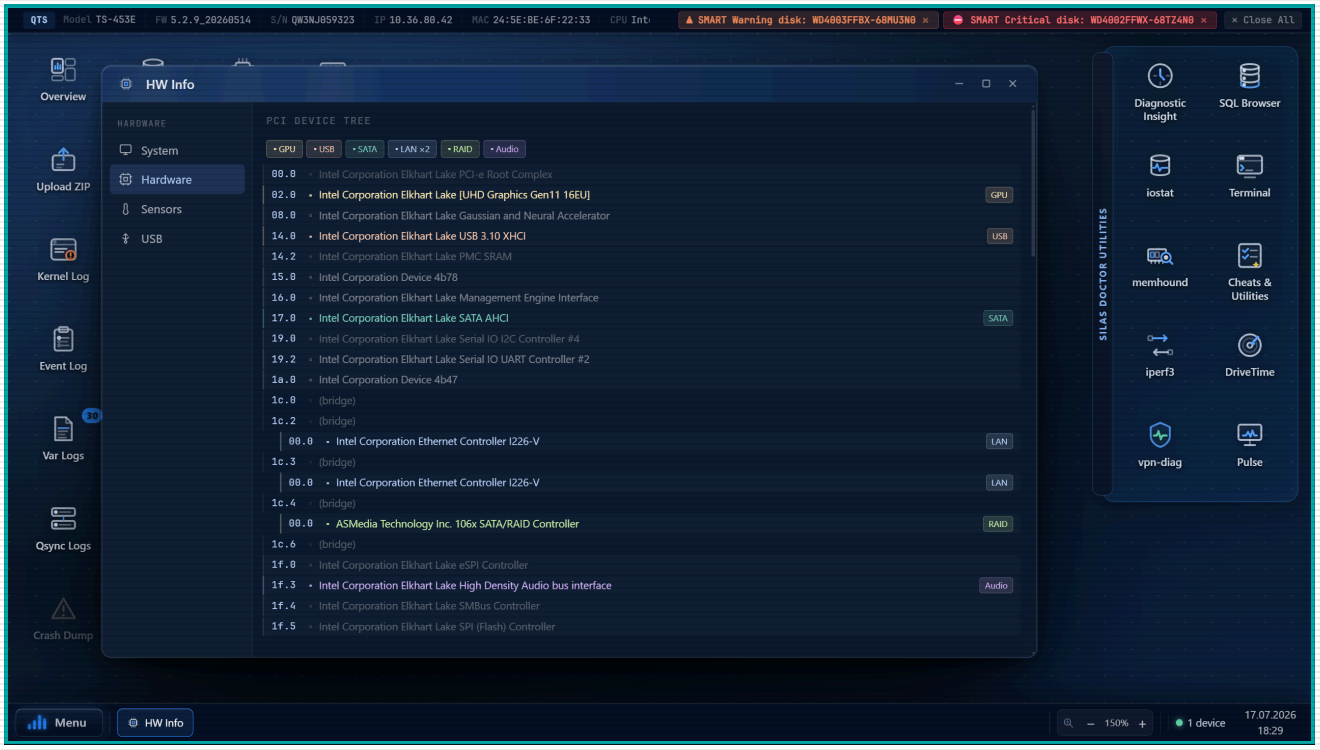


Figure — The HW Info window: the PCI device tree with GPU/USB/SATA/LAN/RAID tags, alongside the Silas Doctor Utilities launcher.

Sensors — fans, temperatures, PSUs

Read from the enclosure's sensor files:

- Fans — RPM, green when spinning.
- Temperatures — CPU, System, and generic probes, coloured by how close they are to their limit.
- PSUs — OK, or a blinking red Fail, with wattage and PSU temperature when the hardware reports them.

> **Important.** These are **enclosure / board** temperatures, captured at the moment of the dump (they do not refresh). Per-**disk** temperatures live in **SMART**, not here.

System & Hardware

- from DMI — manufacturer, model, serial, etc. (QNAP's empty "Not Specified" / "To Be Filled By O.E.M." placeholders are hidden).
- — colour-coded so you can spot NVMe, network, SATA/RAID controllers, GPUs and USB controllers at a glance.
- , the (which chassis is which), and each disk's .

USB

Every attached USB device — from `lsusb`, or reconstructed from kernel messages as a fallback — with bus/device numbers, vendor and product names, IDs, USB version and serial, plus a table of USB-related so you can see disconnects or errors.

Installed Apps

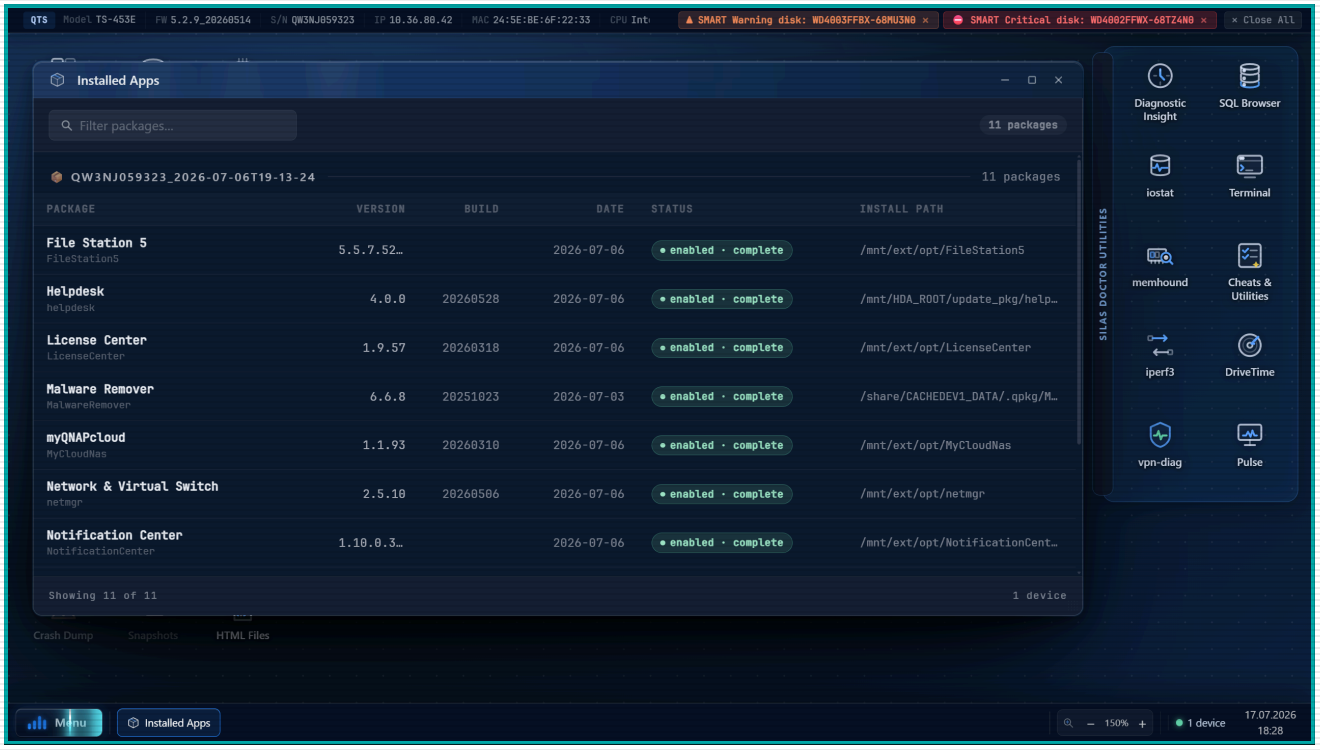


Figure — Installed Apps: every QPKG with version, build, install date, enabled/status and install path.

Reads `qpkg.conf` and lists every QPKG installed on the NAS: , with a filter box (searches name, id, version, or author).

> **Best practice.** HW Info answers "what hardware is in this box and is it physically healthy (fans, temps, PSU)?" A failed PSU or a fan reading 0 RPM here, next to thermal events in the Kernel Log, is a direct

> hardware lead. Installed Apps helps rule an add-on package in or out as the cause of a problem.

Watch it

> In-app demo: HW Info — enclosures, RAID configuration and per-drive serials read out of the bundle.

> In-app demo: Installed Apps — 24 packages with version, build date and install path.

■ PATTERN EDITOR & REGEX HELPER

are the heart of StorageDiagLV: saved signatures for known problems that colour a log line, add a named pill, and link a 📖 Knowledge-Base article. The is where you create and curate them; the teaches you to write the regular expressions they use.

> **How to open it:** Start menu → **Patterns**. (Editing is available to operators/admins.) Or from here: @@OPEN:Patterns|Open the Pattern Editor@@

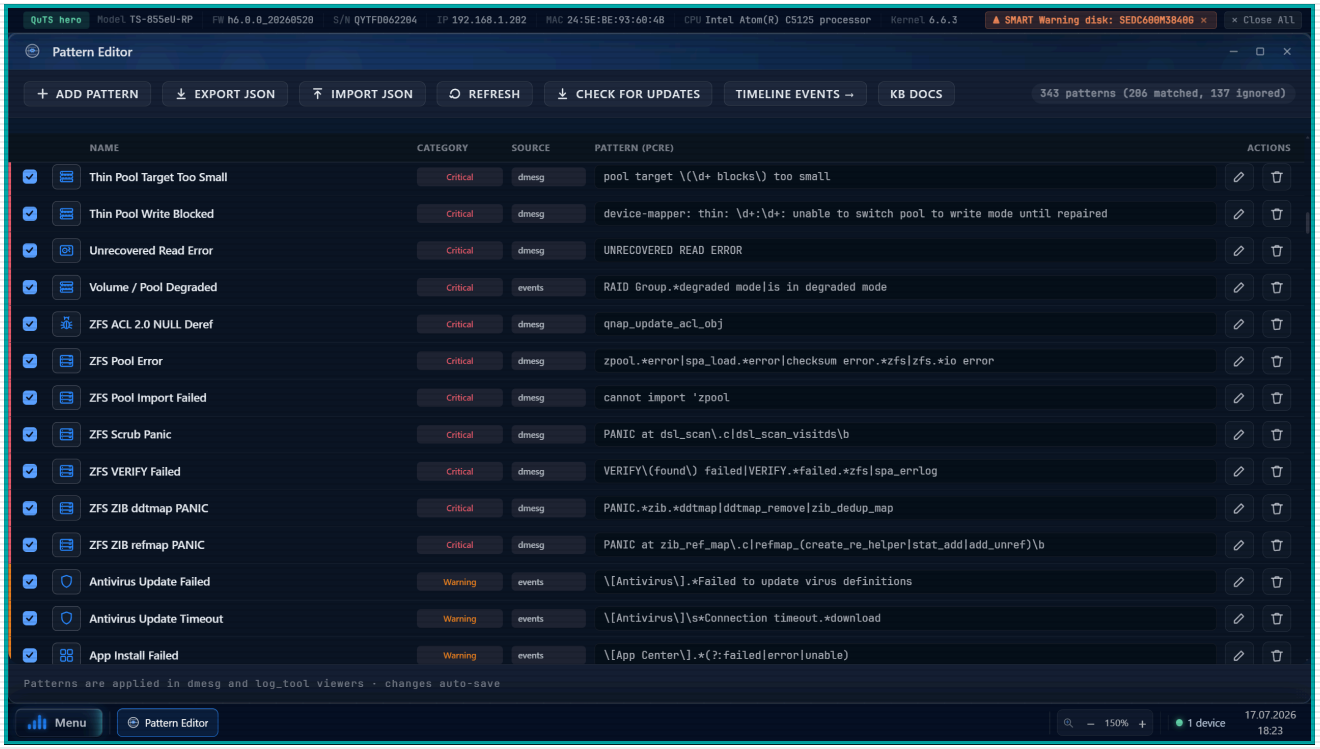
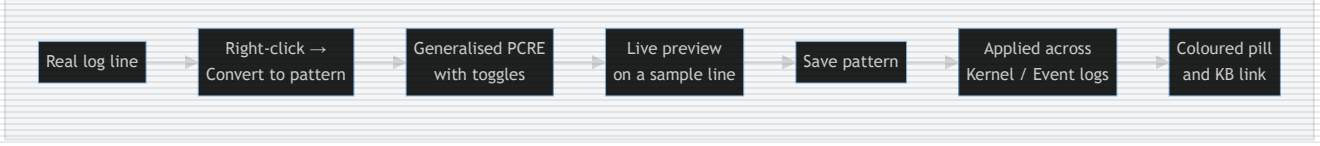


Figure — The Pattern Editor: every detection signature with its category, source and raw PCRE, plus add / import / export / Check-for-updates / KB Docs.



What a pattern is

Each pattern has:

- — human label shown on the matching line's pill.
- — — how severe a match is (this feeds the row colour in the Kernel Log; see 0006. Reading the Kernel Log (dmesg)).

- — where it applies: (kernel log), (QNAP event log), or .
- — the signature itself (no delimiters; matched case-insensitively).
- Optional , , and a (the 📖 link).

Adding or editing a pattern

Click (or the ✎ on a row). Fill in name, category, source and the regex, then .

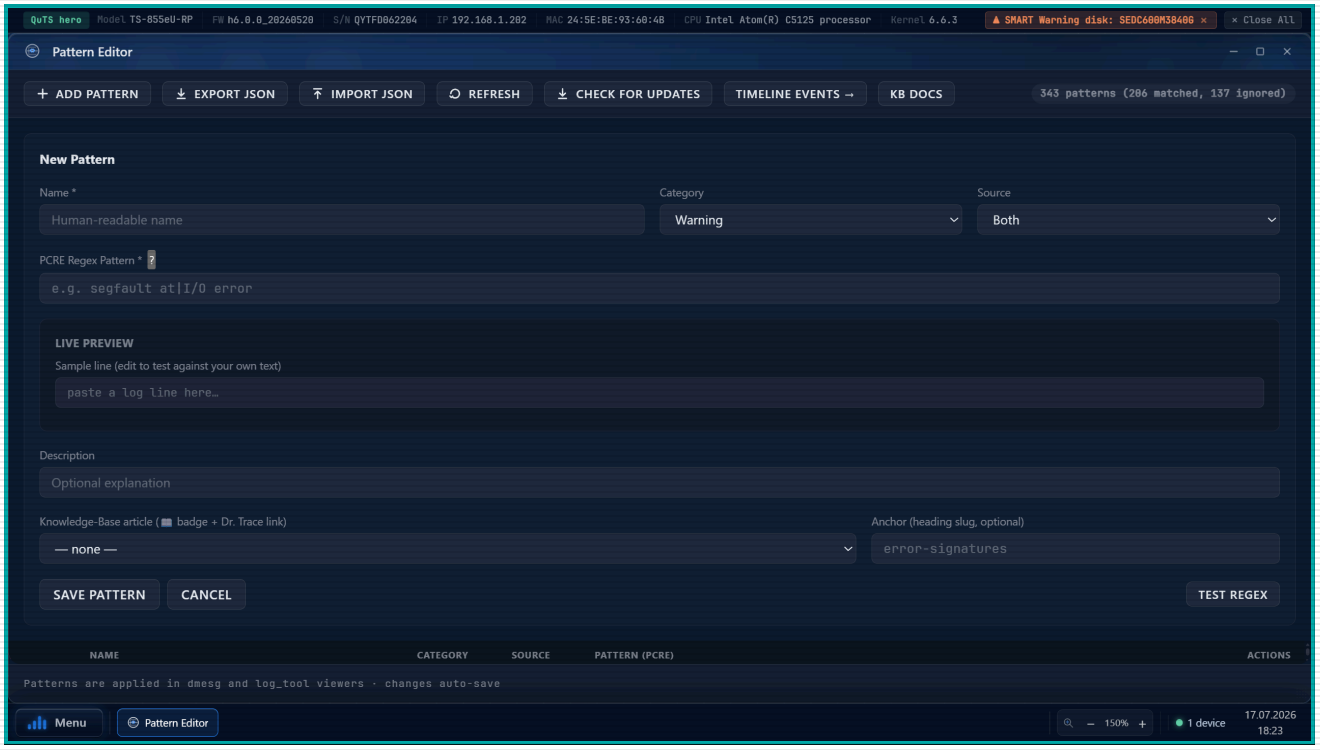


Figure — Creating a pattern: name, category, source and PCRE, with the live-preview tester and a Knowledge-Base link.

The form has a : type a regex and a sample line, and the matched fragment is highlighted in place (capture groups tinted separately), with a symbol-by-symbol explanation underneath — so you see exactly what the pattern catches before you save. flags an invalid expression.

Turning a real log line into a pattern

You rarely write patterns from scratch. In the (and Dr. Trace), . The editor opens pre-filled with a version of that line — the variable parts are automatically masked into regex classes:

- interface names → `(?:eth|bond|br|tun|tap|vpn|wg|ppp|wlan)\d+`
- disks → `sd[a-z]+`, `nvme\d+n\d+`, `md\d+`, `dm-\d+`

- numbers → `\d+`, IP/MAC/UUID/PCI addresses and hex → their standard forms
- runs of spaces → `\s+`

Each type is a [illegible] in the preview panel — flip any off if you want it literal. The same right-click menu offers [illegible] (to silence a benign line) — see ignore rules below.

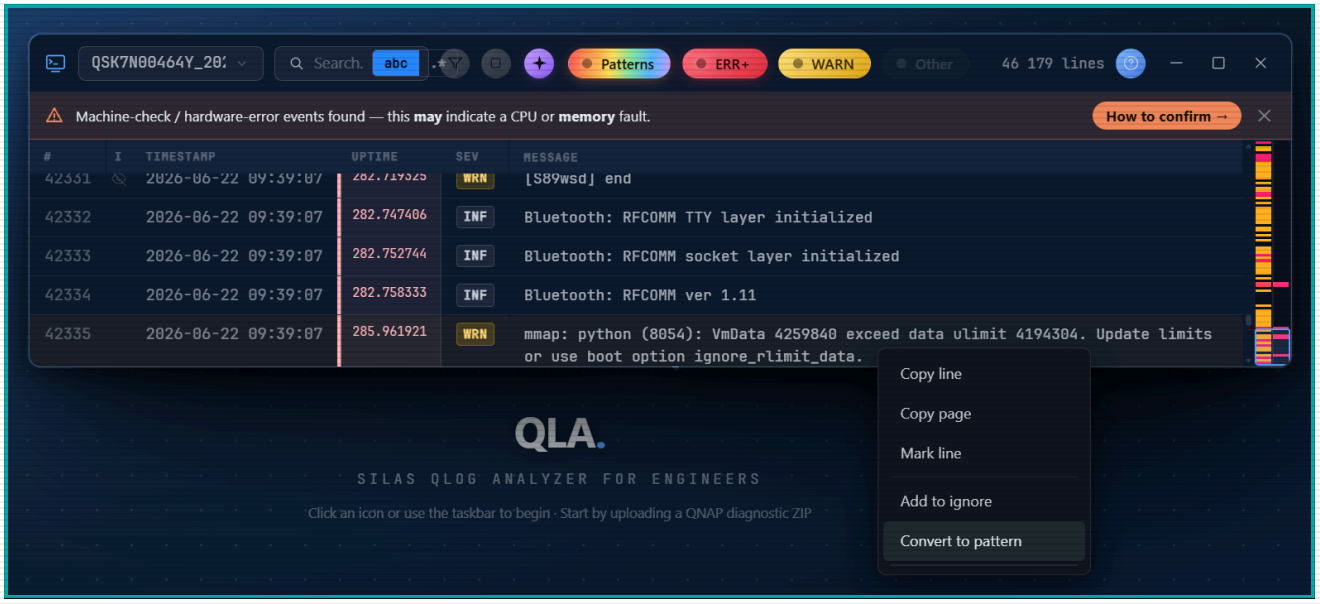


Figure — Right-click a Kernel Log line → [illegible] (alongside Copy line / Copy page / Mark line / Add to ignore).

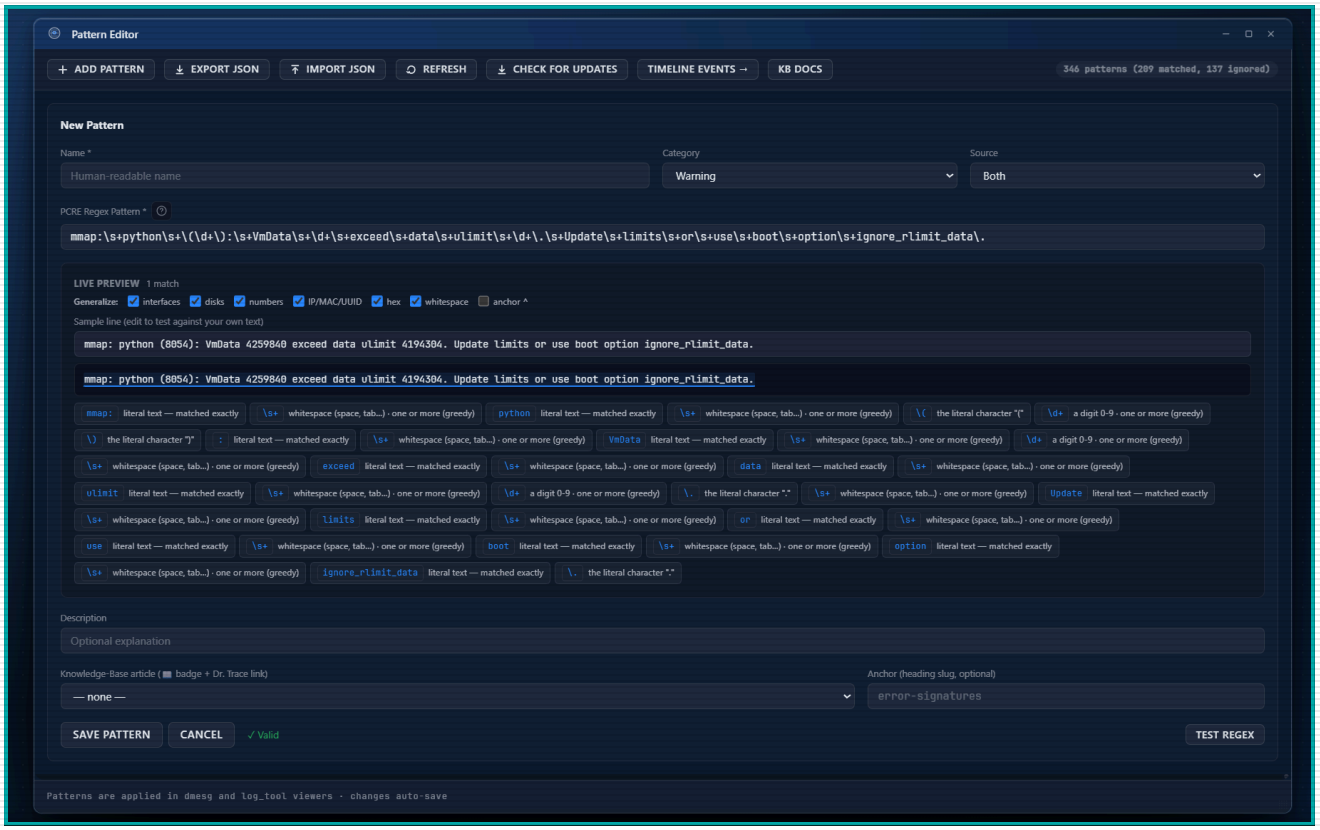


Figure — The Pattern Editor opens pre-filled: the generalised PCRE, the [illegible] toggles (interfaces · disks · numbers · IP/MAC/UUID · hex ·

whitespace · anchor), a live match highlighted on the sample line, and a chip-by-chip explanation of every token — with a green [illegible] and [illegible].

