

Lessons from Optimizing Phoenix Integration Tests



From 10-Minute Black Box to 3-Minute Transparent CI

Rodolfo Carvalho

 praialabs.com

Elixir Vienna Meetup

September 24, 2026

About Me & Praia Labs

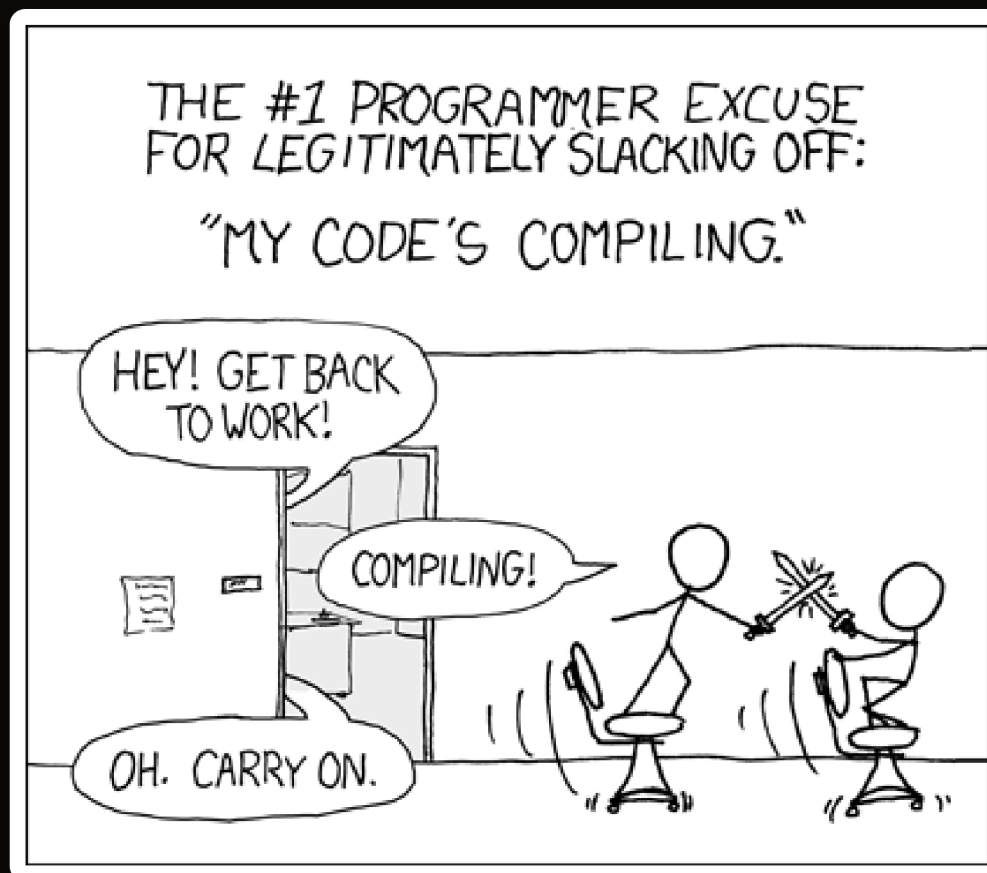
Background Building with LiveView

- Built and scaled **SaaS product** on Phoenix LiveView over the past few years.
- Now consulting through **Praia Labs**, helping teams design, build, and scale applications with Phoenix & LiveView.
- Navigating AI-assisted development, mostly staying in the driver's seat.

Philosophy Caring for Dependencies

- Long-time contributor to **Phoenix**, **LiveView**, **Hex**, and the Elixir ecosystem.
- **Why work upstream?**
 - A strong belief in giving back and actively *caring for our dependencies*.
 - Investing in upstream speed and stability benefits every developer in the room.

Waiting for Long Builds



"The #1 Programmer Excuse for Legitimately Slacking Off: My code's compiling." (XKCD #303)

Context: Testing Phoenix Itself (Upstream)

Scope Framework, Not an App

We are **not** talking about testing a typical app built with Phoenix.

We are testing **Phoenix itself**
(`phoenixframework/phoenix`):

- Testing code generation output end-to-end.
- Matrix across supported Elixir & OTP versions.
- Compatibility with 4 database engines: PostgreSQL, MySQL, MSSQL, SQLite3.

Universal Why This Matters to You

The lessons and wins apply to **any project**:

- **ExUnit concurrency model**: How tests actually get scheduled across CPU cores.
- **Observability**: Custom formatters & CI step summaries.
- **Universal speedups**: In-memory `tmpfs` RAM disks, caching tricks, and Docker ergonomics.

Phoenix Test Taxonomy: What Tests What?

The Phoenix repository contains **4 distinct test suites**:

- **Unit & Core** `/test`:
Unit & functional tests for framework internals (Router, Endpoint, Channels, PubSub).
- **Installer** `/installer/test`:
Validates `mix phx.new` scaffolding logic (does it generate expected files & trees?).
- **Generators** `/test/mix/tasks/`:
Validates `phx.gen.html`, `phx.gen.live`, `phx.gen.auth` code generation against mocked apps.
- **Integration** `/integration_test`:
The real deal. Generates a complete app on disk, runs generators, compiles with `--warnings-as-errors`, verifies `mix format`, connects to real databases, migrates, and runs `mix test` inside the generated app!

The "Integration Test Tax" & The Black Box

- Every Phoenix PR carried an opaque **~10-minute CI tax**.
- **Worse than dots (....):**
 - CI runners buffer standard output line-by-line or by chunk.
 - Because ExUnit outputs progress dots without newlines, **dots didn't stream interactively!**
 - You literally stared at an empty spinning wheel for 8–10 minutes with zero feedback.
- **The Developer Anxiety:**
 - Did MSSQL hang on startup?
 - Is `mix deps.get` stalled on a network timeout?
 - Did compilation fail?
 - Nobody could tell until the whole job finished or timed out.

