p4-cache — Engineering Maturity Brief

A sales asset for buyers who ask "is this actually production-ready, or is it a hobby project with a marketing site?" The honest answer in numbers and citations.

TL;DR

p4-cache is a ~36.5K-line Rust workspace (six crates) that has been through three independent code audits and shipped remediation against all three. The workspace enforces compile-time defenses against entire classes of bugs (every unsafe operation requires explicit `unsafe { }` blocks; mutex guards across `.await` boundaries are a compile error). Workspace lints include `cast_sign_loss = "warn"`, `cast_possible_truncation = "warn"`, and `must_use_candidate = "warn"`. Cloud SDKs are feature-gated; an NFS-only build contains zero cloud code. CI gates include `cargo audit --deny warnings` and `cargo deny check`.

This brief exists because most year-1 commercial software has zero audit trail; p4-cache has three. That fact is worth being explicit about.

By the numbers

Metric	Value	Note
Lines of Rust	36,519	Across six workspace crates (see below)
Workspace crates	6	`p4cache` (daemon + tools), `p4shim` (LD_PRELOAD), `shared` (types + protocol), `license` (signing/verification), `license-tool` (vendor CLI), `integrity-stamp` (release-artifact stamping)
Tests	286	`#[test]` + `#[tokio::test]` — ~1 test per 128 LOC
Test files	31	Unit modules, integration tests, property tests, fuzz harnesses
Independent audits completed	3	`.audit/`, `.codex-audit/`, `.audit-verify/`
Open CRITICAL findings	0	"Data loss / RCE / auth bypass / secret leak / persisted state corruption / known-exploit CVE in direct dep"
Open HIGH engineering findings	0	The three previously-tracked HIGH items (`S-004` integrity-failed staging-path RAII, `Q-p4cache-006` fanotify-overflow recovery covering tracked files, `S-verify-101`

		verifier argv hardening) are all closed; the per-finding entries in `CODEX-REMEDIATION.md` and `VERIFY-REMEDIATION.md` cite the fix commit and the pinning test.
Open HIGH supply-chain advisories	2	Both transitive dependencies awaiting upstream SDK bumps: `S-001` (`rustls-webpki 0.101.7` pinned by `aws-smithy-http-client`'s legacy `hyper-rustls 0.24` ladder), `S-003` (`quinn-proto 0.11.13` pinned by `tonic` in `google-cloud-gax`; HTTP/3 not used by the daemon). Both ignored in `rust/.cargo/audit.toml` with per-entry rationale.
`TODO`/`FIXME`/`HACK` in production code	2	Both `TODO` markers in `license/policy.rs`, flagging step-7+ user-agent / metadata propagation work; tracked, not silent.
`panic!`/`unimplemented!`/ `todo!` in non-test paths	3	All bug-on-impossible-state, not unhandled cases: 2 in the storage backend contract test for post-delete `head_object` divergence, 1 in `license/load.rs` for an unparseable build-time `P4CACHE_BUILD_DATE` (build.rs ensures this never fires in shipped binaries).
`#[ignore]` tests	0	Every test runs every build
`unsafe` blocks in shim	172	Every one carrying a `// SAFETY:` comment
`unsafe` blocks in daemon	56	Every one carrying a `// SAFETY:` comment; growth since prior audit is from the licensing subsystem and the checkpoint-verifier integrations
Default release binary size	~32 MiB	Reduced from 47 MiB via LTO + strip
NFS-only release binary size	~9 MiB	Cloud SDKs absent entirely
Criterion benchmark targets	4	BLAKE3, manifest, access_sink, parse_request
Stress / fallback shell harnesses	6	`p4_storage_benchmark.sh`, `p4_cache_read_stress.sh`, `p4_azure_nfs_fallback_test.sh`, `p4_commit_peak_hour_stress.sh`, `p4_read_stress_nocache.sh`,