The Regex Helper (the button)

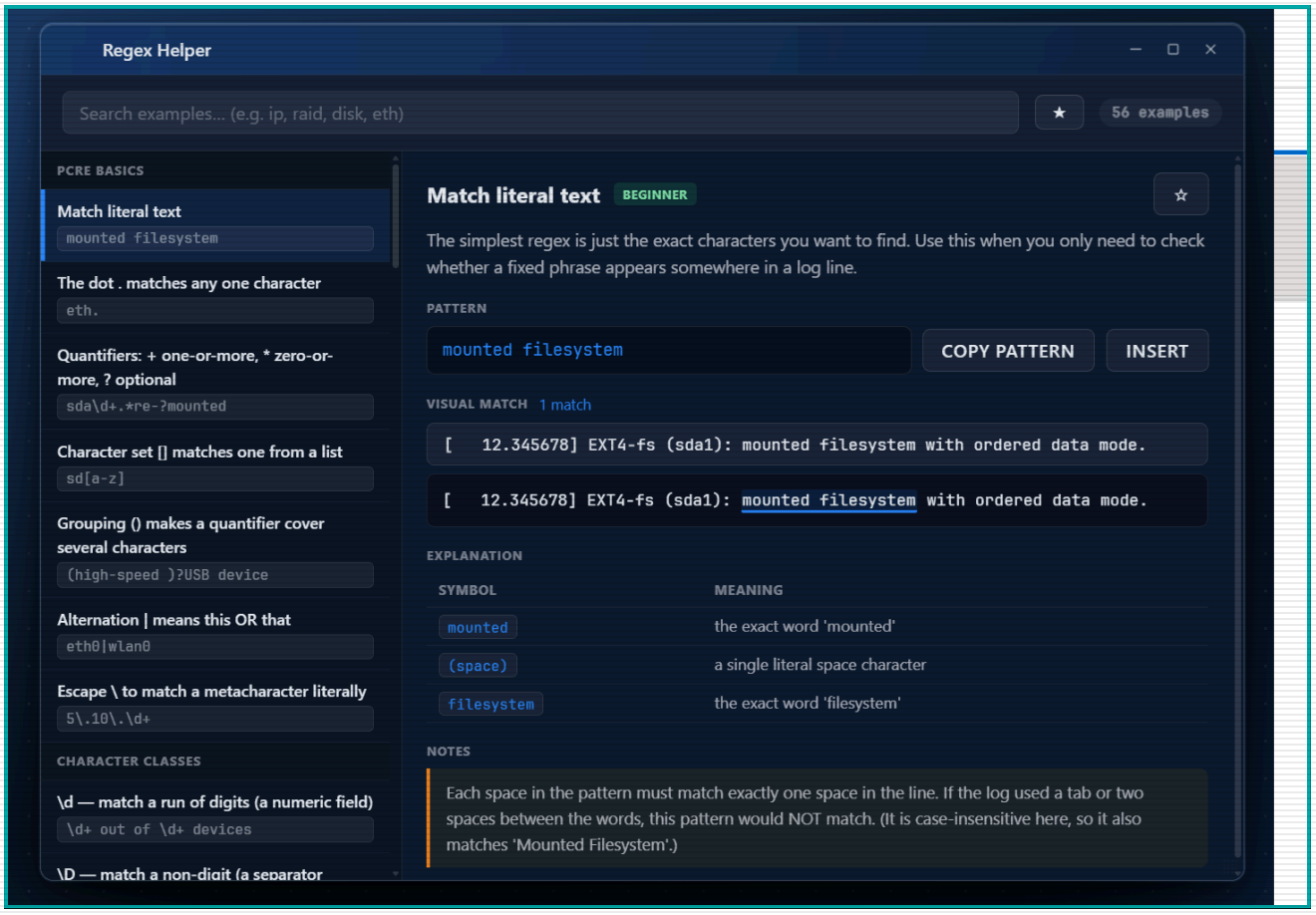


Figure — The Regex Helper: searchable examples with a live visual match, an explanation table, notes, and Copy Pattern / Insert.

Next to the regex field is a button that opens the — an offline, searchable cookbook of ~50 curated examples aimed at Linux/QNAP logs (not generic regex). Type `ip`, `raid`, `disk` or `eth` to filter. Each example shows the pattern, a on an editable sample line, a plain-English explanation table, a "common mistake" note, and / (drops it straight into the editor). Star your favourites. Categories run from PCRE basics through anchors and groups to ready-to-use Linux/QNAP patterns.

> The same examples are also a **browsable reference in this guide** — see 0028. Regex Cookbook.

Ignore rules — silencing benign noise

Ignore rules are the opposite of detection patterns: they mark expected, harmless lines so they stop competing for attention (see 0006. Reading the

Kernel Log (dmesg) for exactly what "ignore" suppresses). In the Pattern Editor's section you can an ignore rule, each with a , a (why it's ignored), and a (all viewers, or dmesg-only, events-only, var/log-only).

Import, export & updates

- – move your pattern set between instances or back it up.
- (admin) – pull newly-curated patterns, KB articles and disk rules from the update server.
- (admin) – author the Knowledge-Base articles that patterns link to (cause + workaround + provenance).

Related editors

Two sibling editors reuse the same idea:

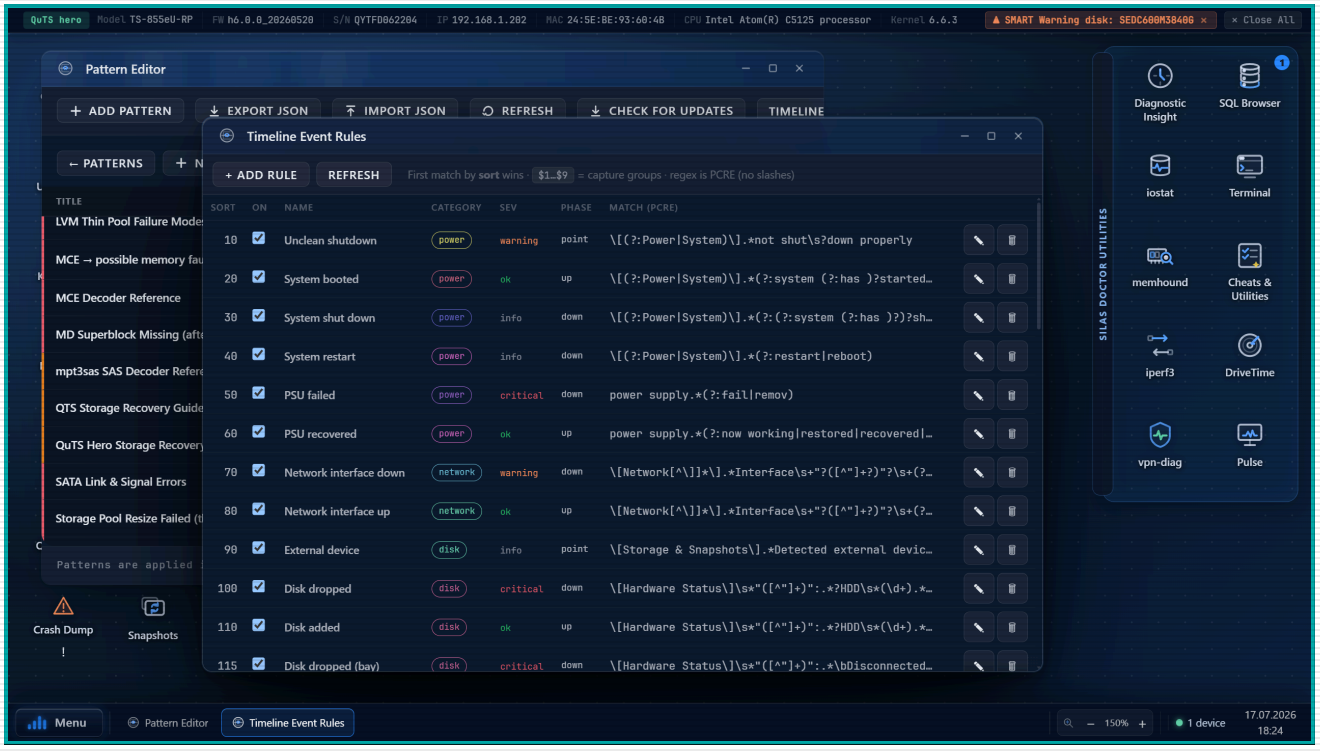


Figure – Timeline Event Rules: the sorted PCRE rules (power / network / disk, with severity and up/down/point phase) that populate the Dr. Trace timeline.

- – the signatures that turn log lines into the incident/life-event lane in Dr. Trace.
- – regexes that classify a drive model into a tier (Desktop / NAS / Enterprise / Surveillance / Archive / ⛔ Blacklist), driving the labels you see in SMART.

> **Best practice.** When a case reveals a signature the tool didn't flag, Convert to pattern it on the spot, generalise with the toggles, verify

> in the live preview, add a category and (ideally) a KB link – and every future bundle with that fault is caught automatically.

Watch it

> In-app demo: The Pattern Editor – 353 signatures, 216 matching this bundle, each with its category and raw PCRE.

> In-app demo: Disk Type Patterns – the 114 model regexes that sort a drive into Desktop, NAS or Enterprise.

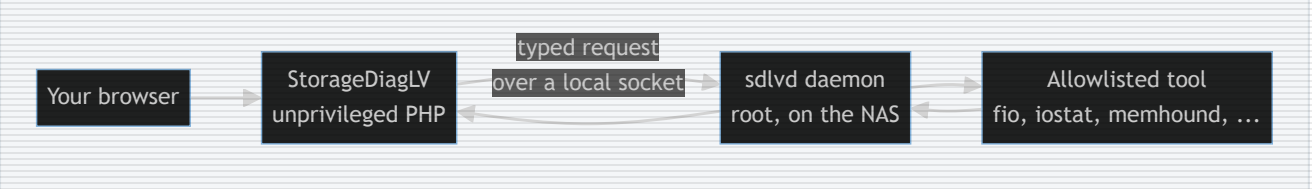
> In-app demo: Writing a new signature with the Regex Helper – it explains, previews and inserts the PCRE for you.

LIVE DIAGNOSTICS & DOCTOR UTILITIES

Everything so far in this guide is — reading an uploaded bundle. StorageDiagLV also has an side: tools that run . These are grouped in the dock and are mostly admin-gated. This local toolset can also be installed straight on a client NAS from the **sdu.qpkg** package ( ), where it runs entirely on that box — reading the drive's own on-box data (note: and other deep telemetry come from the live drive, not from an uploaded bundle's logs).

> **Safety first.** Live tools hand operations to a **companion daemon that runs as root** on the NAS. Use **Safe mode** and never expose the app on the public internet without HTTPS, strong credentials and a firewall — see 0099. About, privacy and licence. Passive bundle analysis has none of this exposure; these live tools do.

How the live tools reach the NAS



The PHP side never runs commands itself — it can only ask the daemon to run one of a fixed, set of operations. Configuration can disable operations, never add them.

The live tools

- — a local system diagnostic audit run on the NAS (requires the QNAP Helpdesk package). A one-click health sweep of the connected machine.
- Streams live per-disk throughput, latency and IOPS, and includes a that finds the weak disk dragging a RAID. The scan can generate its own test load — sequential or random, read, write or a realistic read-write pattern — for anything from , or run in mode that generates no load and simply watches an existing workload (for example a RAID rebuild or a busy iSCSI volume). The first seconds of a test are automatically, so drives spinning up from standby don't skew the result; the skip window is adjustable under (set it to 0 to keep everything).

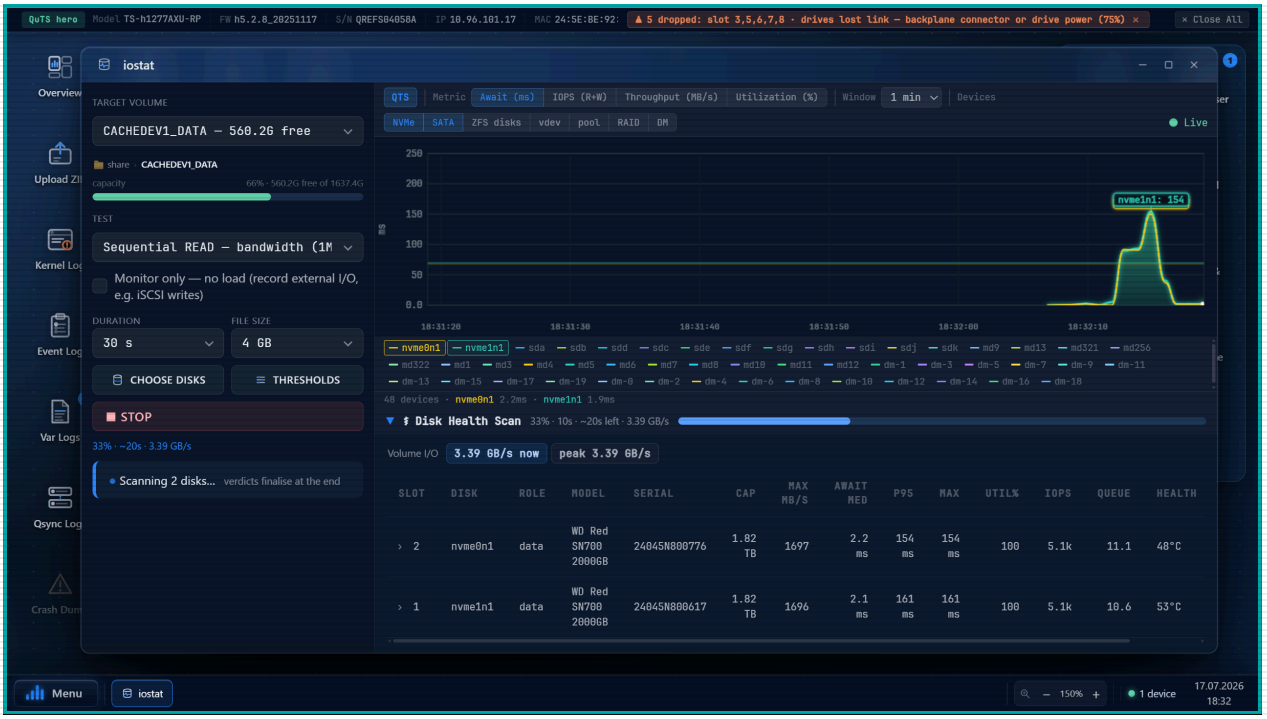


Figure — The live iostat monitor running a Disk Health Scan: await-latency chart plus a per-disk table (bandwidth, latency, IOPS, queue, temperature).

- **memhound** — An original RAM tester with memtest86-style patterns plus a fast test. Its principle is never a false positive — a flagged fault is real to ~99.999 %; a clean pass is strong but not absolute, so when it truly matters confirm with a full run (see 0099. About, privacy and licence).



Figure — memhound during a Quick pass: live memory map, fault counters, and the status panel.

- Live CPU/memory/load meters and a process , with actions (kill, renice, set CPU governor, drop caches), plus tabs for network, VMs, databases and logs. Two of its tabs are diagnostic tools in their own right – a and an – both described below.

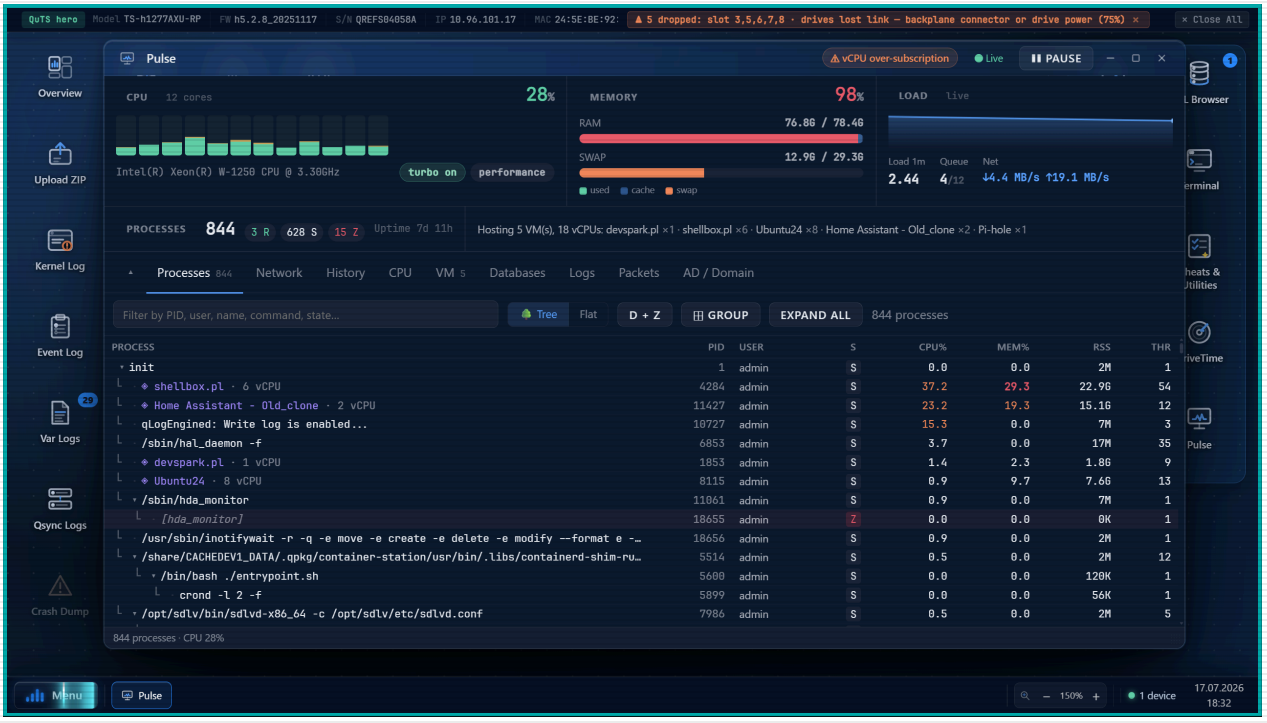


Figure – The Pulse cockpit (Processes tab): CPU/memory/load meters, VM summary, and a filterable process tree.

- between two NAS units (bandwidth, retransmits).
- One window for the questions a network complaint actually raises: a (every device it can reach, with vendor, interface and MAC), a table that puts every resolver side by side so you can see who answers, how fast, and who disagrees, a test (the NAS listens, a client pushes), a that separates latency from loss from MTU discovery, and a you can filter, dissect and follow a stream in.
- A answers "is this CPU throttled or healthy" and reports the result as an explicit , not a score – a busy NAS cannot be benchmarked honestly. A run loads every core with the turbo, governor and cache state you chose, for when you need a figure to compare against a reference machine.
- – per-disk SMART/FARM attribute from an hourly on-NAS collector: the time-series companion to the point-in-time SMART cards

(0008. Reading S.M.A.R.T. disk health), so you can see a value climbing before it trips a hard limit.

- [illegible] – an interactive [illegible] shell on the NAS (admin only), running through the daemon.
- [illegible] – a palette of ready-to-run recovery/diagnostic commands (Storage, LVM, ZFS, System, Network…) that you paste into the Terminal, so you don't have to remember them.
- [illegible] – link quality for a VPN: MTU, jitter, loss, bandwidth and stability.

> **Best practice.** Use the live tools to confirm and act once passive analysis has pointed you at a subsystem: iostat/Disk Health Scan to pin the slow disk a bundle hinted at, memhound to test RAM a mixed-DIMM warning flagged, DriveTime to show a customer the trend behind a SMART verdict.

Capturing and reading network traffic (Pulse → Packets)

The [illegible] tab captures live traffic on the NAS and lets you read it Wireshark-style, right in the browser – for questions like "is the client even reaching the NAS?", "what happened when the share wouldn't open?" or "why is this transfer slow?".

[illegible]

- [illegible] – pick a network port, or leave it on [illegible]. On a QNAP with a Virtual Switch, traffic between a client and the NAS is often not visible on the switch's own interface, so "all interfaces" is the safe default that never misses it.
- [illegible] – the default [illegible] keeps the capture focused on file-sharing traffic; there are also [illegible], [illegible] and per-protocol presets, or you can type your own (an IP address, a port, or a filter expression such as host 10.0.0.5 and port 445).

[illegible]

- [illegible] With no line or seconds limit set, the button reads [illegible]. The NAS writes the complete capture straight to a file and [illegible], even at full line rate. Do whatever you need to reproduce, then press [illegible] and the capture loads. This is the way to "capture everything and be certain nothing was missed".
- [illegible] Tick [illegible] to watch packets scroll past as they are captured. It still records losslessly underneath; the on-screen list is a rolling

view that can lag behind under a heavy load, but [illegible].

- [illegible] Set a packet count or a number of seconds to grab a short, fixed sample instead.

[illegible] Type in the [illegible] – a filename, a host, an IP address, or a status such as a "not found" error – and it scans the [illegible], not just the rows on screen. So if you created a folder over SMB, or hit a permission error, during the capture, you will find it even in a capture of hundreds of thousands of packets. If the kernel ever had to drop packets under an extreme load, a [illegible] count is shown – the tool tells you honestly rather than silently losing data.

[illegible] Each row reads like a traced operation: the protocol, the [illegible] (for example SMB2 mkdir), the [illegible] (the file or folder), and the [illegible] – a green ✓ SUCCESS or a red ✗ OBJECT_PATH_NOT_FOUND. Click a row for the full layer-by-layer breakdown and the raw bytes. Protocols understood include SMB2/3 (with the file/folder path and result), NFS, iSCSI, AFP, FTP, DNS, and the domain protocols [illegible], [illegible] and [illegible].

[illegible] saves the complete capture as a standard file to open in Wireshark; [illegible] loads an existing capture file – for example one a customer sent – to analyse it here.

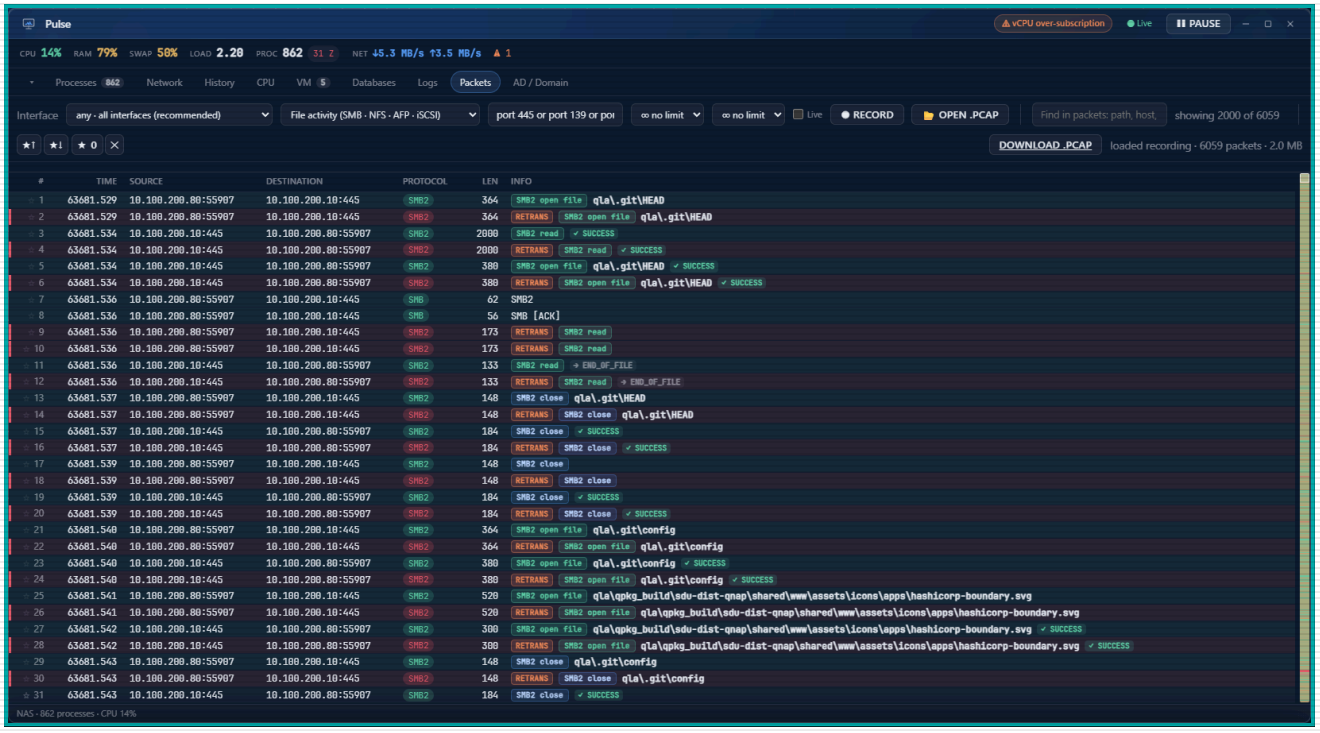


Figure – The Packets analyzer after a capture: each SMB2 operation is shown with its file/folder target and a ✓/✗ outcome, and the search box matches across the whole captured dump.

