Downloaded from buildwithruud.com · Kami Document System

Escaping the 600-Crate Monorepo Tax: Rust Polyrepo Co

2026.11.16 · /blog/rust-polyrepo-nextest-template-db-nixos-cgroups

1. Escaping the 600-Crate Monorepo Tax Without Losing Compile-Time Safety

As our multi-tenant Rust platform (**Atlas**) expanded to power four distinct WebAssembly and SSR applications (**Folio**, **Anchor**, **Plaza**, and **Platform Admin**), keeping every frontend inside the backend Cargo workspace created three compounding bottlenecks: workspace-wide `.cargo-lock` contention, 20-minute CI runs on minor UI edits, and backend `kubectl apply -k` deployments overwriting independently promoted UI images.

We solved this by extracting each product UI into its own trunk-based (`main`) repository while enforcing a strict invariant: **Sync wire contracts and container SHAs—never git branch tips.

1. **Tag-Pinned Wire Contracts (`atlas-api-types`)**: All shared HTTP DTOs and closed domain enums live in a feature-gated crate (`default-features = false`) pinned by annotated git tags (`atlas-api-types-v0.2.1`). A CI `grep -rIn "mirrors backend"` gate hard-fails any PR that redeclares a backend struct locally.
2. **30-Second UAT Tag Promotion**: Pushing an annotated `<app>-uat-YYYY.MM.DD` git tag mirrors the tested GHCR image and rolls the UAT deployment in ~30 seconds, while `deploy_platform_k8s` snapshots and preserves live tag-promoted UI images across Kustomize applies.

2. Sub-Second `cargo-nextest` Template DBs & NixOS Cgroup Isolation

Cross-Process Postgres Template Cloning: Because `cargo-nextest` runs each test in a separate OS process (resetting `OnceLock`) and `#[tokio::test]` spawns a separate runtime per test (bypassing `tokio::sync::Mutex`), our test harness coordinates via `SELECT pg_advisory_lock(8142012026)`, stamps the migration hash into `pg_catalog.shobj_description` (`COMMENT ON DATABASE`), and clones isolated per-test databases (`CREATE DATABASE ... TEMPLATE`) with `synchronous_commit = off` and a 2-connection pool cap.

Control-Plane vs. Data-Plane Cgroups on NixOS: On our 32-thread, 192GB DDR5 bare-metal `NixForge` host, 585-crate Cargo builds saturated NVMe IO and starved `podman.service` (`exit 126` socket timeouts). Pinning `podman.service` to `Nice = -5` / `CPUWeight = 1000` while isolating `woodpecker-agent` in `IOSchedulingClass = idle` with per-branch `CARGO_TARGET_DIR` mounts eliminated CI timeouts completely.

3. Portable Engineering Rule

Let the Rust compiler enforce cross-repo wire contracts via immutable git tags, coordinate multi-process database tests inside PostgreSQL via advisory locks and template cloning, and always isolate your container socket control plane from your build data plane in Linux cgroups.