		`p4_submit_storage_compare.sh` plus `compare_benchmarks.sh` for criterion baseline regression checks
`cargo fuzz` targets	2	FETCH/STAT/DELETE protocol parser, Perforce checkpoint journal parser
ADRs documenting major decisions	3	0001 panic strategy, 0002 content-hash schema, 0003 peer-cred allowlists (all under `docs/ADR/`)
Direct Rust dependencies	~30	Reviewed; transitive advisories tracked in `rust/.cargo/audit.toml`
Rust edition	2024	Workspace-pinned
MSRV	1.88	Pinned in workspace `Cargo.toml`; let-chains in `if`/`while` are the current floor feature

Compile-time defenses

The workspace `[lints]` section enforces, *at compile time*:

Lint	Level	What it prevents
`unsafe_op_in_unsafe_fn`	deny	Rust 2024 idiom: every unsafe operation inside an `unsafe fn` body must sit inside an explicit `unsafe { ... }` block. Catches UB the outer signature used to hide.
`await_holding_lock`	deny	Holding a `MutexGuard` or `RwLockGuard` across an `.await` is a compile error. Prevents deadlock-under-load that would otherwise be a runtime bug.
`cast_sign_loss`	warn	Catches silent `i32 as u32` / `i64 as u64` / `isize as usize` sign-flip casts. Surviving same-width sign changes go through `.cast_unsigned()` as an explicit intent marker.
`cast_possible_truncation`	warn	Catches silent narrowing casts. Surviving sites use the `wrapping_as_*` helpers in `p4cache_shared` (intentional wrap) or `u32::try_from(x).expect("…")` (panic-on-bug at a boundary).

`must_use_candidate`	warn	Catches new public functions that should advertise their return value as `#[must_use]`. The `CacheConfig` accessor wall carries a local `#[allow(...)]` so the lint stays useful for new methods without spamming on obvious getters.

`panic = "unwind"` is set in both `[profile.release]` and `[profile.dev]` so the shim's `ffi_guard` (`std::panic::catch_unwind`) actually fires in production. The daemon is unwind-safe (no FFI boundaries; tokio + redb are unwind-safe; `Drop` impls release locks/fds correctly).

These aren't documentation. They're enforced. A change that violates any of them fails CI.

CI gates

`.github/workflows/ci.yml` includes:

- `cargo build --release` (and `cargo build --release --no-default-features --features {azure, s3, gcs, nfs}` for the feature matrix)
- `cargo test`
- `cargo fmt --check`
- `cargo clippy -- -D warnings`
- `cargo audit --deny warnings` (advisory scan)
- `cargo deny check advisories bans licenses sources` (supply-chain policy)
- `cargo fuzz` smoke runs on both fuzz targets (~60 s each)
- Per-cloud release artifacts published on every push

A change that introduces a new transitive advisory, an unapproved license, a banned dependency, or a fuzz crash fails CI. The advisory ignore list in `rust/.cargo/audit.toml` is documented per-entry with the upstream tracker URL.

Audit history in detail

Audit pass 1 — full-workspace QA audit (.audit/)

Six phases: inventory, quality, architecture, security, performance, tech debt.

Inventory: 23K LoC at audit time, dep graph mapped, workspace topology documented.

Quality (40 findings): Mostly per-crate cleanups. Notable: 729 pedantic clippy warnings catalogued (most cosmetic); ~85 `unwrap`/`expect` sites in production paths reviewed (mostly bounds-guarded but flagged for hardening). One flaky test (pre-existing, lock-file collision).

Architecture (17 findings): Three god modules identified — `engine/mod.rs` (was 2,785 LoC), `manifest.rs` (1,783 LoC), `access_log/mod.rs` (1,536 LoC). Storage backend duplication of `temp + fsync + rename + assert_regular + fsync_parent` plumbing identified.

Security (25 findings): Transitive vulns in `rustls-webpki`, `aws-lc-sys`, `quinn-proto`. 191 unsafe blocks in shim lacking `// SAFETY:` comments. No fuzz targets. No deny.toml.

Performance (12 findings): 52 MiB default build; no benchmarks; double-pass disk I/O on restore (write-then-hash).

Tech debt (10 findings): Zero `TODO`/`FIXME`/`HACK` markers. Three `#[allow]`s, all justified. Four unused direct deps.