Checking an Active-Directory join (Pulse → AD / Domain)

The tab reads a capture and produces a green/amber/red checklist of everything that makes — or breaks — a NAS in an domain:

- DNS (the `_msdcs` service records)
- reachability and (Kerberos needs the clock within about five minutes of the DC)
- whether sign-in actually used rather than falling back to
- and (Windows 11 24H2 requires SMB signing by default)
- directory bind, time sync, and any

Each failing check comes with the fix to try. It needs : run a capture (or import a customer's capture file) while the NAS logs in to, or joins, the domain. On a standalone NAS with no domain traffic it correctly reports "no domain traffic" rather than a false alarm.

Live examples

Short screen recordings of the live tools working on a real NAS.

> In-app demo: iostat Disk Health Scan — spotting the one slow disk dragging a whole RAID's performance down.

> In-app demo: VPN diagnostics — tracing a TTL / ISP / mis-configuration issue behind unstable VPN throughput.

> In-app demo: Run Audit — the local system diagnostic sweep on a connected NAS.

> In-app demo: Diagnostic Insight writing its own report from the NAS it runs on — no bundle needed.

> In-app demo: Pulse — 855 processes and RAM at 85 %, with a warning that the VMs want more vCPUs than the box has.

> In-app demo: cpuhound — a three-second probe, reported as a lower bound rather than a benchmark.

In-app demo: cpuhound Full — every core loaded, with turbo and cache state under your control.

In-app demo: NetScope — sweeping the LAN and naming every device, vendor, interface and MAC it can see.

In-app demo: NetScope — eight resolvers in one table: who answers, how fast, and who disagrees.

In-app demo: NetScope — iperf3 built in: the NAS listens, a remote client pushes, the throughput lands in the panel.

In-app demo: NetScope — the VPN probe reporting latency, loss and MTU discovery separately.

In-app demo: NetScope — picking a packet, reading the dissection and following the stream.

In-app demo: NetScope — creating a folder on the share and watching the SMB2 call for it arrive in the capture.

DISK PRESENCE HISTORY

The window is a per-bay timeline of . It reconstructs connect / disconnect / SMART events from the QNAP event log, so you can see a drive's whole life in the NAS at a glance.

> **How to open it:** Start menu / desktop → **Disk History**. Or from here: @@OPEN:DiskHistory|Open Disk History@@



Figure — Disk Presence History: per-bay presence over time; a dropped bay turns red / struck-through with the fault annotation beneath it.

How to read it

Each entry on the timeline is a with a one-line summary — for example "15/16 present · 1 dropped" — and a grid of . There are two chips per physical slot: the logical and the physical , so you can line up the bay number with the drive.

- chip = the disk was present and healthy in that window.
- chip = present but flagged (a SMART/health event on it).
- chip = the disk in that window (disappeared from the bay).

When a disk drops, the fault that caused it is printed underneath, e.g. "host: 3.5" SATA HDD 4 — Medium error (14:22:24) · Read I/O error (14:22:24)". That tells you the drive didn't just vanish — it hit uncorrectable read errors and the controller dropped it.

Why it matters

A disk that [illegible] across several snapshots is the classic signature of a dying drive or a flaky cable/backplane — often the real story behind "the volume keeps degrading." The presence timeline makes that pattern obvious in a way a single SMART snapshot cannot.

> **Best practice.** Find the bay that goes red, read the fault annotation (medium/read errors → the disk; link resets with no media errors → suspect cable/backplane), then confirm on that bay in **SMART** (0008. Reading S.M.A.R.T. disk health) and cross-check the timing in **Dr. Trace**.

Watch it

> In-app demo: Disk Presence History — 3 375 disk events across 16 bays, all still populated.

VIRTUAL MACHINES

The Virtual Machines window reads the NAS's KVM/QEMU (Virtualization Station / QVS) configuration and logs from the bundle, and lays out every VM with its full hardware profile.

> **How to open it:** Start menu / desktop → **Virtual Machines**. Or from here: @@OPEN:Vms|Open Virtual Machines@@

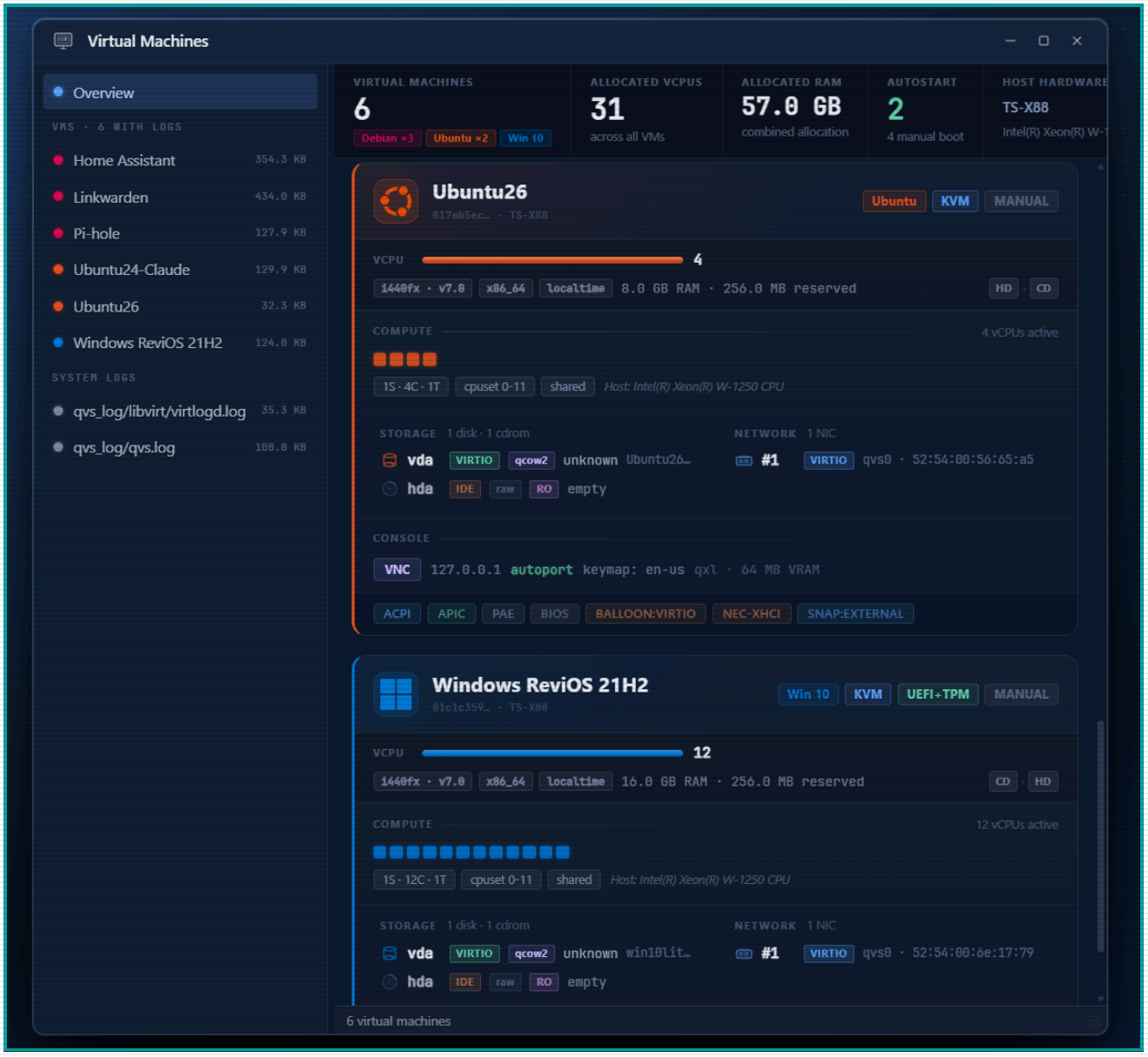


Figure — Virtual Machines: the allocation overview and a per-VM card (OS, vCPU, machine type, storage, network, console, feature badges).

Overview strip

At the top: how many are defined, total , total , and how many are set to . These totals are the first thing to read — see the oversubscription note below.

Per-VM card

Selecting a VM shows its complete profile:

- — name, guest-OS badge (Ubuntu / Debian / Windows…), , boot mode (MANUAL / AUTOSTART), UUID and host node.
- — allocated vs the machine maximum (e.g. 8 / 128), plus a live compute grid, `cpuset` pinning, shared/dedicated, and the host CPU model.
- — chipset (e.g. i440fx), version, architecture (x86_64), clock (localtime/UTC), and reserved memory.
- — each virtual disk with its bus ( ), format ( ), read-only flag, and backing image.
- — each NIC with model (/ e1000…) and MAC.
- — VNC endpoint, keymap, and video (VRAM).
- — ACPI, APIC, BIOS/UEFI, , USB controller, snapshot mode, etc.

How to read it

1. If allocated vCPUs and RAM across all VMs far exceed the host's real cores/RAM, the host is – a common cause of "everything is slow" and of guest CPU-steal (which shows live, see 0017. Live diagnostics and Doctor utilities).
- A `raw` disk on `IDE` performs far worse than `qcow2` / `raw` on ; a non-VIRTIO NIC likewise. These explain slow-VM complaints.
- The left list carries each VM's captured log plus the hypervisor logs (`virtlogd.log` , `qvs.log`) – open them when a specific VM misbehaves.

> **Best practice.** A NAS that hosts VMs and reports storage or memory pressure is often simply over-committed – read this window alongside Memory (0013. Memory, OOM and crash dumps) and Pulse before blaming the hardware.

Watch it

> In-app demo: The Virtual Machines viewer saying plainly that this bundle carries no VM configuration, and where it looked.

■ CONTAINER STATION

The Container Station window surfaces the NAS's container runtimes (Docker / containerd and LXC/LXD) and the related kernel/HAL events found in the logs.

> **How to open it:** Start menu / desktop → **Containers**. Or from here: @@OPEN:Container|Open Container Station@@

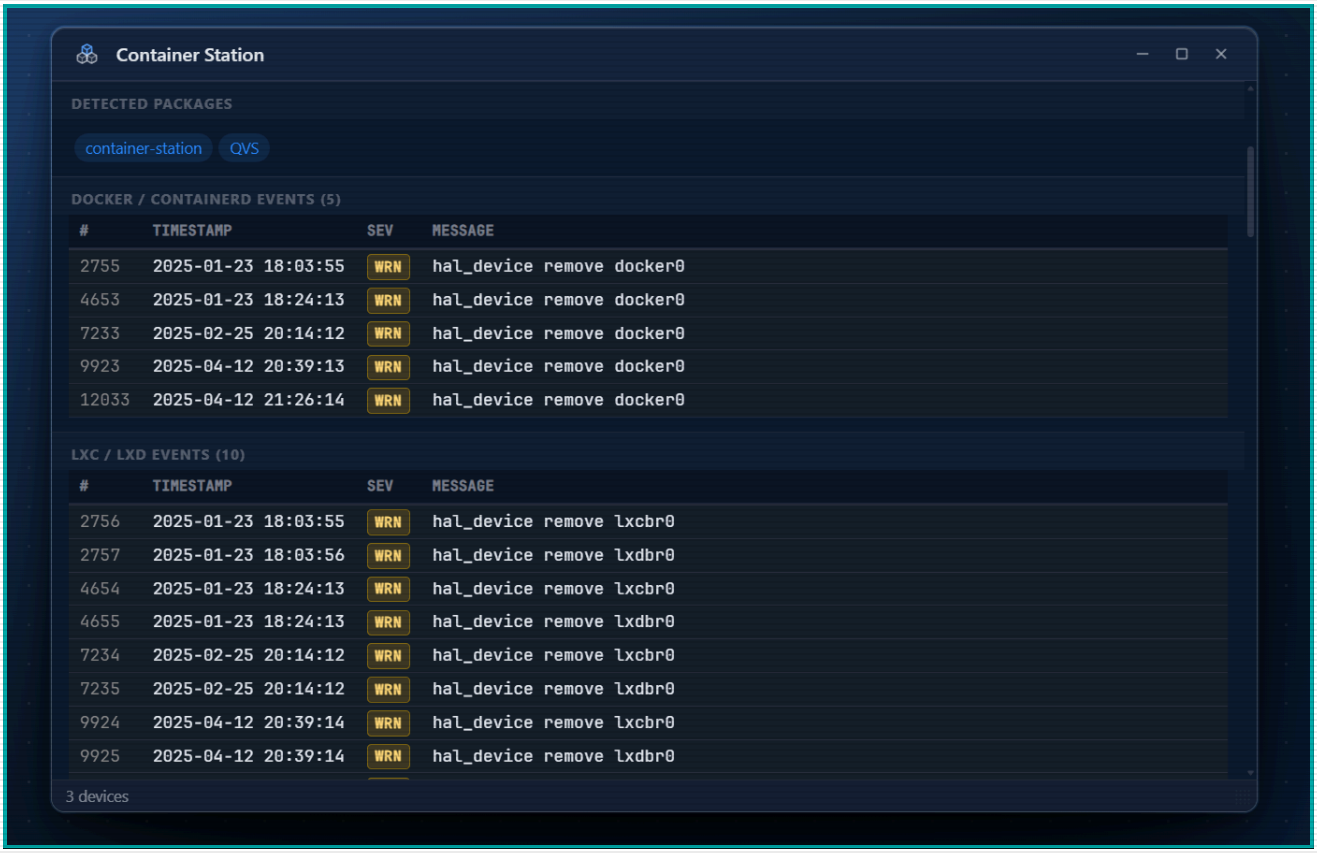


Figure — Container Station: detected runtimes plus the Docker/containerd and LXC/LXD kernel events.

What you see

- Detected packages — the container-related QPKGs installed on the NAS (Container Station, and the QVS virtual-switch that backs container networking).
- Docker / containerd events — kernel/HAL lines about the Docker world, e.g. repeated `hal_device remove docker0` warnings (the Docker bridge being torn down / recreated).
- LXC / LXD events — the same for LXC containers, e.g. `hal_device remove lxcbr0` (the LXC bridge).

How to read it

- **Container Station** — if `container-station` is in `Detected packages`, the NAS is running containers — worth knowing, because a container can be the thing consuming RAM or I/O.
- **Network churn** — repeated `hal_device remove docker0` / `lxcbr0` warnings mean the container network bridge is being added and removed over and over. Occasional entries are normal (start/stop); a `storm` of them points at a container repeatedly restarting or a networking problem.
- **Tie to OOMs** — a container hitting its `memory limit` shows up as an OOM event scoped to a cgroup (see 0013. Memory, OOM and crash dumps) — that is a `container` limit, not the whole NAS running out of RAM. Container network churn here, next to those cgroup OOMs, usually points at one misbehaving container rather than a host fault.

> **Best practice.** Use this window to rule containers in or out early: a NAS "problem" that is really a single crash-looping Docker container looks very different once you see the bridge-remove storm and its cgroup OOMs lined up.

Watch it

> In-app demo: Container Station — the detected runtimes and the docker0 interface-removal warnings.

SNAPSHOTS & REPLICATION

The Snapshots window shows the NAS's volume-snapshot inventory and its replication (off-box backup) jobs, read from the bundle.

> **How to open it:** Start menu / desktop → **Snapshots**. Or from here: @@OPEN:Snapshots|Open Snapshots@@

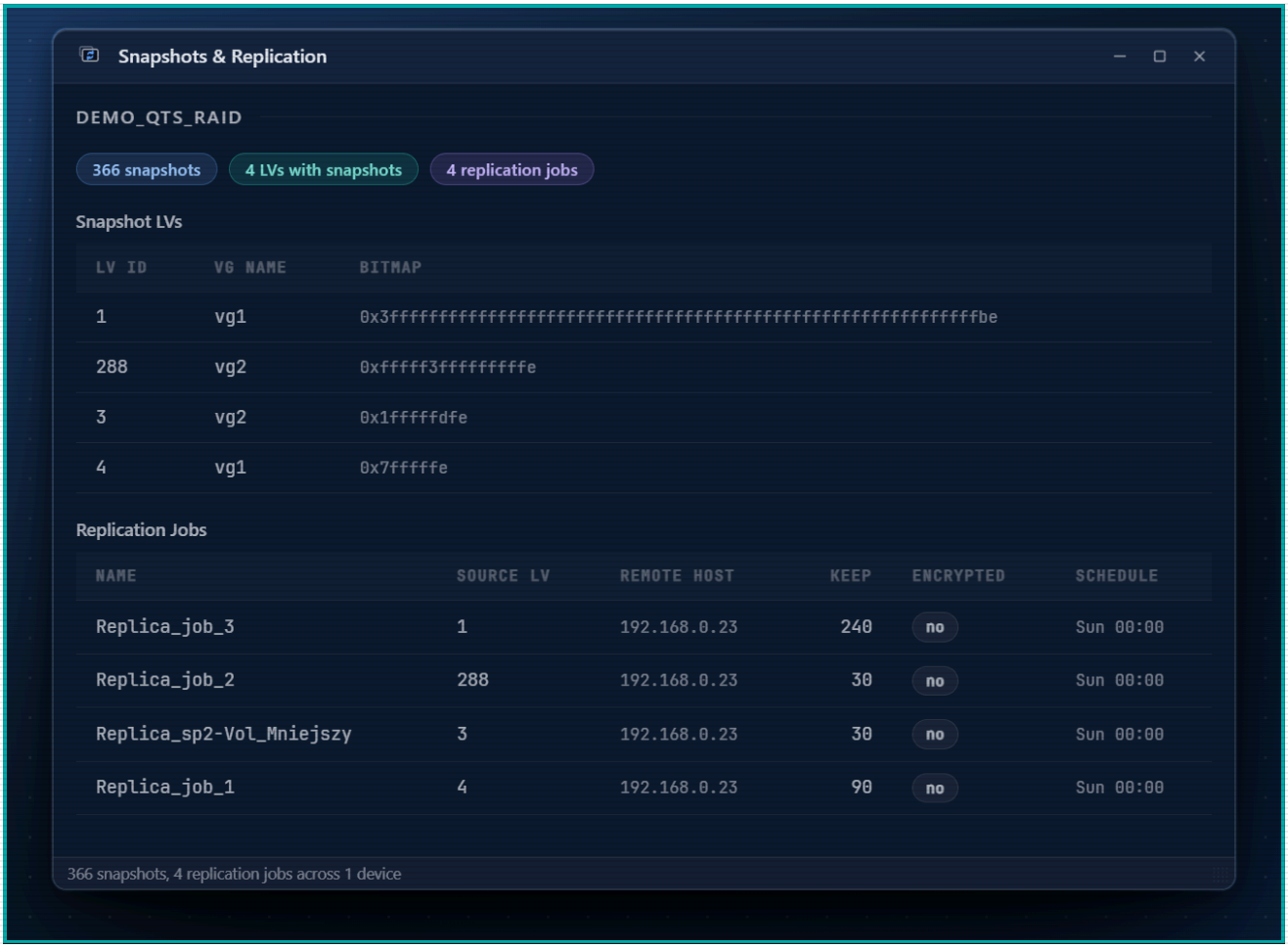


Figure — Snapshots & Replication: the snapshot LVs and the replication jobs (source, remote host, retention, encryption, schedule).

Snapshot LVs

A table of the logical volumes that have snapshots — , , and the snapshot (a hex map of which blocks the snapshots cover). The header chips tell you the totals at a glance: how many snapshots exist and across how many volumes.

Replication Jobs

Each replication (backup-to-another-NAS) job shows:

- and — what's being replicated.
- — the destination NAS.

- Keep — retention (how many copies are kept).
- Encrypted — whether the transfer is encrypted (Yes / No).
- Schedule — when it runs (e.g. Sun 00:00).

How to read it

1. Is the data currently protected? A volume with snapshots is protected against accidental deletion locally; a Healthy replica is what protects it against the NAS itself failing. Check that the important volumes have a job pointing at a Healthy one.
2. Retention is too short? A very small Keep or an infrequent schedule means little history to roll back to.
3. Encryption An Encrypted job sending data to a remote host over the network is worth flagging in a security-sensitive environment.

> **Best practice.** When a customer reports "I lost data", this window tells you immediately whether a snapshot or replica exists to recover from — and how far back it goes.

Watch it

> In-app demo: Snapshots and Replication — 366 snapshots over four logical volumes, with four replication jobs.

SYSTEM TIMELINE

The Timeline window is a straight, filterable chronological view of QNAP system events (from the event log). It complements the other two event views: the Event Log is the raw table (0005. Reading the QNAP Event Log), Dr. Trace correlates events into ranked verdicts (0007. Reading Dr. Trace (the incident timeline)), and the Timeline sits in between — the events in time order, grouped by day, with category filters.

> **How to open it:** Start menu / desktop → **Timeline**. Or from here: @@OPEN:Timeline|Open the Timeline@@



Figure — System Timeline: chronological QNAP events with category filters and severity badges (here a PSU fail → recover pair).

How to read it

- [illegible] — Storage · Power · System · Disk · Network — to cut the noise to the subsystem you're investigating.

- Each event shows a [illegible] (INF / OK / WRN / CRT), the time, and the raw QNAP text (e.g. "[Hardware Status] Host: First power supply failed or was removed").
- [illegible] a red [illegible] followed by a green [illegible] tells you the fault cleared — and how long it lasted (read the two timestamps).
- [illegible] (RAID scrub started/finished, boots) are shown too, so you can confirm housekeeping ran and see what was happening around an incident.

> **Best practice.** Use the Timeline for the sequence of what happened — "PSU failed at 04:39, recovered at 05:03, scrub started at 02:00" — then jump to **Dr. Trace** for the correlated verdict and to the **Kernel Log** for the kernel-side detail of the same moments.

Watch it

> In-app demo: System Timeline — 1 681 QNAP events across five years on a single filterable axis.

ISCSI

The window shows the NAS's iSCSI block-storage configuration — targets, initiators and LUNs — read from the bundle.