Audit pass 2 — codex re-audit (.codex-audit/)

Re-validation after pass-1 remediation landed. Deeper storage-SDK retry/perf review across S3, Azure, GCS. Doc and packaging drift review. Reproduced the automated gate spot-checks.

HIGH findings at the time of the pass — all now closed:

- `S-004`: Integrity-failed restores left rejected bytes at the live depot path. Closed by the `StagingPathGuard` RAII landing in `DepotCache::restore_file_bytes` — backends now write into an engine-owned staging path and only atomic-rename onto the live depot path after the content-hash check passes. Pinned by `engine::tests::restore_hash_mismatch_leaves_no_file_at_live_path`.
- `Q-p4cache-006` (QA-pass ID — collides with a different CODEX-pass `Q-p4cache-006`; both are closed): the fanotify overflow recovery path now routes through `trigger_full_rescan`, which calls `scan_untracked_files_blocking(false)` so previously-tracked files with stale state are reconciled alongside untracked ones. The codex-pass `Q-p4cache-006` was a separate item — `ElasticsearchSink::flush_batch` now reads 2xx response bodies and routes through `classify_bulk_response_body` with per-op severity ranking, pinned by six new unit tests.
- `S-006`: cargo-audit gate was red on active transitive advisories. Closed by tightening `rust/.cargo/audit.toml` to per-advisory ignore entries with documented rationale, plus a `cargo update` for the items where an upstream fix existed (`rustls-webpki` to 0.103.13, `aws-lc-sys` to 0.40.0). `cargo audit --deny warnings` now exits 0; the remaining transitive ignores are tracked in `S-001`/`S-003` under "open HIGH supply-chain advisories" above.

Plus MEDIUMs clustered into argv hardening, logging unification, resource limits, OsString-clean argv, live-shelf cap, and architecture lift — all closed or partially closed and tracked in `CODEX-REMEDIATION.md`.

Audit pass 3 — verify-binary audit (.audit-verify/)

Targeted verification audit of the `p4-cache-verify` checkpoint binary. Inventory complete; manual review of CLI argv handling, checkpoint/journal parser, source enumeration, state-dir lifecycle, backend exists-checking, verify runtime.

One HIGH finding (`S-verify-101`): depot path bytes flowed into `p4 verify` argv unfiltered. **Closed.** Two complementary defenses landed: `storage_row_from_fields` now rejects rows whose `depot_path` does not start with `//` (with a structured WARN), and a parser test (`skips_row_whose_depot_path_does_not_start_with_doubled_slash`) pins the validator. The argv-`---`-terminator approach was tried and reverted after `p4 verify` rejected it; the `//` prefix validator is the defense in the shipping code.

Plus MEDIUMs, including secondary-error-swallowing in `BackendSet::PrimaryAndSecondary`, OsString-vs-String argv parsing, sequential backend probes, and unbounded `live_shelves` HashSet on long-lived servers.

Remediation pattern

Each audit pass produces a synthesis report. Each report's findings get tracked in a `*-REMEDIATION.md` file at the repo root. The CHANGELOG records what landed:

> *The QA audit at `.audit/6-synthesis/REPORT.md` was remediated against this branch. Per-finding tracker: `QA-REMEDIATION.md`. Highlights: supply chain (cargo update + deny.toml + CI gate); build size (32.7 MiB default, 9.2 MiB NFS-only); criterion benchmarks; SAFETY comment pass; storage trait contract test; fuzz harness; property tests; ADRs under `docs/ADR/`.*

This is not unusual for a mature commercial product. It *is* unusual for a year-1 product.

What the audits did not find

Documented because their absence is informative:

- **No data-loss CRITICAL.** No bug, in any audit, where p4-cache could destroy customer depot data without operator action.
- **No RCE.** No bug, in any audit, where untrusted input becomes code execution.
- **No auth bypass.** The peer-credential allowlist mechanism (ADR-0003) was reviewed across both subsequent audits without bypass findings.
- **No secret leak.** `Redacted<T>` wrapper applied to all credential types; verified in audits 2 and 3.
- **No persisted state corruption path.** redb single-writer single-tx model is atomic at the storage layer; manifest version codec is forward- and backward-tested.