> **How to open it:** Start menu / desktop → **iSCSI**. Or from here: @@OPEN:Iscsi|Open iSCSI@@

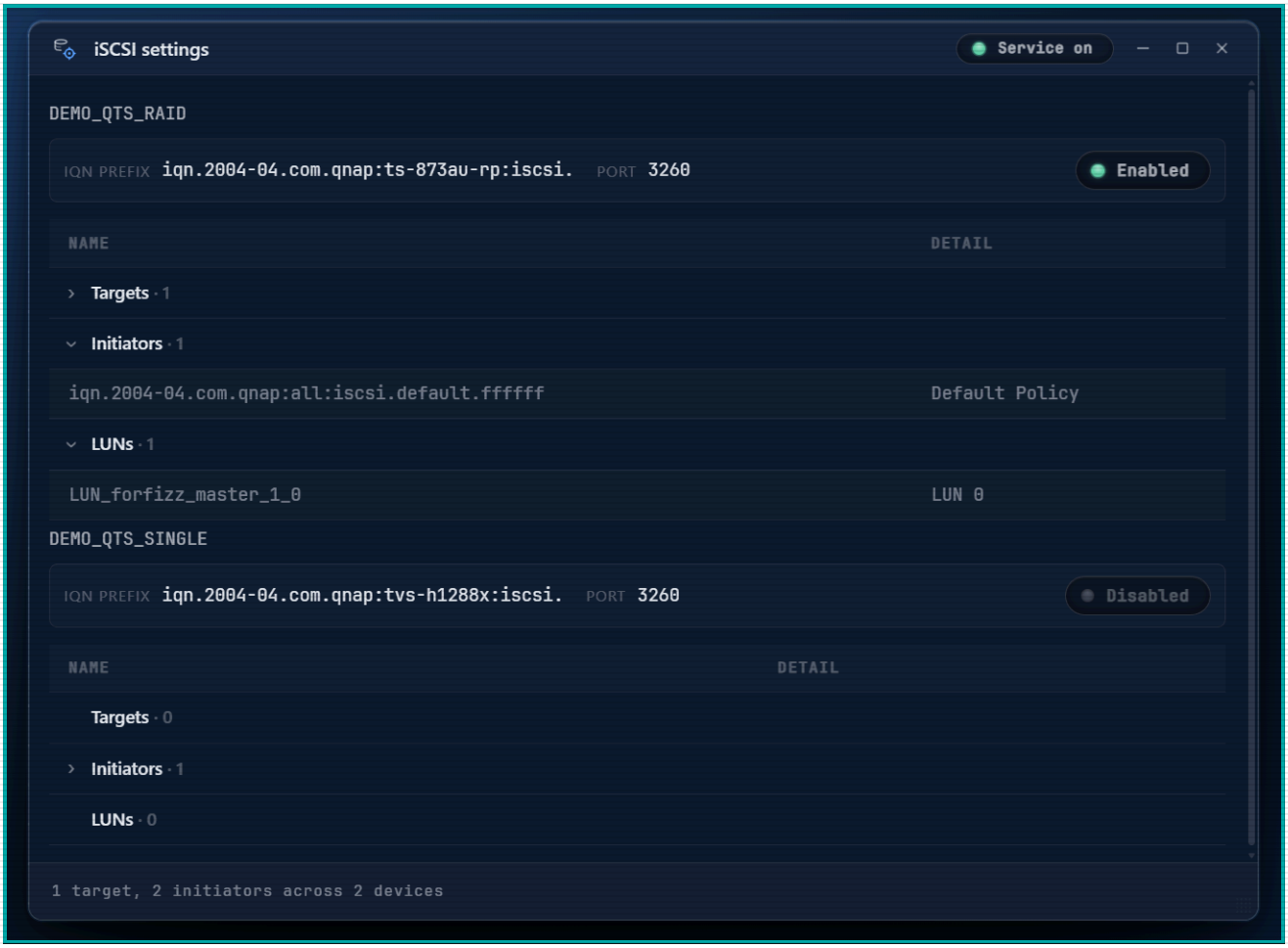


Figure — iSCSI settings: service state, IQN prefix and port, with Targets / Initiators / LUNs.

What you see

- — whether the iSCSI service is , with the base (e.g. `iqn.2004-04.com.qnap:ts-873au-rp:iscsi.`) and the listening (3260 by default).
- — each iSCSI target the NAS presents, with its type (e.g. standalone).
- — the clients allowed to connect, with their IQN and access (e.g. a default allow-all policy, or a specific ACL).
- — the block devices exposed, each with its LUN number.

How to read it

1. If a customer's iSCSI mount "disappeared", first confirm the service is here.
2. – check that the expected LUNs are attached to the target the client uses.
3. – a initiator policy means any initiator that knows the target can connect; a tightened ACL restricts it. Worth flagging for a security review.

> **Best practice.** iSCSI issues are usually either "service/target/LUN not configured as expected" (visible here) or a transport problem (network/MTU) – for the latter, cross-check the **Network** window (0014. Network interfaces) and link events in **Dr. Trace**.

Watch it

> In-app demo: iSCSI – the target, its initiator and the LUN behind it.

SHARES (SMB)

The window shows the NAS's SMB/Samba network file shares — who can reach what, and how each share is protected.

> **How to open it:** Start menu / desktop → **Shares**. Or from here: @@OPEN:Smb|Open Shares@@

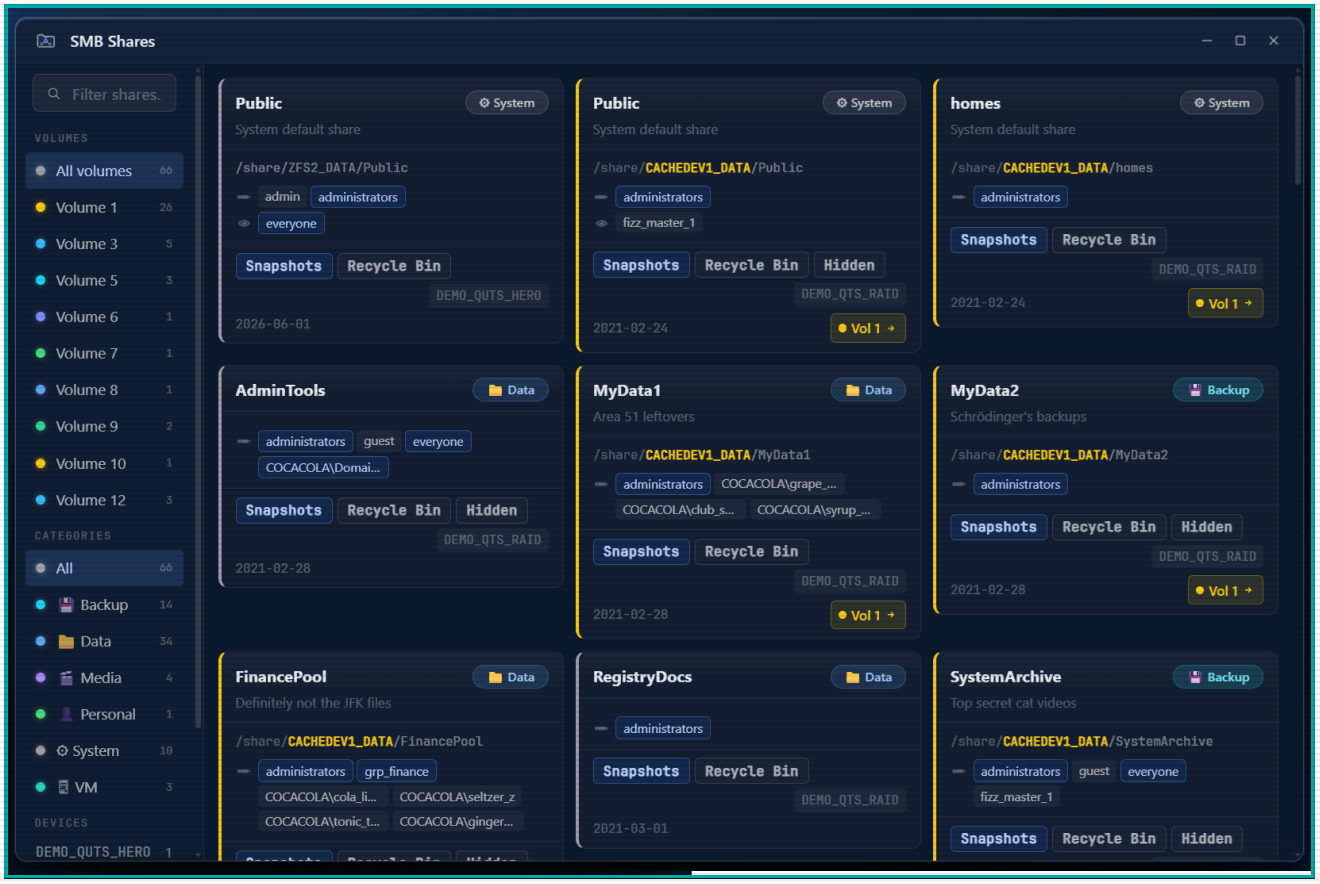


Figure — SMB Shares: per-share cards with path, permission chips, read-only markers and protection badges.

What each share card shows

- and a (built-in default share, e.g. `Public`, `homes`) or (user-created) badge.
- — e.g. `/share/ZFS2_DATA/Public`, so you can see which volume it lives on.
- — the users/groups with access (`admin`, `administrators`, `everyone`, `guest`, or a domain principal like `DOMAIN\Domain Users`). An icon marks entries.
- — , , — the protections/visibility enabled on the share.

- The source device and creation date, and a link to jump to the volume.

How to read it

- **Access exposure** — A share carrying **Everyone** or **Guest** with write access is open to anyone on the network — flag it. A share showing a domain principal means the NAS is AD/LDAP-joined.
- **Recycle bin / Snapshots** — The **Recycle bin** and **Snapshots** badges tell you whether an accidental delete on that share is recoverable — useful when a customer reports lost files.
- **Path / volume** — The path/volume tells you which pool a share sits on — handy when a specific volume is degraded (cross-check 0011. Storage layout — LVM, RAID and DRBD / 0012. ZFS pools (QuTS hero)).

> **Best practice.** Use the sidebar to filter by volume or category (e.g. Backup) when a NAS has dozens of shares, then read permissions and protection badges on the ones that matter.

Watch it

> In-app demo: SMB Shares — 22 shares grouped by volume and category, with per-share access.

PROCESS EXPLORER

The Processes Explorer shows every process that was running when the diagnostic was captured, as a tree, with a summary of the system's load at that moment.

> **How to open it:** Start menu / desktop → **Processes**. Or from here: @@OPEN:Processes|Open Process Explorer@@

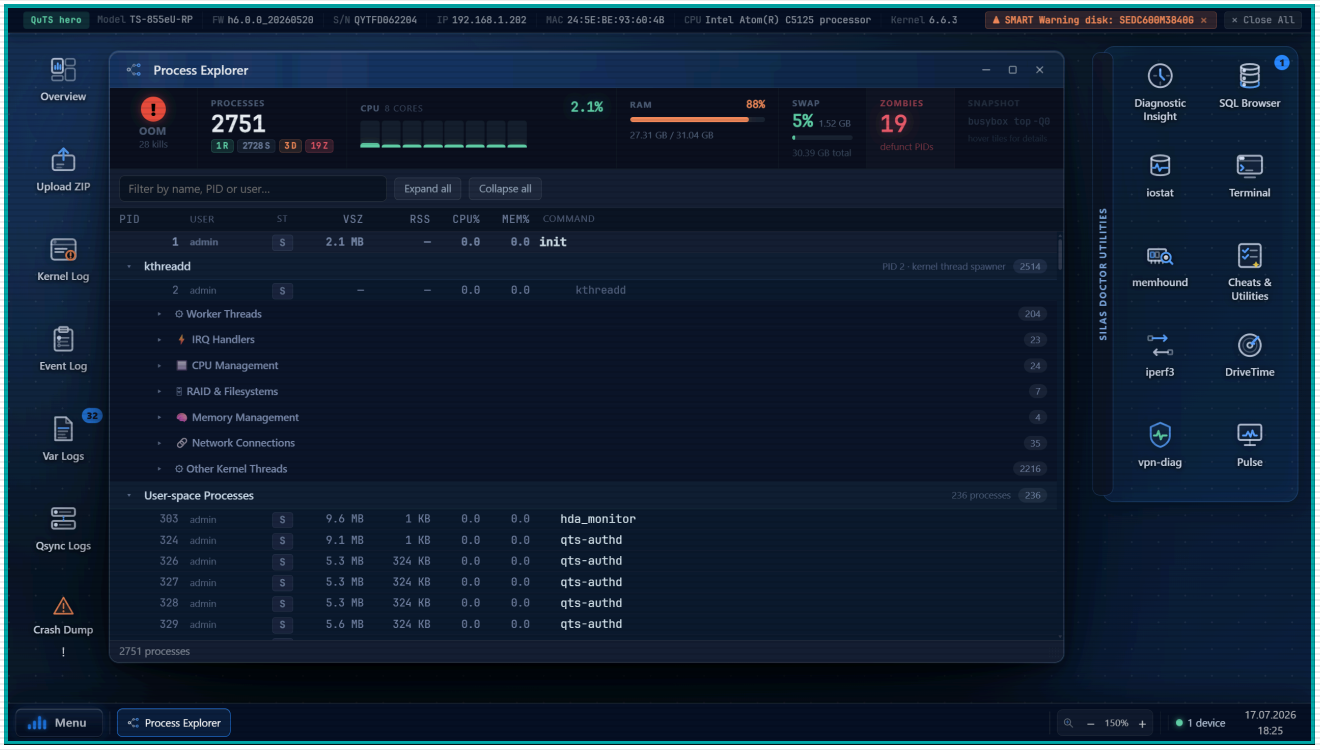


Figure — The Process Explorer: summary tiles (OOM kills · R/S/D/Z states · CPU · RAM · Swap · Zombies) over a filterable, expandable process tree.

The summary tiles

- **Processes** — the total, broken down by state: **R**unning · **S**leeping · **D** (uninterruptible, stuck in I/O) · **Z**ombie (defunct).
- **CPU** — per-core bars and overall %.
- **RAM / Swap** — usage % (a red RAM or high Swap echoes the Memory chapter).
- **Zombies** — count of defunct PIDs.
- **OOM** — how many processes the kernel force-killed (ties to 0013. Memory, OOM and crash dumps).

The process tree

Kernel threads are grouped by subsystem — Worker Threads, IRQ Handlers, CPU Management, RAID & Filesystems, Memory Management, Network Connections, Other — so the hundreds of kernel threads don't bury the real processes.

are listed separately. Columns: . Filter by name/PID/user, and expand/collapse groups.

How to read it

- are stuck in uninterruptible I/O — several of them usually means a (cross-check SMART / the Kernel Log). This is one of the most useful signals here.
- means a parent process isn't reaping its children — a misbehaving service.
- — sort/scan the MEM% / CPU% columns to find what was consuming the box.
- together confirm the NAS was memory-starved at capture time.

> **Note.** This is a **snapshot** taken from the bundle (a `top` capture), not live. For live process control (kill, renice, CPU governor) on a connected NAS, use **Pulse** (0017. Live diagnostics and Doctor utilities).

Watch it

> In-app demo: Process Explorer — 1 539 processes, RAM at 91 %, and 67 defunct PIDs worth asking about. In-app demo: Close-up on Expand all, opening the tree down to the defunct processes.

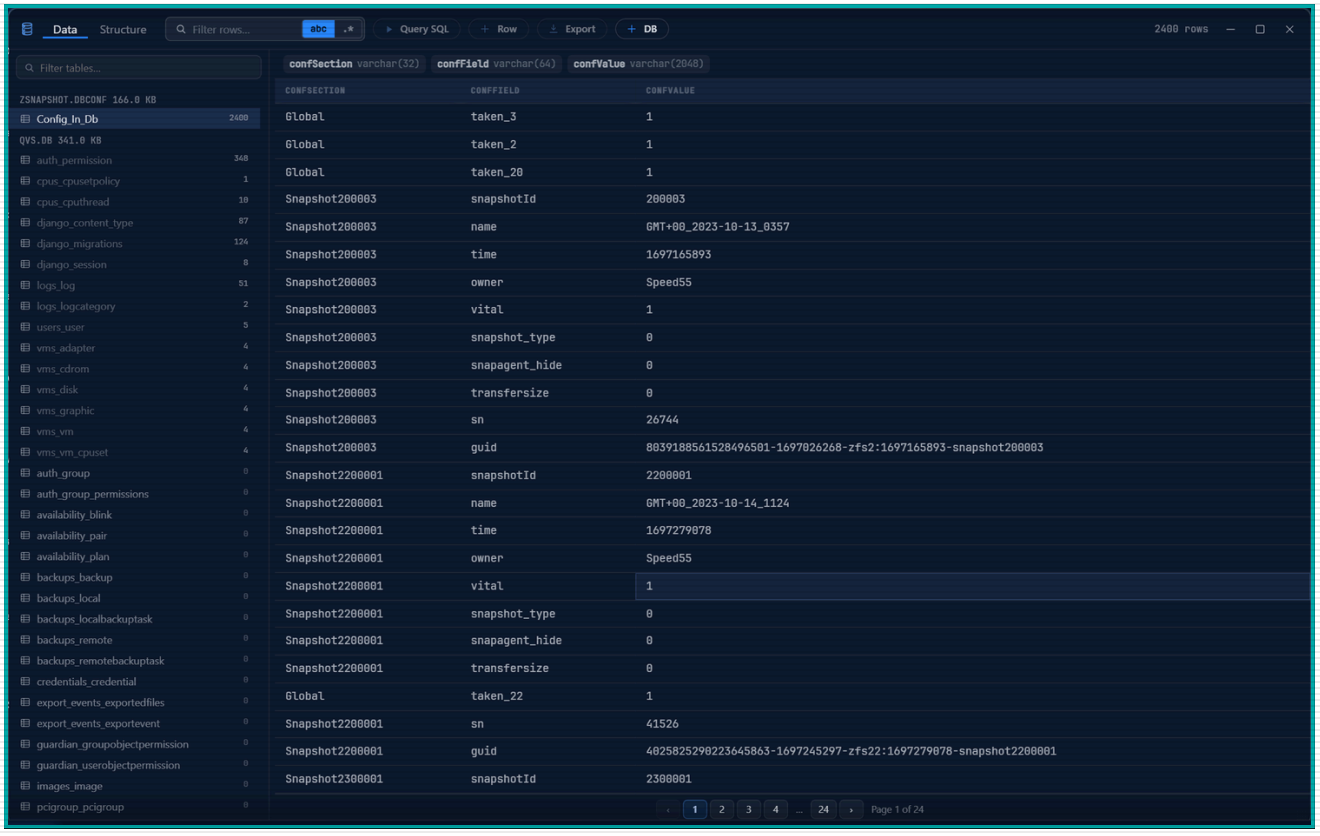
SQL BROWSER

Many QNAP features store their state in [illegible] that ship inside a diagnostic bundle. The [illegible] lets you open and inspect those databases directly — and, carefully, edit them.

> **How to open it:** Start menu / desktop → **SQL Browser**. Or from here: @@OPEN:SqlBrowser|Open the SQL Browser@@

What you can do

- [illegible] — pick a database and table; see the rows, the column names and their types.
- [illegible] — filter rows as you type (toggle [illegible] for PCRE), across a large table (paged).
- [illegible] — run your own `SELECT` against the database when a filter isn't enough.
- [illegible] — save a table/result out.



The SQL Browser — databases and their tables with row counts (left), the selected table's rows with typed column headers (right), and the [illegible] · [illegible] · filter/regex toolbar on top. Here the bundle's `ZSNAPSHOT.DBCONF` snapshot-config database is open.

When it's useful

The SQL Browser is your way into data that isn't surfaced by a dedicated viewer, for example:

- QNAP configuration or databases (e.g. `ZSNAPSHOT.DBCONF`),
- (who was given a public link to what),
- QNAP logs that are stored rather than plain text (some event/connection logs).

Editing — carefully

You can a cell, but the tool makes you confirm first (" ") because:

- edits are written to the – to your NAS's live database;
- (press while editing a cell to discard that one change).

> **Important.** Editing is a diagnostic/what-if tool on a throwaway session copy — use it to test a value or repair a corrupt row for analysis, not as a way to change the real NAS. When the session expires, your edits go with it.

> **Best practice.** Reach for the SQL Browser when a dedicated viewer doesn't show what you need — it's the "raw data" escape hatch over everything the bundle captured in SQLite form.

Watch it

> In-app demo: SQL Browser — opening a SQLite database shipped inside the bundle, read-only.

HBS 3 — BACKUP JOBS & LOGS

When you upload an (Hybrid Backup Sync) diagnostic bundle, StorageDiagLV opens two dedicated views: the configuration and the HBS viewer. This is how you diagnose why a backup is failing.

> **How to open it:** it opens automatically for an HBS bundle, or from the **HBS Jobs / App Logs** icon. Or from here: @@OPEN:Hbs|Open HBS 3 Jobs@@

HBS 3 Jobs

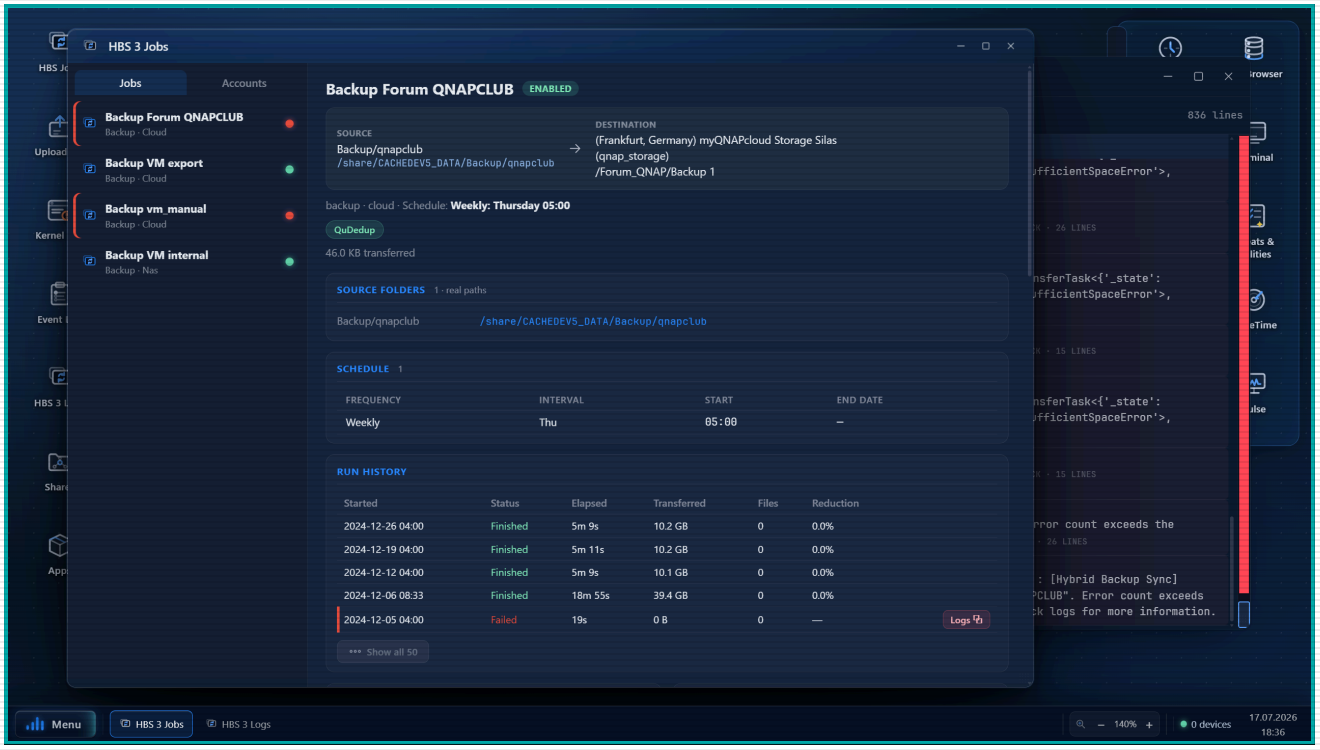


Figure — HBS 3 Jobs: per-job status (green = last run OK, red = failing), source → destination, schedule, and run history.

Each job card shows:

- , , and a — = healthy last run, = failing.
- — Backup / Sync, and target class (or).
- — the real source path and the remote target (e.g. a myQNAPcloud storage region, or another NAS).
- — frequency / interval / start / end.
- and transferred size.
- — per run: started, (Finished / failed), elapsed, transferred, files, and dedup .

a job is your target — open its to see when it started failing, and whether recent runs transferred (nothing backed up). A destination in another region/NAS that's unreachable, or a schedule that never fires, are the usual culprits.

HBS 3 Logs

The log viewer lists every log file in the HBS bundle with an (e.g. `server-error.log 999+`), so you can jump straight to the noisy one. Lines carry a , a (the HBS component + file:line that logged it), and the message — including the structured JSON status entries HBS writes for each transfer (`status`, `substatus`, `files_created`, `bytes_transferred`, …). Search and a filter narrow it down.

> **Best practice.** Start at the job's red status → its run history → then `server-error.log` in the log viewer for the actual error text. The JSON `status` / `substatus` fields in `history.log` tell you exactly which phase (prepare, md5, transfer) the job was in when it stalled.

Watch it

> In-app demo: An HBS 3 diagnosis bundle uploaded, then a backup job opened to its source, schedule and run history.

> In-app demo: The job list, then the exception that actually failed the job — InsufficientSpaceError.

REGEX COOKBOOK (REFERENCE)

> A browsable copy of the in-app **Regex Helper** (open it from the **?** next to the pattern field in the Pattern Editor — see 0016. Pattern Editor and Regex Helper). 56 curated PCRE examples aimed at Linux/QNAP logs; every pattern is matched case-insensitively, exactly as the app applies it.

PCRE Basics

Match literal text · beginner

The simplest regex is just the exact characters you want to find. Use this when you only need to check whether a fixed phrase appears somewhere in a log line.

```
mounted filesystem
```

Matches: `[ 12.345678] EXT4-fs (sda1): mounted filesystem with ordered data mode.`

Symbol	Meaning
`mounted`	the exact word 'mounted'
`(space)`	a single literal space character
`filesystem`	the exact word 'filesystem'

> **Note:** Each space in the pattern must match exactly one space in the line. If the log used a tab or two spaces between the words, this pattern would NOT match. (It is case-insensitive here, so it also matches 'Mounted Filesystem'.)

The dot . matches any one character · beginner

A dot stands for any single character (except a newline). Use it when one position in the text can vary, like the number after an interface name.

```
eth.
```

Matches: `[ 156.789012] e1000e 0000:00:19.0 eth0: NIC Link is Up 1000 Mbps Full Duplex`

Symbol	Meaning
`eth`	the literal letters 'eth'

Symbol	Meaning
`.`	any single character (here it matches the '0')

> **Note:** The dot needs exactly ONE character to be present. 'eth.' matches 'eth0', 'eth1', even 'eth ' — but it will NOT match a bare 'eth' at the very end of a line with nothing after it, and it never crosses a newline.

Quantifiers: + one-or-more, * zero-or-more, ? optional · beginner

Quantifiers say how many times the thing before them may repeat: + means at least one, * means any number including none, ? means zero or one (optional). Use them when lengths and optional pieces vary.

```
sda\d+.*re-?mounted
```

```
[ 91.234567] EXT4-fs (sda1): re-mounted. Opts: (null)
```

Symbol	Meaning
`sda`	the literal letters 'sda'
`\d+`	one or more digits (+ needs at least one) — matches '1'
`.*`	any characters, zero or more of them (* allows none)
`re`	the literal letters 're'
`-?`	an optional hyphen: zero or one '-' (? makes it optional)
`mounted`	the literal word 'mounted'

> **Note:** .* is greedy: it first grabs as much as possible, then backs off just enough to let 're-?mounted' match. On a line with two 're...mounted' phrases it would prefer the LAST one. Also note + fails on zero occurrences where * would succeed.

Character set [] matches one from a list · beginner

Square brackets match exactly one character out of the choices inside them. A dash makes a range, like [a-z] for any lowercase letter. Great for 'any disk letter' or 'any digit'.

```
sd[a-z]
```

```
[ 8.123456] sd 3:0:0:0: [sdc] Attached SCSI disk
```

Symbol	Meaning
`sd`	the literal letters 'sd'
`[a-z]`	exactly one letter from a through z (here it matches 'c')

> **Note:** [a-z] consumes exactly ONE character and only a letter — it will not match the digit in 'sd0' (use [0-9] for that). Because the app matches case-insensitively, [a-z] here also covers A–Z.

Grouping () makes a quantifier cover several characters · beginner

Parentheses bundle several characters into one unit so a quantifier applies to the whole group. Here () plus ? makes an entire word optional, so the pattern matches with or without it.

```
(high-speed )?USB device
```

Matches: `[ 3.214567] usb 1-1: new high-speed USB device number 2 using xhci_hcd`

Symbol	Meaning
`(`	start of a group
`high-speed`	the literal text 'high-speed ' (with trailing space)
`)`	end of the group
`?`	makes the whole preceding group optional (zero or one time)
`USB device`	the literal text 'USB device'

> **Note:** Without the parentheses, 'high-speed ?USB device' would attach the ? only to the single space before it, not to the whole word 'high-speed'. The group is what lets one quantifier control multiple characters.

Alternation | means this OR that · beginner

The pipe lets you accept one option OR another. Use it to match any of several interface names, error words, or states in a single pattern.

```
eth0|wlan0
```

Matches: `[ 3.456789] r8169 0000:03:00.0 eth0: Link is Up - 1000Mbps/Full duplex`

Symbol	Meaning
`eth0`	first alternative — the literal text 'eth0'
`|`	OR
`wlan0`	second alternative — the literal text 'wlan0'

> **Note:** The pipe has the widest reach, splitting everything on its left from everything on its right. 'eth0|1' means 'eth0' OR '1' — NOT 'eth0/eth1'. Bound it with a group like 'eth(0|1)' when you only mean the last character to vary.

Escape \ to match a metacharacter literally · beginner

A backslash strips the special meaning from characters like . ( ) [ ]. Use . to match a real dot, which is essential for version numbers and IP addresses.

```
5\.10\.\d+
```

`[ 0.000000] Linux version 5.10.60-qnap (root@buildhost) (gcc version 8.3.0) #1 SMP PREEMPT`

Symbol	Meaning
`5`	the literal digit 5
`\.`	an escaped dot — matches a real '.' character, not 'any character'
`10`	the literal digits '10'
`\.`	another literal dot
`\d+`	one or more digits (matches '60')

> **Note:** Forget the backslash and '.' becomes 'any character': '5.10' would also match '5x10' or '5-10'. Escape the special characters . ( ) [ ] { } + * ? ^ $ | \ whenever you need their literal form.

Character Classes

\d — match a run of digits (a numeric field) · beginner

\d matches a single digit 0–9. Add + to grab a whole number, like a device count, PID or sector number in a kernel line.

```
\d+ out of \d+ devices
```

```
md/raid:md1: raid level 5 active with 4 out of 4 devices, algorithm 2
```

Symbol	Meaning
`\d`	a single digit, 0 through 9
`+`	one or more of the previous item, so \d+ is a whole number
`out of`	literal text, spaces included, matched exactly
`devices`	literal word that anchors the match to the RAID summary

> **Note:** \d only matches ASCII 0–9; it will NOT match a decimal point, so on '2.00 TB' \d+ stops at the dot and captures only '2'. Use \d+.\d+ for decimals.

\D — match a non-digit (a separator between numbers) · beginner

\D matches any single character that is NOT a digit — a letter, space, colon or punctuation. Handy for stepping over the separators in an address like the SCSI H:C:T:L id.

```
sd \d\D\d\D\d\D\d
```

```
sd 1:0:0:0: [sda] Attached SCSI disk
```

Symbol	Meaning
`sd`	literal text 'sd' and a space
`\d`	a single digit 0–9
`\D`	a single NON-digit; here it lands on each ':' separator

> **Note:** \D is the exact opposite of \d, so it also matches spaces and letters — it is not 'the colon'. If a field could be multi-digit (e.g. sd 10:0:0:0) this fixed \d\D\d... shape breaks; use \d+ pieces joined by ':' instead.

\s and \S — a space, then a whitespace-delimited token · beginner

\s matches one whitespace character (space or tab); \S matches one non-whitespace character. \S+ grabs a whole token up to the next space — great for pulling one field out of a log line.

```
Opts:\s\S+
```

```
EXT4-fs (dm-1): re-mounted. Opts: data=ordered
```

Symbol	Meaning
`Opts:`	literal label to anchor the match on
`\s`	exactly one whitespace character (space, tab, etc.)
`\S`	one NON-whitespace character
`+`	one or more, so \S+ is the whole token until the next space

> **Note:** \S+ is greedy but stops at the first whitespace, so it grabs 'data=ordered' but NOT a value that itself contains a space. Remember \s also matches tabs and newlines, not only the space bar.

\w — word characters (a device name like sdb) · beginner

\w matches one 'word' character: a letter, digit or underscore ([A-Za-z0-9_]). \w+ captures an identifier such as a block-device name inside brackets.

```
\[\w+\]
```

Matches: `sd 2:0:0:0: [sdb] Attached SCSI disk`

Symbol	Meaning
`\[`	a literal '[' (escaped, because [is special in regex)
`\w`	one word character: A-Z, a-z, 0-9 or underscore
`+`	one or more, so \w+ is the whole name 'sdb'
`\]`	a literal ']'

> **Note:** \w does NOT include the hyphen, so on a name like 'dm-3' the pattern \w+ matches only 'dm' and stops at the '-'. Use [\w-]+ if you need to include hyphens.

\W — a non-word character (the '-' in dm-4) · beginner

\W matches any single character that is NOT a word character — punctuation, spaces, hyphens. Useful to match the separator between a name and a number, e.g. the '-' in a device-mapper name.

```
dm\W\d+
```

Matches: `device-mapper: thin: Data device (dm-4) discard unsupported`

Symbol	Meaning
`dm`	literal text 'dm'
`\W`	one NON-word character; here it matches the '-'
`\d`	a single digit
`+`	one or more digits

> **Note:** \W is the opposite of \w, so it also matches spaces and brackets — not just '-'. It will NOT match an underscore, because '_' counts as a word character (part of \w).

:xdigit: — a POSIX class for hex digits · intermediate

A POSIX class names a set of characters; :xdigit: means one hexadecimal digit (0–9, a–f, A–F). Use it to match a hex address or register value after '0x'.

```
0x:xdigit:+
```

Matches: `pci 0000:00:1f.2: reg 0x24: [mem 0xf7d00000-0xf7d007ff]`

Symbol	Meaning
`0x`	literal '0x' prefix that marks a hex number
`:xdigit:`	one hex digit: 0–9, a–f or A–F (a POSIX class)
`+`	one or more, so it grabs the whole hex value

> **Note:** The double brackets are mandatory: the outer [...] is the character class and [:xdigit:] is the POSIX name inside it. Writing [:xdigit:] (single brackets) instead just matches the literal letters :,x,d,i,g,t. Also note JavaScript's regex engine does NOT support POSIX classes — use [0-9A-Fa-f] there.

Quantifiers

+ — one or more (disk block count) · beginner

Uses + to mean 'one or more of the preceding item'. Great for grabbing a run of digits (like a sector count) or a run of letters (like a device name) whose length you don't know in advance.

```
\[sd[a-z]+\] \d+ 512-byte
```

Matches: `sd 6:0:0:0: [sdf] 3907029168 512-byte logical blocks: (2.00 TB/1.82 TiB)`

Symbol	Meaning
`\[`	a literal '[' (the bracket is escaped so it isn't a character class)
`sd`	the literal text 'sd'
`[a-z]+`	one or more lowercase letters (matches the 'f' in sdf, or 'aa' in sdaa)
`\]`	a literal ']'
` `	a literal space
`\d+`	one or more digits (the whole block count 3907029168)
`512-byte`	the literal text ' 512-byte'

> **Note:** + demands AT LEAST ONE character. If the digits were missing (e.g. '[sdf] 512-byte') \d+ fails to match, whereas \d* would still match zero digits. Use + when the field must be present, * when it may be empty.

* — zero or more (kernel timestamp padding) · beginner

Uses * to mean 'zero or more of the preceding item'. Perfect for optional padding — the kernel right-aligns its timestamp with a variable number of leading spaces, so you must allow any count, including none.

```
^\[ *\d+\.\d+\] usb
```

```
[ 14.523187] usb 1-1: new high-speed USB device number 3 using xhci_hcd
```

Symbol	Meaning
`^`	anchor: start of the line
`\[`	a literal '['
`*`	zero or more spaces (soaks up the ' ' padding — or none at all)
`\d+`	one or more digits (the seconds part, 14)
`\.`	a literal dot (escaped, so it is NOT 'any character')
`\d+`	one or more digits (the microseconds, 523187)
`\]`	a literal ']'
`usb`	the literal text ' usb'

> **Note:** * happily matches ZERO, so ' *' also matches a timestamp printed with no padding like '[14.523187]'. If you truly require at least one

> space, use ' +' (or ' {1,}') instead. Also note . is escaped — an unescaped '.' would match any character.

? — optional group (ATA port vs device) · intermediate

Uses ? to make the item just before it optional (0 or 1 time). Here it lets one pattern match both a whole-port error (ata2:) and a specific-device error (ata2.00:) on the SATA link.

```
ata\d+(\.\d+)?: exception
```

Matches: `ata2.00: exception Emask 0x0 SAct 0x0 SErr 0x0 action 0x6 frozen`

Symbol	Meaning
`ata`	the literal text 'ata'
`\d+`	one or more digits (the port number, 2)
`(`	start a group so ? can apply to the whole thing
`\.\d+`	a literal dot then one or more digits (the '.00' device suffix)
`)`	end the group
`?`	makes the preceding group optional — 0 or 1 occurrence
`: exception`	the literal text ': exception'

> **Note:** ? binds only to the token immediately before it — that is why the '.00' is wrapped in (…). Because the group is optional, this pattern also matches port-level lines like 'ata2: exception'. If you only want per-device errors, drop the ? (and the parentheses).

{n,m} — a counted range (IPv4 octets) · intermediate

Uses {min,max} to require a repeat count between two bounds. \d{1,3} matches each IPv4 octet, which is 1 to 3 digits long. Related forms: {n} is exactly n, {n,} is n or more.

```
\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}
```

Matches: `2026-07-17,09:41:12,admin,10.100.200.10,---,[Users] User admin logged in from 10.100.200.10 via HTTPS`

Symbol	Meaning
`\d{1,3}`	between 1 and 3 digits (one IPv4 octet, e.g. 10 or 100)
`\.`	a literal dot separating the octets (escaped)

Symbol	Meaning
{1,3}	the range form: at least 1, at most 3 repeats
{n}	(sibling form) EXACTLY n repeats — e.g. \d{4} for a 4-digit year
{n,}	(sibling form) n OR MORE repeats — e.g. \d{2,} for two-plus digits

> **Note:** Braces count characters, not numeric value: \d{1,3} will happily match the impossible octet '999' (as in 10.100.200.999). Regex checks LENGTH only, so validate the value separately if it matters. Also, a stray space — '{1, 3}' — breaks the quantifier in most engines; write it with no space.

.? vs . — lazy stops at the first delimiter · advanced

Shows why lazy matching matters. On a line with several bracketed tags, greedy .* races to the LAST ']', swallowing everything between. Adding ? makes it lazy so it stops at the FIRST ']' — capturing just the app tag.

```
\[.*?\]
```

Matches a line like `[Storage & Snapshots] [Pool1] Now starting to rebuild RAID group 1.`

Symbol	Meaning
\[	a literal '['
.	any single character
*	zero or more of the preceding item (of '.')
?	placed AFTER * it makes the match lazy — take as few characters as possible
\]	a literal ']' — the first one the lazy match can reach

> **Note:** The greedy version [.] on this same line matches '[Storage & Snapshots] [Pool1]' — from the first '[' to the LAST ']', eating the '] [' in the middle. Lazy [.?] stops at the first ']' and yields just '[Storage & Snapshots]'. Whenever a line has repeated delimiters, prefer lazy (or a negated class like ^*]).

Anchors

Line must START WITH "kernel:" · beginner

The caret ^ anchors the match to the very start of the line, so the text right after it must be the first thing on the line. Classic use: find lines that begin with a literal tag like "kernel:".

```
^kernel:
```

Matches: `kernel: [12345.678901] EXT4-fs (dm-0): mounted filesystem with ordered data mode`

Symbol	Meaning
`^`	anchor to the start of the line (a position, it matches no character)
`kernel`	the literal word "kernel"
`:`	a literal colon character

> **Note:** It will NOT match a normal syslog line like "Jul 17 10:23:45 NAS kernel: ..." — the date, time and hostname come first, so "kernel:" is no longer at the start. If your log adds a timestamp prefix, drop the ^ or match the prefix too.

Line must END WITH "Link is Down" · beginner

The dollar sign $ anchors the match to the end of the line, so nothing may follow the matched text. Use it to catch lines that finish with a specific status such as an interface going down.

```
Link is Down$
```

Matches: `<4>[ 842.115200] r8169 0000:03:00.0 eth0: Link is Down`

Symbol	Meaning
`Link is Down`	the literal text "Link is Down" (the spaces are matched literally too)
`$`	anchor to the end of the line (a position, it matches no character)

> **Note:** A single trailing space or a Windows carriage return (\r) after "Down" breaks it: "Link is Down " will NOT match, because $ needs "Down" to be the last characters. Trim trailing whitespace, or write "Down\s*$".

Whole word "sda" only (\b word boundary) · beginner

\b marks a word boundary — the edge between a word character (letters, digits, underscore) and anything else. Wrapping a token in \b...\b matches it only as a standalone word, not as part of a longer one.

```
\bsda\b
```

```
[ 15.234567] sd 0:0:0:0: [sda] Attached SCSI disk
```

Symbol	Meaning
`\b`	a word boundary: the edge where a word char meets a non-word char (here between "[" and "s")
`sda`	the literal disk name "sda"
`\b`	the closing word boundary (here between "a" and "]"), forcing "sda" to be a whole word

> **Note:** It deliberately will NOT match the partition "sda1" or a typo "sdaa" — a digit or letter right after "sda" is still a word character, so the closing \b fails. Use "\bsda\d*\b" if you also want the partitions.

Embedded "hci" only (\B non-boundary) · intermediate

\B is the exact opposite of \b: it matches a position that is NOT a word boundary, i.e. sitting inside a run of word characters. Use it to find a fragment only when it is glued into a larger identifier.

```
\Bhci\B
```

```
[ 2.345678] xhci_hcd 0000:00:14.0: xHCI Host Controller
```

Symbol	Meaning
`\B`	a non-boundary: the previous char must be a word char (here the "x" before "hci")
`hci`	the literal fragment "hci"
`\B`	another non-boundary: the next char must be a word char (here the underscore "_" in "xhci_hcd")

> **Note:** Underscore counts as a word character (\w), which is why the "_" after "hci" keeps \B true. It will NOT match "hci" standing alone or spaced out like " hci " — the surrounding spaces create real word boundaries, so \B fails.

Kernel log-level tag at line start · intermediate

Combine the start anchor ^ with a structured token to require something in an exact position. Here it matches the kernel priority tag like <6> only when it is the first thing on the line.

```
^<\d>
```

`<6>[ 842.115200] usb 1-1: new high-speed USB device number 4 using ehci-pci`

Symbol	Meaning
`^`	anchor to the start of the line
`<`	a literal less-than sign
`\d`	exactly one digit, 0–9 (kernel levels run 0–7)
`>`	a literal greater-than sign

> **Note:** \d here is a single digit, so it will NOT match a two-digit facility+level tag like "<30>" (after "3" it expects ">" but sees "0"). It also fails if anything — even one leading space — sits before the "<". Use "^<\d+>" to allow multi-digit tags.

Groups

Capturing group: pull the ATA port (slot) number · beginner

Wrap the part you want to keep in parentheses to pull a single sub-value out of a line. Here it grabs the ATA port (physical link/slot) number from an ata identify message so you know which disk the line is about.

```
ata(\d+)\.\d+
```

`ata3.00: ATA-9: WDC WD40EFRX-68WT0N0, 82.00A82, max UDMA/133`

Symbol	Meaning
`ata`	literal text 'ata' (matched case-insensitively)
`(\d+)`	capturing group 1: one or more digits — the port/slot number you extract
`\.`	an escaped, literal dot
`\d+`	one or more digits — the device sub-number (e.g. 00)

> **Note:** The dot is escaped as . on purpose — a bare . matches ANY character, so 'ata3X00' would wrongly match. The value you want lives in group 1, not group 0 (the whole match is 'ata3.00').

Non-capturing group: match a repeated block without numbering it · intermediate

Use (?:...) to group and repeat part of a pattern when you do NOT want a numbered capture cluttering your results. The classic case: match the four dotted octets of an IPv4 address in a QNAP connection log.

```
(?:\d{1,3}\.){3}\d{1,3}
```

```
[Authentication] User "admin" logged in successfully from 10.0.1.55 via HTTP.
```

Symbol	Meaning
`(?:`	start a non-capturing group — groups tokens but creates no numbered capture
`\d{1,3}`	1 to 3 digits — one IP octet
`\.`	a literal dot separating octets
`)`	end the group
`{3}`	repeat that whole group exactly 3 times (the first three octets)
`\d{1,3}`	the final octet

> **Note:** \d{1,3} checks shape, not range, so it also accepts impossible octets like 999. Swap (?:...) for a capturing (...) only if you actually need to pull an octet out — otherwise (?:...) keeps your capture list clean.

Named group: label the device and sector of an I/O error · intermediate

Give a capture a readable name with (?...) so you can pull it out by name instead of counting parentheses. Here it labels the failing block device and the sector from a kernel block-layer I/O error.

```
dev (?<dev>\w+), sector (?<sector>\d+)
```

```
blk_update_request: I/O error, dev sda, sector 1234567 op 0x0:(READ) flags 0x0 phys_seg 8 prio class 0
```

Symbol	Meaning
`dev`	literal text 'dev ' that precedes the device name
`(?<dev>\w+)`	named group 'dev': one or more word characters — the block device (sda)
`, sector`	literal separator text between the two values
`(?<sector>\d+)`	named group 'sector': one or more digits — the failing sector

> **Note:** Read the values back by name (PHP: $m['sector']; JS: m.groups.sector). It will NOT match older kernels that log 'sector

12345' without the 'dev sdX,' phrasing on the same line.

Lookahead: keep a number only when a unit follows it · advanced

Positive lookahead (?=...) matches a value only when it is followed by something specific — without eating that something. Here it grabs the available-memory kilobyte count only when a 'K/' unit follows it.

```
\d+(?=K/)
```

Sample input: `Memory: 16384000K/16777216K available (14336K kernel code, 2049K rwdata, 4096K rodata)`

Symbol	Meaning
`\d+`	one or more digits — the value you actually keep
`(?=K/)`	positive lookahead: only match if 'K/' comes immediately next, but do NOT consume it

Note: The lookahead is zero-width — 'K/' is not part of the match, so you get just the digits. It will NOT match '16384000 K/' (a space before K) because the lookahead demands 'K' right after the last digit.

Negative lookahead: match every DID_ result except DID_OK · advanced

Negative lookahead (?!...) matches only when the following text is NOT what you list. Here it flags any SCSI DID_ host-status that is not DID_OK — i.e. it catches only the failures and skips the healthy result.

```
DID_(?!OK)\w+
```

Sample input: `sd 4:0:0:0: [sdc] tag#0 FAILED Result: hostbyte=DID_BAD_TARGET driverbyte=DRIVER_OK cmd_age=0s`

Symbol	Meaning
`DID_`	literal prefix 'DID_' (SCSI host-status)
`(?!OK)`	negative lookahead: fail the match if the next characters are 'OK'
`\w+`	one or more word characters — the actual error name (BAD_TARGET, NO_CONNECT, ...)

Note: Because this app matches case-insensitively, (?!OK) also rejects lowercase 'did_ok'. That is the point: it deliberately will NOT match the healthy 'DID_OK', so only real failures survive.

Lookbehind: grab the PID after 'process ' in an OOM kill · advanced

Lookbehind (?<=...) matches a value only when a fixed string comes right before it, without pulling that string into the match. Here it extracts just the PID that follows 'process ' in the kernel OOM-killer line.

```
(?<=process )\d+
```

Sample input: `Out of memory: Killed process 12345 (mysqld) total-vm:8402168kB, anon-rss:8200000kB, file-rss:0kB`

Symbol	Meaning
`(?<=process )`	positive lookbehind: require the text 'process ' immediately before, but don't include it in the match
`\d+`	one or more digits — the process ID

> **Note:** PCRE lookbehind must be FIXED length: (?<=process\s+) with \s+ throws 'lookbehind assertion is not fixed length'. Keep it a literal like 'process ', or capture with a normal group instead.