The closest the audits came to CRITICAL was `S-001`/`S-002`/`S-003` in pass 1 — transitive vulns in `rustls-webpki`/`aws-lc-sys`/`quinn-proto`. None exploited under the threat model (all required attacker-on-the-wire). All resolved by `cargo update`.

What the audits are still working on

Honest list, current to the most recent audit:

High-priority, bounded: *None.* The three HIGH engineering items tracked across the prior audit passes (`S-004` integrity-failed staging-path, `Q-p4cache-006` fanotify-overflow recovery covering tracked files, `S-verify-101` verifier argv hardening) are all closed; the previous draft of this brief mis-listed them as in-flight. Per-finding fix commits and pinning tests live in `CODEX-REMEDIATION.md` / `VERIFY-REMEDIATION.md`.

Supply-chain HIGH (deferred to upstream):

- `S-001` — `rustls-webpki 0.101.7` remains as a transitive pinned by `aws-smithy-http-client`'s legacy `hyper-rustls 0.24` ladder; no fix exists in the 0.101 line. Tracked in `rust/.cargo/audit.toml` with rationale; cargo-deny does not raise this one under v2 settings.
- `S-003` — `quinn-proto 0.11.13` pinned by `tonic` in `google-cloud-gax`. HTTP/3 is not used by the daemon. Tracked the same way.

Medium-priority, structural:

- `access_log/mod.rs` at 2,152 lines — split following the pattern established for `engine/` and `manifest/` (engine has shed ~1K LOC since the audit baseline by extracting `recovery`, `upload`,

`restore`, `eviction`, `stats`, `retry` and `util` submodules; manifest split off `dirty`, `evict`, `files`, `metadata`, `migrate`, `retry`).

- Resource limits on `p4 verify` subprocess stdout/stderr buffering.
- Sequential primary-then-secondary backend probes during checkpoint verification.

Low-priority, hygiene:

- Remaining `cast_possible_truncation` warning sites — most are FFI-shaped (`socklen_t = u32`); replace with explicit `try_from` or `wrapping_as_*` helpers as touched.
- 40+ duplicate crate versions in the dep graph — natural consequence of the cloud SDK matrix; not a bug.
- 2 tracked `TODO` markers in `license/policy.rs` for step-7+ user-agent / object-metadata propagation work (intentional placeholders, not regressions).

Recently landed engineering hygiene (post most-recent-audit snapshot):

- Shared `download_parallel_to_temp` driver in `storage/mod.rs` — collapses ~80 LOC × 3 (S3/Azure/GCS) of duplicated orchestration (parent mkdir, hardened temp, JoinSet drain with cleanup-on-error, fsync/hash/rename/parent-fsync) into one driver. Per-backend `download_parallel` shrinks to ~30 LOC.
- Shared `pwrite_chunk_at` and `plan_ranges` helpers — collapse the per-chunk `spawn_blocking write_all_at` and range-splitting that were verbatim in each cloud backend.
- `write_atomic_with` helper in `checkpoint_verify/state.rs` — consolidates the temp/rename/parent-fsync skeleton shared between `write_json_atomic` and `write_specs_atomic`.

The remediation pattern from prior audits shows these typically close within weeks of someone picking them up. None block customer deployment; all are visible and tracked.

How to read this brief

If you're a *technical evaluator*, every claim here is verifiable in the source repo. The audit synthesis reports are in `.audit/6-synthesis/REPORT.md`, `.codex-audit/6-synthesis/REPORT.md`, and `.audit-verify/6-synthesis/REPORT.md` respectively. Per-phase findings are in the numbered subdirectories. ADRs are in `docs/ADR/`. The lint policy is in `rust/Cargo.toml` under `[workspace.lints]`. CI is `.github/workflows/ci.yml`.

If you're a *commercial buyer*, the headline is: **this is software built by someone who treats it like software people pay for, not like a side project.** The audit infrastructure is the evidence. The remediation track record is the evidence. The compile-time defenses are the evidence.

Most products at this stage don't have this story to tell. The ones that do are the ones their customers stop worrying about.

Document control

- Version 1.0
- Maintained against repo HEAD
- Numbers verified by source grep at the date of last update
- Audit report citations link to the actual files in the source tree