Linux Examples

Kernel uptime timestamp [1234.567890] · beginner

Matches the leading `[seconds.microseconds]` timestamp that the kernel stamps on every dmesg line. Use it to detect timestamped kernel logs or to strip the prefix before parsing the message.

```
^\[\s*\d+\.\d{6}\]
```

Sample input: `[ 1234.567890] EXT4-fs (dm-3): mounted filesystem with ordered data mode.`

Symbol	Meaning
`^`	anchor: the match must start at the beginning of the line
`\[`	a literal [(escaped, because a bare [starts a character class)
`\s*`	zero or more spaces (the kernel right-aligns the seconds field)
`\d+`	one or more digits = the whole-seconds part
`\.`	a literal dot (escaped; a bare . means 'any character')
`\d{6}`	exactly six digits = the microseconds part
`\]`	a literal]

> **Note:** Won't match logs captured without timestamps, nor `dmesg -T` output that shows human dates like `[Fri Jul 17 09:12:33 2026]`; also `\d{6}` assumes exactly six fractional digits.

Network interface name (eth0 / bond1 / tun0) · beginner

Matches a classic Linux network interface name by trying a list of common prefixes followed by an index number. Great first lesson in alternation (the | 'or' operator).

```
\b(?:eth|bond|tun|wlan|br)\d+\b
```

Sample text to test against: `[ 45.678901] bond0: Enslaving eth1 as a backup interface`

Symbol	Meaning
`\b`	word boundary so we don't match these letters glued inside a bigger word
`(?:`	start a non-capturing group (groups choices without saving them)
`eth|bond|tun|wlan|br`	the alternation: match ANY ONE of these interface prefixes
`)`	end the group
`\d+`	one or more digits = the interface index (the 0 in bond0)
`\b`	word boundary at the end

> **Note:** Won't match systemd 'predictable' names such as `enp3s0`, `eno1`, or `ens160` — they don't start with these prefixes; add an `en\w+` alternative to cover them.

SATA/SCSI disk name (sda, sdb, sdaa) · beginner

Matches a SATA/SCSI whole-disk name: the letters 'sd' followed by one or more drive letters. This is the name in `[sdb]` messages, not a partition.

```
\bsd[a-z]+\b
```

Sample text to test against: `[ 3.872011] sd 4:0:0:0: [sdb] 3907029168 512-byte logical blocks: (2.00 TB/1.82 TiB)`

Symbol	Meaning
`\b`	word boundary (start of the device token)
`sd`	the literal letters 'sd' (SCSI/SATA disk prefix)
`[a-z]+`	one or more letters = the drive suffix (a, b, ... z, then aa)

Symbol	Meaning
`\b`	word boundary at the end

> **Note:** The `+` on `[a-z]+` matters for the 27th+ disk (`sdaa`); because it stops at digits, on `sdb1` it captures only `sdb` . It will NOT match NVMe (`nvme0n1`) or software-RAID (`md0`).

NVMe device / partition (nvme0n1, nvme0n1p2) · intermediate

Matches an NVMe namespace name and its optional partition suffix. Shows how to make a whole chunk optional with (?:...)?

```
\bnvme\d+n\d+(?:p\d+)?\b
```

Sample text: `[ 5.129384] EXT4-fs (nvme0n1p2): mounted filesystem with ordered data mode`

Symbol	Meaning
`\b`	word boundary (start of the device token)
`nvme`	the literal letters 'nvme'
`\d+`	the controller number (the 0 in nvme0)
`n`	literal 'n' = the namespace marker
`\d+`	the namespace number (the 1 in n1)
`(?:p\d+)?`	OPTIONAL partition suffix: 'p' + digits; the trailing ? makes the whole group optional
`\b`	word boundary at the end

> **Note:** NVMe partitions use a `p` separator, unlike SATA — so this will NOT match `sdb1` or eMMC `mmcblk0p1` ; the mandatory `nvme…n…` skeleton is NVMe-specific.

RAID array / device-mapper node (md1, dm-3) · intermediate

Matches either a Linux software-RAID array (`mdN`) or a device-mapper node (`dm-N` , used by LVM/thin/crypt). QNAP storage stacks show both constantly.

```
\b(?:md\d+|dm-\d+)\b
```

Sample text: `[ 12.345678] md/raid1:md1: active with 2 out of 2 mirrors [2/2] [UU]`

Symbol	Meaning
`\b`	word boundary
`(?:`	start a non-capturing group for the two alternatives
`md\d+`	'md' + digits = a software-RAID array (md0, md1, ...)
`|`	OR
`dm-\d+`	'dm-' + digits = a device-mapper node (LVM / thin / crypt)
`)`	end the group
`\b`	word boundary at the end

> **Note:** The `dm-3` alternative needs the literal hyphen; a hyphen-less `dm3` will NOT match, and neither will the friendly LVM path form like `vg1-lv_data`.

PCI address 0000:03:00.0 (domain:bus:device.function) · intermediate

Matches a full PCI address in domain:bus:device.function form, all in hexadecimal. Kernel driver-bind lines lead with this.

```
\b[0-9a-f]{4}:[0-9a-f]{2}:[0-9a-f]{2}\.[0-7]\b
```

Sample input: `[ 1.234567] pci 0000:03:00.0: [1b21:1166] type 00 class 0x010601`

Symbol	Meaning
`\b`	word boundary
`[0-9a-f]{4}`	four hex digits = the PCI domain (usually 0000)
`:`	a literal colon separator
`[0-9a-f]{2}`	two hex digits = the bus
`:`	a literal colon
`[0-9a-f]{2}`	two hex digits = the device (slot)
`\.`	a literal dot
`[0-7]`	one digit 0-7 = the function number
`\b`	word boundary at the end

> **Note:** This requires the 4-digit domain, so it will NOT match the short `03:00.0` form (domain omitted) that some tools print; write the domain as optional if you need both.

MAC address (24:5e:be:12:34:ab) · intermediate

Matches a colon-separated MAC address: six pairs of hex digits. Teaches the {N} repetition count on a group.

```
\b(?:[0-9a-f]{2}:){5}[0-9a-f]{2}\b
```

Sample to match: `[ 3.456789] r8169 0000:03:00.0 eth0: RTL8168h/8111h, 24:5e:be:12:34:ab, XID 541, IRQ 129`

Symbol	Meaning
`\b`	word boundary
`(?:[0-9a-f]{2}:)`	one group = two hex digits followed by a colon (an octet plus its separator)
`{5}`	repeat that group exactly five times (the first five octets)
`[0-9a-f]{2}`	the final two hex digits (sixth octet, with no trailing colon)
`\b`	word boundary at the end

> **Note:** Shape-only match: it will NOT match a MAC printed with dashes (`24-5e-be-12-34-ab`) or Cisco dotted form (`245e.be12.34ab`); it also can't tell a real MAC from any same-shaped hex string.

IPv4 address (192.168.1.100) · beginner

Matches a dotted-quad IPv4 address by shape: four numbers of one to three digits joined by dots. Shows repeating a group exactly three times.

```
\b(?:\d{1,3}\.){3}\d{1,3}\b
```

Sample to match: `[ 88.123456] nfs: server 192.168.1.100 not responding, still trying`

Symbol	Meaning
`\b`	word boundary
`(?:\d{1,3}\.)`	one group = one-to-three digits then a dot (an octet plus separator)
`{3}`	repeat that group exactly three times (the first three octets)
`\d{1,3}`	the final octet, one to three digits (no trailing dot)
`\b`	word boundary at the end

> **Note:** This is a shape check, not a range check: it happily matches impossible octets like `999.999.1.1`, and it does NOT match IPv6; to

enforce 0–255 you need a much longer per-octet alternation.

UUID (filesystem / partition 8-4-4-4-12) · advanced

Matches a canonical UUID: five hex groups of length 8-4-4-4-12 joined by hyphens. Seen on mount, blkid, and RAID-member lines.

```
\b[0-9a-f]{8}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{12}\b
```

Matches: `[ 15.678901] XFS (dm-2): Mounting V5 Filesystem 4f8b2a1c-3d9e-4f21-b8c7-1a2b3c4d5e6f`

Symbol	Meaning
`\b`	word boundary
`[0-9a-f]{8}`	eight hex digits (the first UUID field)
`-`	a literal hyphen separator
`[0-9a-f]{4}`	four hex digits (this group appears three times)
`[0-9a-f]{12}`	twelve hex digits (the final node field)
`\b`	word boundary at the end

Note: Matches only the standard hyphenated 8-4-4-4-12 form; it will NOT match a hyphen-less 32-hex UUID or one wrapped in braces like `{...}`.

QNAP Examples

mdraid disk failure (degraded array, md127) · intermediate

Catches the kernel line QNAP logs when a member drive drops out of a software-RAID pool and the array goes degraded. Use it to find which disk (e.g. sdd3) fell out of which array (e.g. md127).

```
md/raid\d+:md\d+: Disk failure on sd[a-z]+\d+
```

Matches: `md/raid5:md127: Disk failure on sdd3, disabling device.`

Symbol	Meaning
`md/raid`	literal 'md/raid' tag the kernel prints for a RAID array
`\d+`	one or more digits — the RAID level (5 here)
`:md`	literal ':md' beginning the array's device name
`\d+`	one or more digits — the array number (127 here)
`: Disk failure on`	the literal failure phrase

Symbol	Meaning
`sd`	literal 'sd' — the SATA/SCSI disk prefix
`[a-z]+`	one or more lowercase letters — the drive letter (dd)
`\d+`	one or more digits — the partition number (3)

> **Note:** Requires a trailing partition number: it will NOT match a whole-disk failure logged as 'sdd' with no digit, because \d+ needs at least one digit after the letters.

ZFS pool suspended by I/O failure (QuTS hero) · beginner

On QuTS hero (ZFS) NAS units, this is the kernel warning printed when a pool takes an unrecoverable I/O error and ZFS freezes it. Use it to spot a hard pool stall / suspension.

```
Pool '[^']+' has encountered an uncorrectable I/O failure
```

Sample line: `WARNING: Pool 'zpool1-1' has encountered an uncorrectable I/O failure and has been suspended.`

Symbol	Meaning
`Pool '`	literal text plus the opening single-quote before the pool name
`[^']+`	one or more characters that are NOT a single-quote — the pool name
`' has encountered an uncorrectable I/O failure`	the literal ZFS message ('/' here is just a normal character, not a delimiter)

> **Note:** This matches the exact prose. It will NOT catch the shorter 'pool I/O is currently suspended' status line, nor a checksum-only ('uncorrectable error') event — those use different wording.

DRBD connection-state change (HA pairing) · intermediate

In QNAP HA setups the two nodes replicate over DRBD; this catches a connection-state transition line so you can see when the link went from Connected to a failure state.

```
drbd \w+: conn\( \w+ -> \w+ \)
```

Sample line: `drbd r1: conn( Connected -> NetworkFailure )`

Symbol	Meaning
`drbd`	literal 'drbd ' prefix
`\w+`	one or more word characters – the resource name (r1)
`: conn`	literal ': conn'
`\(`	a literal '(' – escaped, because a bare (starts a group
`\w+`	the old connection state (Connected)
`->`	the literal arrow between the two states
`\w+`	the new connection state (NetworkFailure)
`\)`	a literal ')' – also escaped

> **Note:** \w+ matches a single unbroken word, so it will NOT match a volume-qualified resource logged as 'drbd r0/0 drbd1:' (the space and slash break \w+) – only the plain 'drbd :' form.

SSD cache created for a storage pool · beginner

A QNAP event-log line recorded when an SSD cache is created and bound to a pool. Use it to confirm the cache exists and at what RAID level it was built.

```
\[SSD Cache\] Created SSD cache with RAID \d+
```

Matches e.g. `[SSD Cache] Created SSD cache with RAID 1 for storage pool 1.`

Symbol	Meaning
`\[SSD Cache\]`	the literal '[SSD Cache]' tag – brackets escaped so they aren't read as a character class
`Created SSD cache with RAID`	the literal event text
`\d+`	one or more digits – the RAID level (1, or e.g. 10)

> **Note:** The word 'Created' is literal, so this will NOT match cache enable/remove/full events. Also, a level written without a space ('RAID1') would fail because the pattern expects 'RAID ' then digits.

PCIe AER corrected/uncorrected error · advanced

Catches PCI Express Advanced Error Reporting lines – often seen with NVMe drives, HBAs, or Thunderbolt/expansion cards on QNAP. Repeated AER events point at a flaky slot, riser, or card.

```
pcieport \d{4}:\d{2}:[0-9a-f]{2}\.\d: AER:
```

Matches: `pcieport 0000:00:1c.4: AER: Multiple Corrected error received: 0000:03:00.0`

Symbol	Meaning
`pcieport`	literal driver tag for a PCIe root/port
`\d{4}`	exactly 4 digits — the PCI domain (0000)
`:`	a literal colon
`\d{2}`	exactly 2 digits — the bus number (00)
`[0-9a-f]{2}`	exactly 2 hex digits — the device number (1c); the /i flag also allows uppercase like 1C
`\.`	a literal dot before the function number
`\d`	one digit — the PCI function (0–7)
`: AER:`	the literal ' AER:' error tag

> **Note:** The device field is hex ([0-9a-f]) but bus/domain here are \d only — a bus number containing a hex letter (rare, but possible) would NOT match. It also won't match errors reported under a different tag such as 'nvme' or 'igb'.

SATA link / drive ATA exception · intermediate

The classic libata error line printed when a SATA drive throws an exception (timeout, reset, CRC). Recurring ata exceptions on one port usually mean a bad disk, cable, or backplane slot.

```
ata\d+\.\d+: exception Emask 0x[0-9a-f]+
```

Matches: `ata1.00: exception Emask 0x0 SAct 0x400000 SErr 0x0 action 0x6 frozen`

Symbol	Meaning
`ata`	literal 'ata'
`\d+`	one or more digits — the ATA port number (1)
`\.`	a literal dot
`\d+`	one or more digits — the device number (00)
`: exception Emask 0x`	the literal exception text plus the '0x' hex prefix

Symbol	Meaning
`[0-9a-f]+`	one or more hex digits — the error-mask value; /i also accepts uppercase A-F

> **Note:** This requires the '.NN' device suffix, so it will NOT match port-level link lines like 'ata5: SATA link down (SStatus 0 SControl 300)', which have no dot-and-device part.

Thin pool out-of-data-space (dm-thin) · beginner

The critical device-mapper line printed when a thin (over-provisioned) pool runs out of physical data blocks and switches to error mode — writes will start failing. A top-priority alert on QNAP thin pools.

```
device-mapper: thin: \d+:\d+: switching pool to out-of-data-space
```

Example: `device-mapper: thin: 253:4: switching pool to out-of-data-space (error IO) mode`

Symbol	Meaning
`device-mapper: thin:`	the literal dm-thin subsystem tag
`\d+`	one or more digits — the block device major number (253)
`:`	a literal colon separating major and minor
`\d+`	one or more digits — the minor number (4)
`: switching pool to out-of-data-space`	the literal critical message

> **Note:** This is the hard 'full' event. It will NOT match the earlier 'reached low water mark for data device' warning, nor the recovery line 'switching pool to write mode' after space is added.

Kernel WARNING backtrace header (CPU/PID/at file:line) · advanced

The header line of a kernel WARN_ON()/backtrace, naming the CPU, process, and the source file:line that tripped. Useful for grouping call-trace incidents by the failing kernel location.

```
WARNING: CPU: \d+ PID: \d+ at \S+:\d+
```

Example: `WARNING: CPU: 2 PID: 3311 at kernel/workqueue.c:2698 process_one_work+0x4a8/0x510`

Symbol	Meaning
`WARNING: CPU:`	the literal warning header
`\d+`	one or more digits — the CPU number (2)
`PID:`	literal ' PID: '
`\d+`	one or more digits — the process id (3311)
`at`	literal ' at '
`\S+`	one or more non-space chars — the source path; it's greedy, so it grabs the whole 'path:line' then backtracks to hand ':line' back
`:`	a literal colon before the line number
`\d+`	one or more digits — the source line number (2698)

> **Note:** This matches the modern format only. Older kernels emit 'WARNING: at kernel/workqueue.c:2698 ...' with no 'CPU:'/'PID:' fields, and that form will NOT match.

Snapshot created for a shared folder / volume / LUN · intermediate

A QNAP event-log line recorded when a snapshot is taken. Use it to confirm scheduled snapshots are running and to see the snapshot name and its target.

```
Created snapshot "[^"]+" for (shared folder|volume|LUN) "[^"]+"
```

Example: `[Snapshot] Created snapshot "S1-20240115-1200" for shared folder "Public".`

Symbol	Meaning
`Created snapshot "`	the literal text and the opening double-quote before the snapshot name
`[^"]+`	one or more characters that are NOT a double-quote — the snapshot name
`" for`	literal closing quote plus ' for '
`(shared folder|volume|LUN)`	alternation — matches any one of these three target types
`"`	literal space and the opening quote before the target name
`[^"]+`	the target's name (Public)
`"`	the closing double-quote

> **Note:** It expects straight ASCII double-quotes. If a firmware locale logs the name in typographic 'curly' quotes or single quotes, both the literal " and the [^"] class will fail to match.

Ready to Use

IPv4 address (octet-validated) · intermediate

Matches a full IPv4 address like 10.0.0.5 and rejects impossible octets (no 999.999). Use it to pull client/server IPs out of NFS, DHCP, iSCSI and network kernel messages.

```
\b(?:(?:25[0-5]|2[0-4][0-9]|1[0-9][0-9]|[1-9]?[0-9])\.){3}(?:25[0-5]|2[0-4][0-9]|1[0-9][0-9]|[1-9]?[0-9])\b
```

Sample text

```
nfs: server 10.0.0.5 not responding, still trying
```

Symbol	Meaning
`\b`	word boundary, so the number is not glued to other digits/letters
`(?:...)`	a group that is used for structure but not captured
`25[0-5]`	the octet 250–255
`2[0-4][0-9]`	the octet 200–249
`1[0-9][0-9]`	the octet 100–199
`[1-9]?[0-9]`	the octet 0–99 (optional first digit)
`\.`	a literal dot separating octets
`{3}`	exactly three 'octet-dot' groups, then one final octet

> **Note:** It will NOT match IPv6, and because it stops at the address it ignores a trailing port: in 10.0.0.5:22 it captures only 10.0.0.5. Word boundaries also let it grab the first four numbers of a longer dotted string like a 1.2.3.4.5 version.

IPv6 address (full or ::-compressed) · advanced

Matches an IPv6 address, both fully written out and the common shortened form with one :: run (e.g. fe80::211:32ff:fe12:3456). Good for duplicate-address and neighbour-discovery kernel lines.

```
(?:[0-9a-f]{1,4}:){2,7}[0-9a-f]{1,4}|(?:[0-9a-f]{1,4}:){1,6}:(?:[0-9a-f]{1,4}:){0,5}[0-9a-f]{1,4}
```

`eth0: IPv6 duplicate address fe80::211:32ff:fe12:3456 detected!`

Symbol	Meaning
`[0-9a-f]{1,4}`	one hextet: 1 to 4 hex digits (a-f case-insensitive)
`:`	the literal colon between hextets
`(?:...:){2,7}`	the fully-written form: 2 to 7 'hextet:' groups then a final hextet
`|`	OR — try the next alternative form
`(?:...:){1,6}:`	the compressed form: some hextets, then the :: double colon
`(?:...:){0,5}`	the hextets that follow the :: (may be none)

> **Note:** It does NOT match a bare :: run at the start like ::1 (loopback) or :: itself, nor IPv4-mapped forms (::ffff:192.168.1.1) or a %eth0 zone suffix. For those you need extra alternatives.

MAC address (colon-separated) · beginner

Matches a standard six-byte MAC address written with colons, like 24:5e:be:2a:1b:0c. Use it to grab hardware addresses from bridge, bonding and NIC-driver messages.

```
(?:[0-9a-f]{2}:){5}[0-9a-f]{2}
```

`br0: received packet on eth0 with own address as source address (addr:24:5e:be:2a:1b:0c, vlan:0)`

Symbol	Meaning
`[0-9a-f]{2}`	exactly two hex digits = one byte/octet
`:`	the literal colon separator
`(?:[0-9a-f]{2}:){5}`	five 'byte-colon' groups
`[0-9a-f]{2}`	the sixth and final byte

> **Note:** It only understands the colon format. It will NOT match dash-separated MACs (24-5e-be-2a-1b-0c) or the Cisco dotted form (245e.be2a.1b0c). Add a character class for the separator if you need those.

UUID / GUID (36-character 8-4-4-4-12) · beginner

Matches a canonical 36-character UUID such as a1b2c3d4-e5f6-7890-abcd-ef1234567890. Use it for by-uuid mount paths, filesystem/pool identifiers and systemd device names.

```
[0-9a-f]{8}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{12}
```

Sample text: `systemd[1]: Started File System Check on /dev/disk/by-uuid/a1b2c3d4-e5f6-7890-abcd-ef1234567890.`

Symbol	Meaning
`[0-9a-f]{8}`	the first block: 8 hex digits
`-`	a literal hyphen between blocks
`[0-9a-f]{4}`	a 4-hex-digit block (appears three times)
`[0-9a-f]{12}`	the final block: 12 hex digits

> **Note:** It requires the hyphens, so it will NOT match the compact 32-char form (a1b2c3d4e5f6...) or brace-wrapped Windows GUIDs. It also accepts any hex, so it does not enforce the version/variant nibbles a strict RFC 4122 check would.

Kernel uptime timestamp [seconds.microseconds] · beginner

Matches the bracketed seconds-since-boot stamp at the start of a dmesg line, like [12.345678]. Use it to detect or strip the kernel's default PRINTK_TIME prefix.

```
^\[\s*\d+\.\d+\]
```

Sample text: `[ 12.345678] usb 1-1: new high-speed USB device number 2 using xhci_hcd`

Symbol	Meaning
`^`	anchor to the start of the line
`\[`	a literal opening square bracket
`\s*`	any right-aligned padding spaces (zero or more)
`\d+`	one or more digits (the whole-seconds part)
`\.`	a literal dot
`\d+\]`	the microseconds digits, then the closing bracket

> **Note:** It only matches the default seconds.microseconds format. It will NOT match a human wall-clock prefix ([Fri Jul 17 10:22:01 2026]) or an ISO/syslog timestamp that some collectors prepend instead.

PCI address (domain:bus:device.function) · intermediate

Matches a full PCI BDF address like 0000:00:17.0. Use it to tie a driver message (ahci, nvme, igb, r8169) to the exact PCIe slot it came from.

```
\b[0-9a-f]{4}:[0-9a-f]{2}:[0-9a-f]{2}\.[0-7]\b
```

Sample line: `ahci 0000:00:17.0: AHCI 0001.0301 32 slots 6 ports 6 Gbps 0x3f impl SATA mode`

Symbol	Meaning
`\b`	word boundary before the address
`[0-9a-f]{4}`	the 4-hex-digit PCI domain (usually 0000)
`:[0-9a-f]{2}`	colon then the 2-hex-digit bus
`:[0-9a-f]{2}`	colon then the 2-hex-digit device (slot)
`\.[0-7]`	dot then the single-digit function (0-7)
`\b`	word boundary after the address

> **Note:** It requires the 4-digit domain. It will NOT match the older short form printed without it (00:17.0). If a tool omits the domain, add an optional (?:[0-9a-f]{4}:)? at the front.

NVMe namespace block device (nvmeXnY) · beginner

Matches an NVMe namespace block device such as nvme0n1, and optionally its partition (nvme0n1p1). Use it to spot which NVMe disk a message refers to.

```
\bnvme\d+n\d+(?:p\d+)?\b
```

Sample line: `nvme0n1: p1 p2 p3`

Symbol	Meaning
`\b`	word boundary so it starts a fresh token
`nvme`	the literal text 'nvme'
`\d+`	the controller number (one or more digits)
`n\d+`	'n' then the namespace number
`(?:p\d+)?`	optional partition suffix like p1

Symbol	Meaning
`\b`	trailing word boundary

> **Note:** This matches the block/namespace device, NOT the bare controller/char device nvme0 (that is just \bnvme\d+\b), and NOT the generic-char alias ng0n1.

WWN / WWID (64-bit hex, 0x or naa. prefixed) · intermediate

Matches an 8-byte World Wide Name written as 0x5000c500a1b2c3d4 or naa.5000c500a1b2c3d4. Use it to identify SAS/FC disks and multipath targets by their permanent address.

```
(?:0x|naa\.)[0-9a-f]{16}\b
```

Sample input:

```
mpt3sas_cm0: handle(0x000a), sas_addr(0x5000c500a1b2c3d4)
```

Symbol	Meaning
`(?:0x|naa\.)`	the prefix: either 0x or naa.
`[0-9a-f]{16}`	exactly 16 hex digits = the 64-bit WWN
`\b`	word boundary so it does not run into more hex

> **Note:** The exact {16} count is deliberate: it skips shorter hex like handle(0x000a) and matches only the real address. It will NOT match a 128-bit (32-hex) extended WWID or a colon-grouped form (50:00:c5:00:...).

USB topology path (bus-port chain) · intermediate

Matches the kernel's USB device path like usb 1-1 or usb 2-1.4.1 (bus number, then the port chain through any hubs). Use it to follow one USB device across enumeration messages.

```
\busb\s+\d+-\d+(?:\.\d+)*\b
```

Sample input:

```
usb 1-1: new high-speed USB device number 2 using xhci_hcd
```

Symbol	Meaning
`\busb`	the literal word 'usb' on a boundary
`\s+`	one or more spaces
`\d+`	the bus number

Symbol	Meaning
`-\d+`	hyphen then the root-hub port number
`(?:\.\d+)*`	zero or more '.port' steps for downstream hubs
`\b`	trailing word boundary

> **Note:** This is the sysfs/kernel topology form. It will NOT match the lsusb style 'Bus 001 Device 004', nor the idVendor:idProduct pair (1d6b:0002) that identifies the product rather than the physical path.

TROUBLESHOOTING

The common "it's not working" cases and what to do. Most reports are one of these.

Upload & session

- **Upload fails / "Failed to read bundle"** — The limit scales to the free disk space on the analyzer server — free up space there, or generate a smaller report. Very large multi-GB bundles are normally fine.
- **"Session not found"** — Sessions are cleaned up automatically after their lifetime. Just upload the bundle again (or re-click 🔬 on the Salesforce case).
- **"Smart values: none parsed"** — The bundle didn't include the drive SMART data. Re-generate the diagnostic report on the NAS — some older or partial bundles omit it.

Live tools (connected NAS)

- **"Daemon not connected" / Live tools show nothing** — The live tools need the daemon running on the target NAS, and the NAS reachable from the analyzer. Confirm it's running (`ps aux | grep sdlvd`) and start it if not.
- **"Permission denied" / "Requires admin"** — The root — check your role in User Management (0031. User management and roles).
- **Process / Filesystem / Memory shows nothing** — Same cause — they stream from the live daemon; no daemon, no data. memhound also needs its binary present on the NAS and the operation enabled.

Patterns

- **Pattern not matching** — Confirm it's in the Pattern Editor, then test it in the live preview (the → a sample line) — see 0016. Pattern Editor and Regex Helper. Editing a built-in pattern is done in the Pattern Editor, where the change persists.

Reading the data

- **Events are missing / unknown lines** — That's normal — its colour came from the kernel/event's own severity level, not a pattern. Read it anyway (0006. Reading the Kernel Log (dmesg)).
- **"Events are from a previous boot"** — Dr. Trace is telling you the incident is older than the current boot, so the disk/topology mapping may not match — treat it as a lead to confirm, not a fact.

> **Golden rule.** Every finding is a **lead to verify**, produced by pattern-matching over log text. Never act destructively (wipe, re-init,

> replace a disk) on the tool's word alone — confirm across SMART, the Kernel Log, and Dr. Trace first.

GLOSSARY

Term	Meaning
ARC	Adaptive Replacement Cache — ZFS's in-RAM read cache
await	Average wait time for an I/O operation (milliseconds)
backplane	The board connecting the drive bays to the NAS motherboard
CRIT	Critical severity (most urgent)
DMI	Desktop Management Interface — hardware inventory (model, DIMMs…)
DRBD	Distributed Replicated Block Device — HA block replication
FARM	Field Accessible Reliability Metrics — Seagate deep disk health (live-only)
fio	Flexible I/O Tester — the storage benchmark engine
HBS	Hybrid Backup Sync — QNAP's backup/replication app
HCL	Hardware Compatibility List — which disks are validated for a NAS
IOPS	Input/Output Operations Per Second
iSCSI	Internet Small Computer Systems Interface — block storage over IP
L2ARC	ZFS second-level (SSD) read cache
LVM	Logical Volume Manager (QTS storage stack)
MCE	Machine Check Exception — a CPU/hardware error
mdstat	The Linux software-RAID status (`/proc/mdstat`)
memtester	The built-in RAM fault tester
MTU	Maximum Transmission Unit (network frame size)
NCQ	Native Command Queuing — SATA command reordering
NVMe	Non-Volatile Memory Express — modern SSD interface
OOM	Out Of Memory — the kernel killing processes to survive
PCRE	Perl-Compatible Regular Expressions (the pattern syntax)
QPKG	QNAP Package — an installable NAS app
QTS	QNAP's ext4/LVM operating system
QuTS hero	QNAP's ZFS-based operating system
RAID	Redundant Array of Independent Disks
SAS	Serial Attached SCSI — enterprise disk interface
SATA	Serial ATA — the common disk interface
SLOG	ZFS separate intent log (sync-write log device)

Term	Meaning
SMART	Self-Monitoring, Analysis and Reporting Technology (disk self-health)
daemon	The privileged root daemon that runs the live tests on the NAS
SDU	StorageDiag Utilities — the local-only toolset (`sdu.qpkg`)
vdev	Virtual Device — a ZFS building block (mirror, raidz…)
ZFS	The Zettabyte File System (QuTS hero)

USER MANAGEMENT & ROLES

Access to StorageDiagLV is [illegible] — who can do what:

Role	Can do
[illegible]	Everything — passive analysis, the live tools, the root [illegible], [illegible], user management, and daemon configuration.
[illegible]	Passive analysis [illegible] the live diagnostic tools — but [illegible] the Terminal or user management.
[illegible]	Passive analysis only (no live tools).

> **How to open it:** Start menu → **User Management** (admin only). Or from here: @@OPEN:Users|Open User Management@@

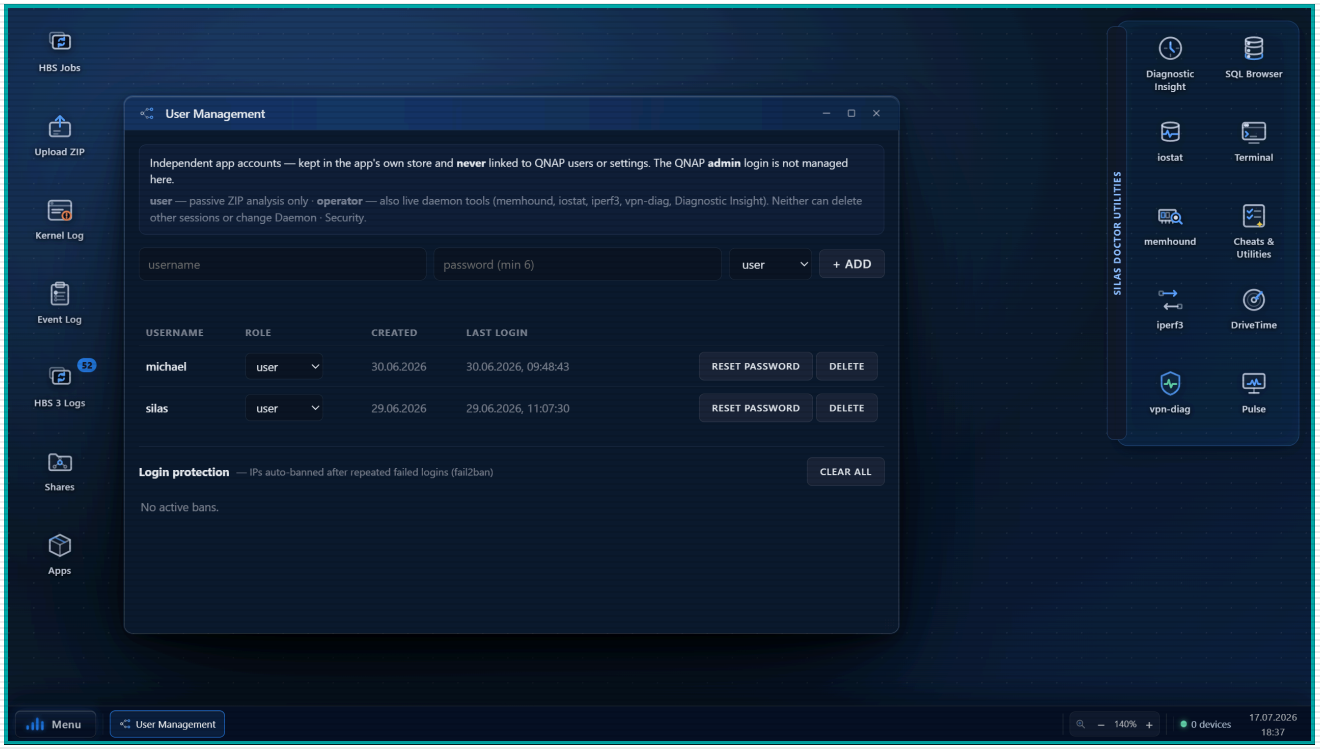


Figure — User Management: the app's own operator/user accounts (separate from the QNAP admin login), plus the fail2ban login-protection panel.

- [illegible] Sessions use secure, HttpOnly cookies in a private session store, isolated from other apps on the box.
- The QNAP [illegible] signs in with the NAS credentials; [illegible] and [illegible] accounts are created and managed here in the app.
- [illegible] protects the login against brute-force attempts.

Daemon security (for the live tools)

The live tools talk to the [illegible] daemon, which runs as [illegible] on the NAS. Its operations are allowlisted in the compiled binary — configuration can

only [illegible] operations, never add new ones. It ships three security presets:

- [illegible] – permissive (open socket, all ops, incl. a root shell). Development only.
- [illegible] – moderate.
- [illegible] – strict.

> **Important.** Run production in **safe** or **locked**, and never expose the app on a public, internet-facing host without HTTPS, strong credentials, and a firewall in front. See 0099. About, privacy and licence.

Watch it

> In-app demo: User Management — the roles that decide who may edit patterns and run the live tools.

DRIVETIME ANALYZER

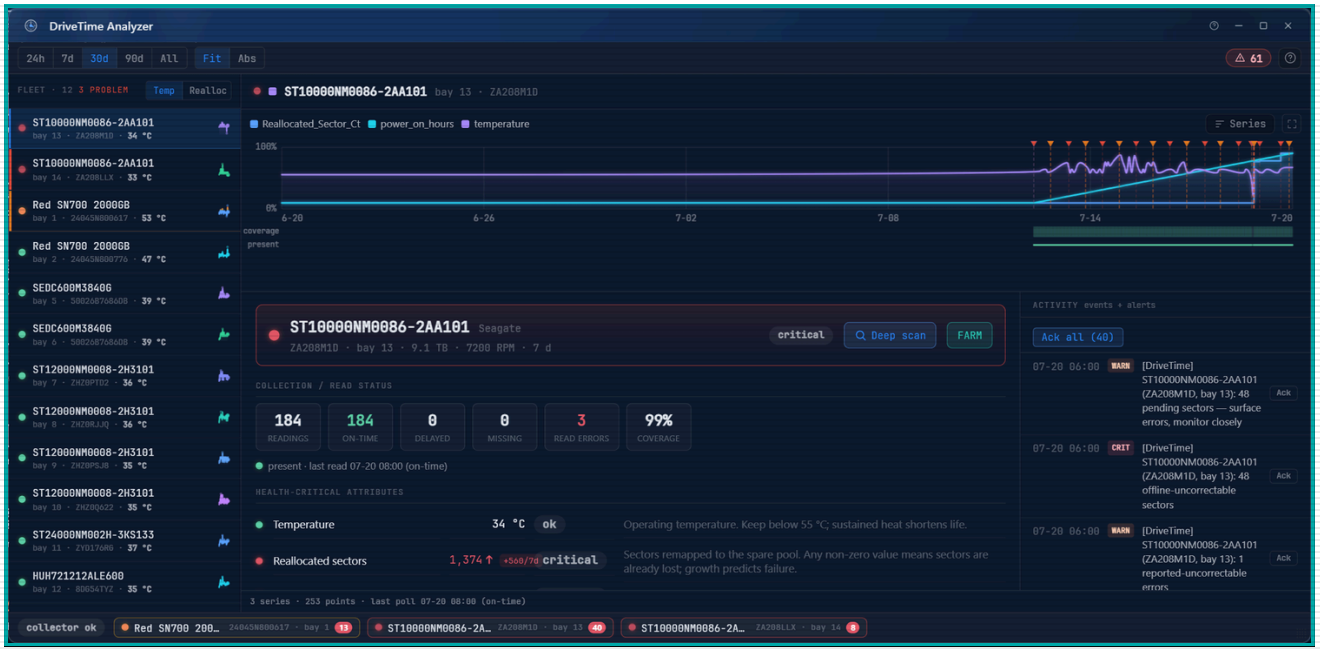
DriveTime is the fleet-wide drive-health view. Where the SMART viewer (0008. Reading S.M.A.R.T. disk health) shows one snapshot, DriveTime shows every disk over time – a wall of sparklines that makes a slow drift or a sudden step obvious at a glance, and turns raw SMART counters into ranked, explained findings.

> DriveTime reads a running history that a small on-NAS collector records at an adaptive cadence, plus the live SMART data. On an uploaded bundle it shows whatever history the bundle contains.

The fleet view

Each disk is one compact row: an identity colour (the same cool palette used across the app), the model, the bay, the serial, and the current temperature – followed by a sparkline of the selected metric.

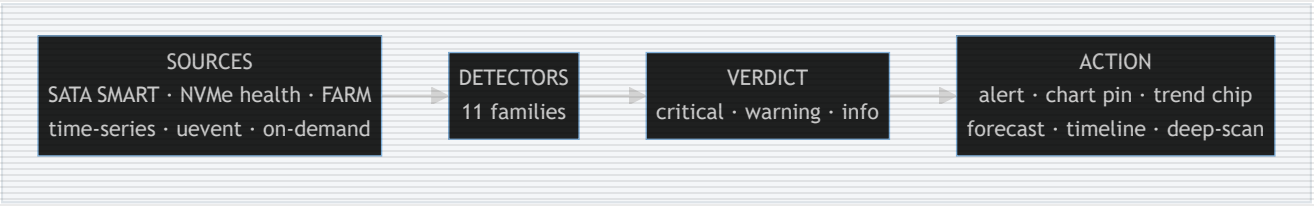
- Time range – `24h · 7d · 30d · 90d · All`.
- Scaling – `Auto` (auto-scale each sparkline to its own min/max, so shape is visible) or `Fixed` (a shared absolute scale, so disks are comparable to each other).
- Metric – `Temp · Realloc · Pending · CRC …`, or `All` at once, which lays the disks out as a responsive `All` (Temp | Realloc | Pending | CRC cells that drop from four columns to one as the window narrows).
- `Fleet` – `Fleet` (the overview), `Disk` (one disk, full charts with alert markers pinned on the timeline), `Log` (human-readable history), `Alerts` (what tripped a detector).
- A combined `Health` per disk always reflects its `Now` current signal, so a green fleet really is green.



The fleet view — every disk as a row with its temperature and a sparkline (left), the selected drive's trend chart and health-critical attributes (centre), and the live alert feed (right). Here a Seagate in bay 13 is : reallocated sectors are climbing — 1,374, up 560 in seven days.

How a verdict is reached

DriveTime doesn't just threshold one number. It runs a small pipeline, and the built-in help draws it as a :



- — the raw material: the drive's own SATA SMART attributes, NVMe health log, Seagate data (from a live deep-scan), the recorded , kernel , and scans.
- — eleven families of rules, each looking at one failure mode (see below).
- — every detector resolves to , , or .
- — what the verdict does in the UI: raise an , a marker on the chart at the moment it happened, show a or , place it on the , or suggest a .

The detector families

Open the → tab to see every detector, its live threshold, and — for admins — an editable value with / . The headline families:

Sector integrity (SATA SMART)

Media defects the drive reports through its own counters.

Detector	Trips at	Verdict
Reallocated sectors	raw ≥ 50 (or a fast 24 h burst)	critical
Pending sectors	raw ≥ 5	critical
Offline uncorrectable	raw ≥ 1	critical
Reported uncorrectable	raw ≥ 10	critical

The reallocated-sectors rule is written:

$$\text{crit} \;\Leftrightarrow\; \text{raw}_5 \geq 50 \;\vee\; \Delta_{24h} \geq 10$$

— i.e. a drive is critical either when the lifetime count crosses 50 when it gains 10 in a single day. A fast burst predicts imminent mechanical failure even below the lifetime threshold.

Interface & backplane (SATA SMART)

Errors on the cable / controller path, the platters.

- — `raw ≥ 100` → warning. CRC alone never condemns a disk.
- — if CRC rises on disks of one HBA together, the culprit is the , not the drives. This is the rule that stops you replacing a healthy disk.

The other families

Temperature, spin-up / mechanical, pending-vs-reallocated dynamics, SSD wear & lifetime, NVMe critical-warning bits, power-on-hours sanity, and a check — see below. Each is listed with its live threshold in the tab.

Tamper & SMART-reset detection

Some counters are supposed to . If one suddenly , the drive's SMART log was reset or the physical drive was swapped under the same slot. DriveTime flags that regression rather than trusting the new, artificially clean numbers — useful against a returned "known-good" drive that is really a factory-erased failing one.

The two rules DriveTime lives by

> **No invented "days-to-death".** DriveTime never prints a fake countdown to failure — the honest research says SMART simply can't predict that precisely.

> **Absence of a flag is not proof of health.** A green board means nothing tripped a detector, not that every disk is guaranteed healthy — SMART misses a large share of real failures, and almost all SSD wear. Always read it as "no evidence of a problem", and confirm anything suspicious in the SMART viewer.

Admin: tuning detectors

Admins can adjust or disable any detector from → : change a threshold, untick to disable a family, then ; restores the shipped values. Changes apply to this analysis only — an uploaded bundle keeps the shipped defaults unless you change them here.

Demo clips

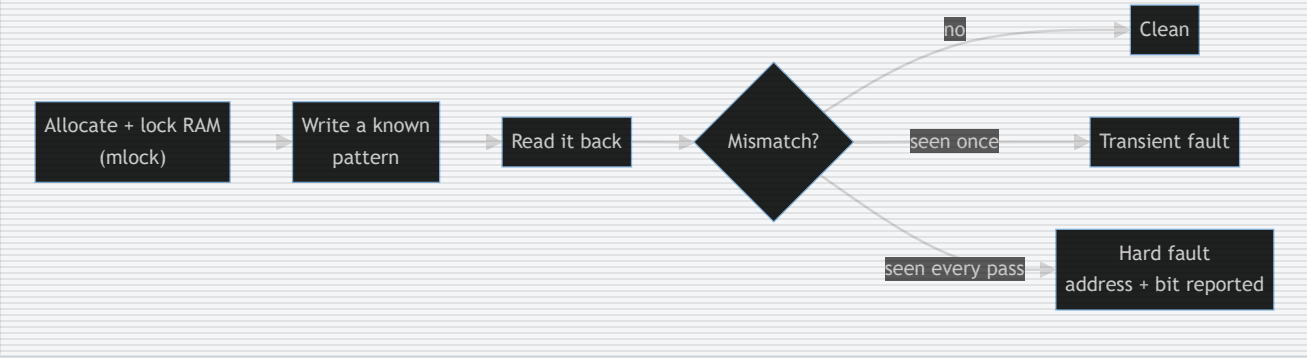
Short screen recordings of this window at work.

> In-app demo: A real read/write scan on a live pool, charted per disk beside the history it keeps.

MEMHOUND — THE RAM TESTER

hunts faulty RAM on a NAS. It allocates memory, it so the system can't swap it out from under the test, writes known patterns, reads them back, and reports the of any mismatch. Every suspected fault is : a one-off glitch is flagged , a defect that repeats is a confirmed fault.

> A live tool — it needs the **sdlvd** daemon on the target NAS and an **operator** or **admin** role. Reachable from **Live diagnostics** (0017. Live diagnostics and Doctor utilities).



Why "on a running NAS" matters

A classic memory test (like a boot-time MemTest) needs the machine offline. memhound is built to run on a NAS that's : it tests the RAM that is currently , keeps a safety margin, and backs off if other applications suddenly need memory — so it can look for faults without taking storage down. The deeper profiles go further and briefly free up almost all RAM (see below).

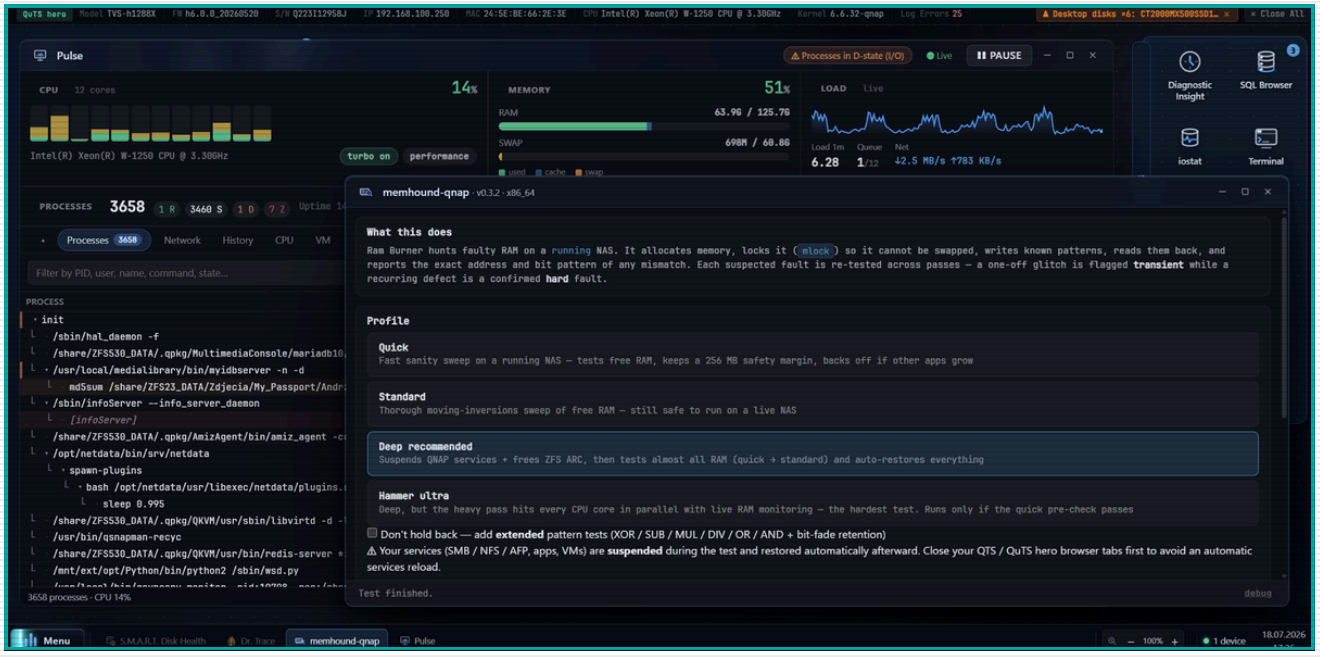
The four profiles

Pick the profile that matches how much disruption you can accept. They escalate from "safe on a live NAS" to "hardest possible test".

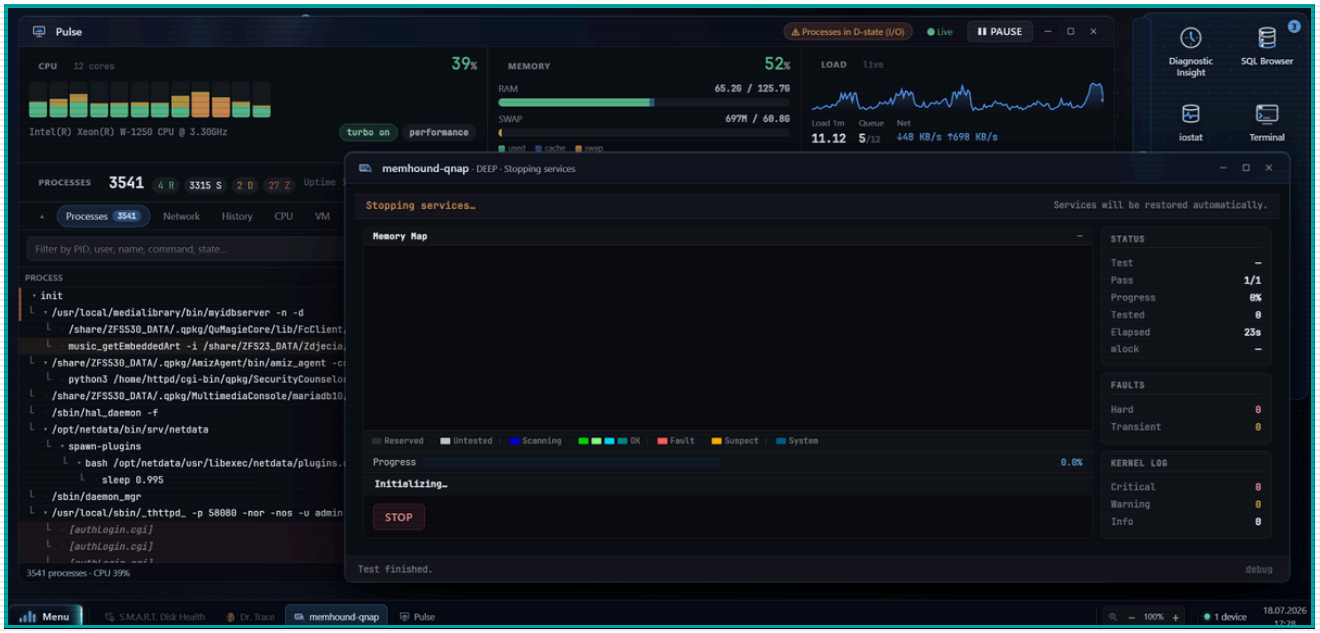
Profile	What it does	Disruption
	Fast sanity sweep of RAM; keeps a 256 MB safety margin and backs off if other apps grow.	None — safe on a live NAS.
	Thorough sweep of free RAM.	None — still safe on a live NAS.

Profile	What it does	Disruption
(recommended)	and frees the ZFS cache, then tests almost RAM (a quick pass followed by a standard pass) and everything afterwards.	Services paused for the duration.
(ultra)	Deep, a heavy pass that hits with live RAM monitoring — the hardest test. Runs the quick pre-check passes first.	Highest — full load, services paused.

> ⚠ **Deep and Hammer suspend your services** (SMB / NFS / AFP, apps, VMs) for the duration and restore them automatically when finished. **Close your QTS / QuTS hero browser tabs first** — an open tab can trigger an automatic services reload mid-test.



The profile chooser — Quick, Standard, Deep (recommended) and Hammer. The start button shows how much RAM is currently available to test.



Starting a Deep run: services are suspended first, then restored automatically when the test finishes.

Extended patterns (optional toggle)

Ticking adds a second battery of tests on top of the profile:

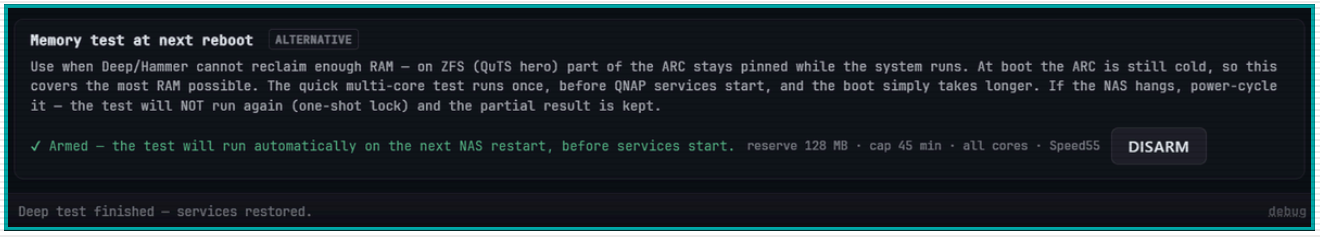
- – writes derived with . Different operations stress the bus and the cells with different bit transitions, catching faults a single pattern would miss.
- – writes a pattern, , then re-reads to catch cells that slowly lose their charge (a "soft" fault a fast sweep walks straight past).

Passes

The field repeats the whole run N times. More passes raise the chance of catching an fault and are what let memhound separate (seen once) from (seen every pass).

Run it at the next reboot

For the widest possible coverage without fighting the operating system for memory, . memhound schedules itself to run – when almost all RAM is free and cold – then lets the NAS finish booting normally. A small safety reserve and a hard time cap mean a stuck test can never hold the box down.



Armed for the next reboot — a memory reserve, a time cap and all CPU cores, with a one-click [illegible].

The test methods, in plain terms

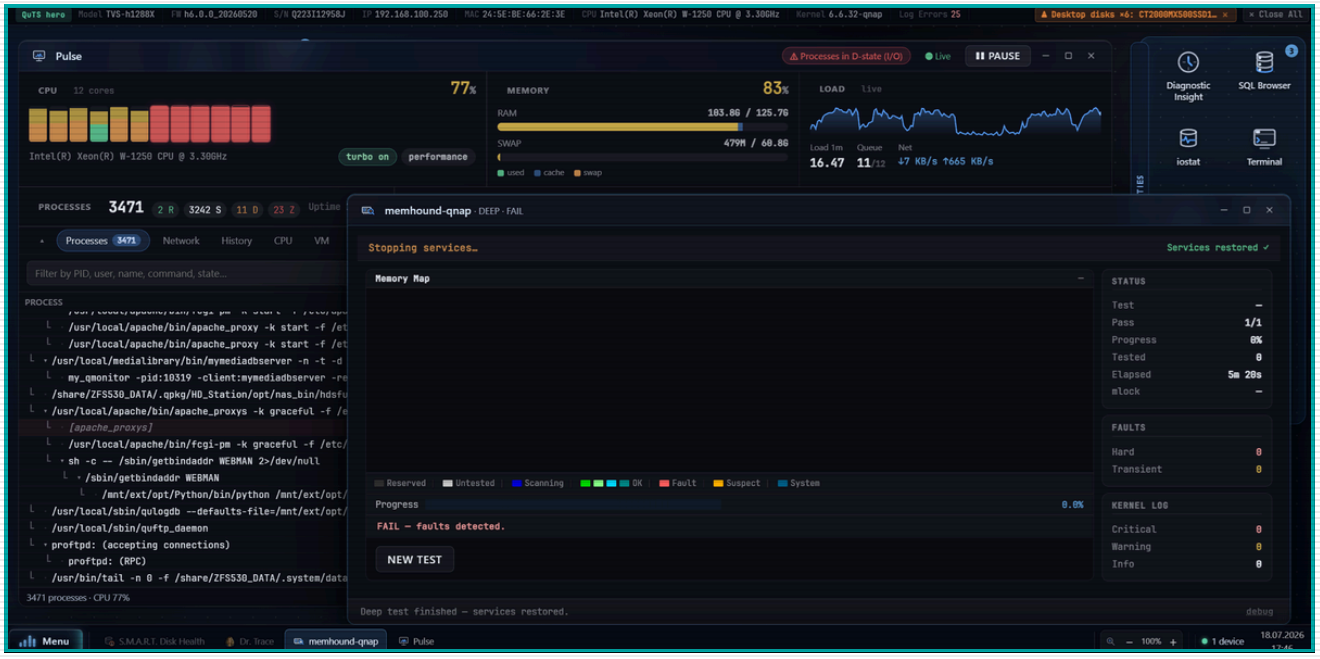
- [illegible] — fill memory with a value, read it back; any byte that comes back different is a fault, reported with its address and the exact bits that flipped.
- [illegible] — write a pattern everywhere, verify and rewrite its [illegible] while sweeping up through memory, then again sweeping down. This catches cells that are fine in isolation but flip when a [illegible] cell changes (address-line and coupling faults) — the workhorse of a good RAM test.
- [illegible] (extended) — the XOR / SUB / MUL / DIV / OR / AND battery above.
- [illegible] (extended) — the write-wait-reread test above.

Reading the result

- [illegible] — the tested RAM read back clean. As with any live test this covers the RAM that was [illegible]; use a deeper profile to widen coverage.
- [illegible] — a mismatch seen on one pass only. Treat as a lead: re-run with [illegible] to confirm or clear it.
- [illegible] — a mismatch that repeats. This is a real, reproducible defect; the report's [illegible] and [illegible] point at the failing module. Replace the DIMM (and re-test) — and if this NAS also shows a [illegible] warning or [illegible] errors in the logs, cross-check with 0013. Memory, OOM and crash dumps.



A clean Quick sweep — zero hard or transient faults across the tested span.



A Deep run that caught a fault — "FAIL — faults detected", with services already restored.

Demo clips

Short screen recordings of this window at work.

> In-app demo: Choosing between Quick, Standard, Deep and Hammer, with an explicit safety reserve.

> In-app demo: The memory map painting as each region is tested, so a fault gets an address and not just a count.

> In-app demo: Services stopped to reach almost the whole DIMM, then restored automatically when the run ends.

SYSTEM AND VAR LOGS

The and the answer most questions, but they are not the only logs in a bundle. QNAP's services each keep their own file under `var/log`, and when a specific subsystem misbehaves — the hardware abstraction layer, the iSCSI utility, a vendor driver — that subsystem's own log says more than the kernel ever will. The window is how you read them.

> **How to open it:** Start menu / desktop → **System Logs**. Or from here: @@OPEN:Varlog|Open System Logs@@

Pick a file first

The window opens with a rather than content, because a bundle carries a dozen or more service logs and there is no sensible default. Each entry shows its , which is the fastest hint about where the activity was: a log that grew to a few hundred kilobytes was busy, and a 150-byte log means the service barely spoke.

Typical files you will see:

File	What writes it
`hal_daemon.log`, `hal_lib.log`, `hal_module.log`	the hardware abstraction layer — bays, fans, enclosures, LEDs
`hal_util_net.log`	network-related hardware calls
`iscsi_util.log`	the iSCSI helper
`SDMD.log`	the storage daemon
vendor driver logs	for example a GPU or accelerator driver shipped as a package

Use and to walk to the previous or next file without going back to the chooser.

Reading it

Once a file is open the controls are the ones you already know from the Kernel Log, and they behave the same way:

- filters as you type, with a toggle for pattern matching.
- hides everything below the level you pick.
- on the right is a heat map of the whole file, so a dense band of errors is visible before you read a single line.
- apply here too, so a known signature gets its coloured pill and its Knowledge-Base badge exactly as it would in the

kernel log.

A pattern whose source is set to var/log only ever fires in this window, which is how service-specific signatures stay out of your kernel-log reading.

When to come here

Reach for these logs when the kernel log tells you that something failed but not why the service reacted the way it did. Two common cases:

- [illegible] The kernel reports the disk vanishing; `hal_daemon.log` reports what the enclosure hardware told the NAS at that moment.
- [illegible] The kernel shows the connection dropping; `iscsi_util.log` shows the helper's own view of the session.

> **Best practice.** Read the kernel log first to fix the time of the incident, then open the service log for the subsystem involved and jump to that timestamp. Working the other way round means reading a lot of routine service chatter before you know what you are looking for.

Watch it

> In-app demo: System / Var Logs — choosing a service log by size, then paging through it.

MANUAL DISK ANALYZER

Sometimes there is no diagnostic bundle. A customer sends a photo of the QNAP disk-health screen, or pastes the output of a disk-health command into a ticket, and that is all you get. The takes that fragment and runs it through the disk-health logic the SMART viewer uses, so you get the same verdict, the same attribute explanations and the same compatibility lookup you would get from a real bundle.

> **How to open it:** Start menu / desktop → **Manual Disk**. Or from here: @@OPEN:ManualDisk|Open the Manual Disk Analyzer@@

Three ways to get the data in

Drop raw disk-health output into the text box and press . The parser accepts the common layouts: the output of a disk-health command, and the table QNAP's own web interface shows.

This is the one that saves a support engineer real time. Paste an image with Ctrl+V, or drag the file in, and the attribute table is read straight off the picture. The reading happens – the image is never uploaded, never sent to a server, and no AI service is involved. That is why it works on a machine with no internet access, and why it is safe for a customer screenshot that may contain a serial number.

For a photo too poor to read automatically, or a single attribute quoted in an email, add rows yourself with and fill in the ID and value.

Fill in what you know

Above the attribute table there are fields for and the (HDD, SATA SSD or NVMe). None of them are required to get a verdict, but two of them change what you get:

- drives the disk-classification lookup, so filling it in tells you whether the drive is a NAS, enterprise, desktop, surveillance, archive or blacklisted model.
- , if you add it, enables the check – whether that exact drive is listed for that exact NAS.

matters more than it looks: the same attribute number means different things on a SATA disk and on an NVMe device, so choosing the wrong type produces confident nonsense.

Read the result

Press and the result is rendered as a normal SMART card. Everything in 0008. Reading S.M.A.R.T. disk health applies without exception: the header verdict is the worst level across all attributes, the failure family (reallocated, pending, uncorrectable, CRC) is what decides whether a disk is dying, and a value that has fallen below its threshold is the drive's own pre-failure signal.

empties the form. Your work survives a page reload, so a half-finished disk is still there if the browser restarts.

What it cannot do

The verdict is only as good as the numbers you fed it. Two limits worth stating to a customer:

- There is no history, so nothing here can tell you whether a count is climbing. For a trend you need the on-NAS collector described in 0032. DriveTime Analyzer.
- Per-head reliability data and vendor deep-scan results come from a live drive, so they cannot be recovered from a screenshot or a text paste.

> **Best practice.** Use this when a bundle is not coming — a pre-sales question, a drive already pulled out of the chassis, a customer who cannot run the diagnostic. When a bundle is available, upload it instead: it carries every disk in the enclosure plus the logs that explain what the disks did.

Watch it

> In-app demo: The Manual Disk Analyzer — where a pasted screenshot or text goes in, and where the verdict comes out. In-app demo: Close-up on the paste target that reads a SMART screenshot in the browser, and the Analyze button.

> In-app demo: Several SMART screenshots dropped in at once, read locally and turned into the attribute table.

> In-app demo: The hand-entered disk scored by the same rules as a bundle's — a warning on a Seagate NAS drive.

QSYNC AND APP-LOG BUNDLES

Not every diagnostic you receive is a whole-NAS capture. QNAP applications produce their own debug bundles — Qsync, Hybrid Backup Sync, Container Station and others — and those arrive as a much smaller archive containing just that application's logs and configuration. Upload one and the workspace opens a dedicated viewer for it instead of the usual storage windows.

> **How to open it:** upload an application bundle and the right window opens on its own. You can also reach it later from the Start menu — **Qsync Logs** for a Qsync capture. Or from here: @@OPEN:Qsync|Open Qsync Logs@@

Start from the file sizes

An app bundle is a pile of log files, so the window leads with a file list ordered by size, each entry showing its size. That layout answers the first question by itself: which file was actually busy during the incident. A log that grew to megabytes is where the story is; a handful of kilobytes usually is not.

Switch between the Logs and Files tabs to move between reading a log and browsing what else the archive contains.

Reading a log

Pick a file and the reading controls are the ones from the Kernel Log, working the same way: search with an optional regex toggle, a filter to hide routine lines, and a heat map of the whole file down the right-hand side. Detection patterns apply here too, so a known signature still gets its pill and its Knowledge-Base badge.

Databases inside the bundle

Some applications keep their state in a database rather than a text log. When the bundle contains one, the Open Databases button becomes active and shows how many it found. That opens the same read-only database viewer described in 0026. SQL Browser — useful for Qsync in particular, because the pairing and sync-state tables explain behaviour the text logs only hint at.

Which bundles are supported

Bundle	Window
Qsync	Qsync Debug Logs

Bundle	Window
Hybrid Backup Sync	HBS 3 Jobs — see 0027. HBS 3 — backup jobs and logs
other application captures	the generic app-log viewer

One format caveat: the uploader accepts ZIP, so an application that hands the customer a `.tar.gz` needs repacking before upload.

> **Best practice.** When a customer reports that a sync or backup is failing but the NAS itself looks healthy, ask for the **application** bundle rather than the full diagnostic. It is far smaller, it arrives faster, and it contains the one log that actually explains the failure.

Watch it

> In-app demo: A Qsync debug bundle — the file tree with per-file sizes, and the databases the bundle carries.

FILE BROWSER AND HTML REPORTS

Every dedicated viewer in this guide reads a specific file and presents it a specific way. Two windows do the opposite: they give you the bundle as it actually is. Use them when you need something no viewer surfaces, or when you want to confirm with your own eyes that a file really says what a viewer claims.

File Browser

A two-column file manager over the extracted bundle, laid out the way a classic file manager is: name, attributes, size, date and time, with `[..]` to go up and directories in brackets.

> **How to open it:** Start menu / desktop → **File Browser**. Or from here: @@OPEN:Filebrowser|Open the File Browser@@

It shows the bundle [illegible], so it is the fastest way to answer three questions:

- [illegible] A viewer that says "nothing found" is often telling the truth — the NAS never wrote that artifact. Browsing to the expected directory settles it.
- [illegible] A kernel log of a few hundred kilobytes instead of several megabytes usually means the ring buffer was cleared shortly before the capture, which limits how far back you can read.
- [illegible] Bundles vary by firmware and by NAS model. Browsing is how you find the configuration file or service log that no viewer has a window for.

Click a file to read it. Text opens for reading; binary content is identified as binary rather than dumped as garbage.

HTML Reports

QNAP builds its summary report inside the bundle, named after the device and typically a couple of megabytes of formatted HTML. This window lists the reports it found, with the path inside the archive and the size, and opens one in a new browser tab.

> **How to open it:** Start menu / desktop → **HTML Reports**. Or from here: @@OPEN:HtmlViewer|Open HTML Reports@@

Its real use is verification. Several viewers in this guide are built from this report rather than from raw log files — the process list, the ZFS pool status and parts of the hardware inventory all come from sections of

it. So when a viewer shows nothing, opening the report answers the question a parsed view cannot: is the data genuinely absent from the capture, or did the parser fail to find it? Reading the original, in the layout QNAP wrote it, settles that in seconds.

It is also the artifact a QNAP engineer is most likely to quote back at you, so being able to open the same page they are reading removes a whole class of misunderstanding. And it is the fallback when you want to work through a capture the old way, top to bottom, without any of the tool's interpretation in between.

> **Best practice.** Treat these two windows as the escape hatch, not the starting point. Work through the dedicated viewers first, because they apply the detection patterns, the thresholds and the verdicts. Drop to the File Browser or the raw report when a viewer is silent and you need to know whether that silence means "healthy" or "the data was never captured".

Watch it

> In-app demo: The File Browser over an extracted bundle, showing it exactly as QNAP packed it.

> In-app demo: QNAP's own summary report, listed with its path inside the archive.

ABOUT, PRIVACY & LICENCE

What StorageDiagLV is — and isn't

StorageDiagLV is a for analysing QNAP NAS logs and hardware state. It is and is made by, affiliated with, endorsed by, or supported by QNAP. It exists a diagnostic bundle and help you reach a conclusion faster.

Its findings come from over log text. They can be wrong, incomplete, or misleading — , and never act destructively (wipe, re-init, replace a disk) on its word alone.

Privacy — what leaves your machine

-
- AI analysis happens only if an administrator has explicitly configured an AI analyzer, and even then it is opt-in per run — before anything is sent the tool warns you and first lists the known, Knowledge-Base-documented issues so you can verify manually.
- The tool's whole purpose is : recognise known signatures in the logs, then (standard operating procedures).
- The app uses login/authorization and a small telemetry channel for maintenance purposes — for example, to pull updated (disk-compatibility) data and updated . The telemetry code belongs to the author and , including if the author determines the program is being put to other, unknown uses.

The components — a toolkit, not one program

- — the toolkit: passive log analysis, the live (daemon-backed) diagnostics, and all the viewers described in this guide.
- — the toolset, installed directly on a client NAS from the **sdu.qpkg** package. Its tools run — for example the SMART tool reads the drive's on-box data/logs. Note that deep telemetry such as can only be read from a drive, never reconstructed from an uploaded bundle's logs.
- — an RAM tester. It uses test patterns in the spirit of memtest86, plus an original RAM test. Its guiding principle is to :

  - If memhound a RAM fault, it is genuine to about .
  - If it does flag a fault — and essentially all RAM was actually freed and testable — there is still roughly a chance the RAM is faulty. When it truly matters, the definitive test is a full boot-time pass. (A clean memhound result is strong, but not absolute proof.)
- , authored for this project — not repackaged third-party software.

The root daemon — handle with care

The dashboard itself is . Its live tests hand operations to a on the NAS. In development mode that daemon accepts operation, including a full root shell — left that way, anything that can reach it owns the NAS.

- Use (Preferences → Daemon · Security).
- Do expose the app on a public, internet-facing host without , , and a in front. Obscurity is not security.

Licence & Disclaimer

Your purchase grants you a non-exclusive licence to install and use this application on your own QNAP system(s). You may not resell, redistribute, or republish it or its source. All rights remain with the author.

This software is provided "AS IS", without warranty of any kind. There is no guarantee that it runs without error, fits any particular purpose, or that its findings are correct. Results come from heuristic pattern-matching over log text and may be wrong, incomplete, or misleading — treat every verdict as a lead to verify, and never act destructively (wipe, re-init, replace a disk) on its word alone.

This application is not made by, affiliated with, endorsed by, or supported by QNAP. QNAP has not reviewed or confirmed any information, finding, or recommendation it produces, and bears no responsibility for the tool or its output. "QNAP", "QTS", "QuTS hero" and related marks belong to QNAP Systems, Inc.

The dashboard itself is unprivileged and never runs commands; its live tests hand operations to a daemon that runs as root on the NAS. In development mode that daemon accepts every operation, including a full root shell — left that way, anything that can reach it owns the NAS. Use Safe mode (Preferences →

Daemon · Security) and do not expose the app on a public, internet-facing host without HTTPS, strong credentials, and a firewall in front. Obscurity is not security.

Limitation of liability. To the maximum extent permitted by law, the author is not liable for any data loss, hardware damage, downtime, or other damages arising from use of this software, and total liability will not exceed the amount you paid. Nothing here limits liability that cannot be excluded by law, including your mandatory consumer rights.

> By clicking **I accept** you agree to these terms. If you do not agree, click **Decline** and the application will not open. (You see this once, on first launch.)