You Can't Optimize What You Can't Measure

The Intuitive First Attempt: `mix test --slowest`

- ExUnit provides `--slowest N` and `--slowest-modules`.
- **The Profiling Trap:**
 - In ExUnit, passing `--slowest` automatically forces `--trace`!
 - `--trace` sets `--max-cases 1` (forcing all async tests to run **serially**) and sets `timeout: :infinity`!
 - Profiling destroyed the suite's concurrency and blew up wall-clock time!
- **Job-Level Opacity:**
 - Monolithic `test.sh` bundled system packages, DB boot, dependency fetch, compilation, and test execution into a single timed blob.

Stage 1: Gaining Visibility (PR #6825 & PR #6817)

CI Step Decomposition (PR #6825)

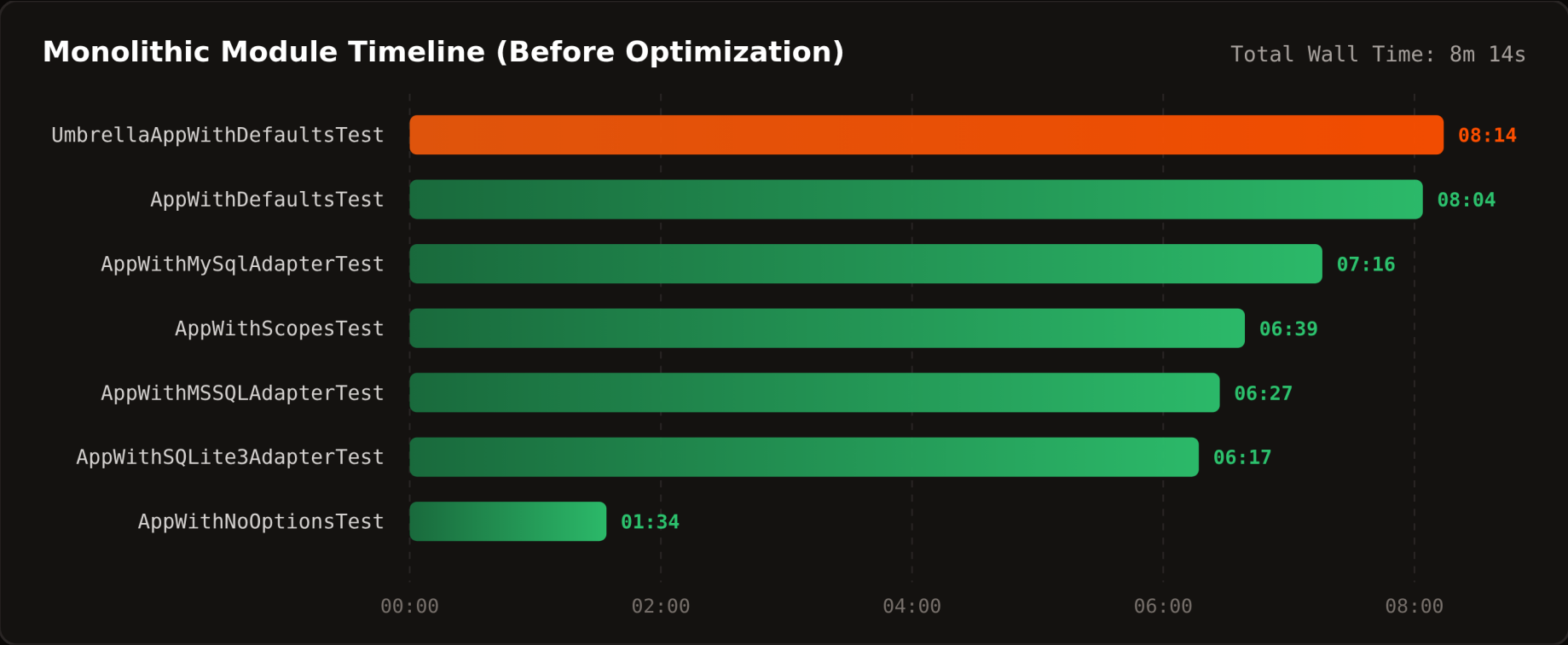
- Replaced monolithic `test.sh` with discrete GitHub Actions workflow steps.
- Surfaced exact phase timings in the Actions UI:
 - Package install: ~15s
 - DB healthcheck wait: ~20s
 - Deps & NIF compile: ~45s
 - Test execution: ~8m+

SummaryFormatter (PR #6817)

- Custom lightweight ExUnit formatter.
- Runs at **full async concurrency** (`max_cases: 8`).
- Preserves active timeout protection.
- Appends a structured Markdown duration table directly to `$GITHUB_STEP_SUMMARY` !

Baseline Timeline: The Monolith in Action

Timeline rendered by `SummaryFormatter` before splitting:



All 7 modules launch at 00:00. Fast modules finish early; 2 stragglers drag past 8 minutes.

The Straggler Effect: vCPUs Idling

- Standard GitHub Actions Linux runners have **4 vCPUs** (16 GB RAM).
- **What happened during execution:**
 - $T = 0$: ExUnit starts all 7 modules (8 slots available).
 - $T = 1\text{m}30\text{s}$: Fast modules finish (`AppWithNoOptionsTest`).
 - $T = 7\text{m}00\text{s}$: Most modules finish.
 - $T = 7\text{m to } 8\text{m}+$:
 - **1~2 cores** still executing tests.
 - **Other vCPUs sit completely idle** doing nothing!
- The entire test suite wall-clock time was bounded by monolithic stragglers.

Revelation #1: ExUnit's Scheduling Model

- How does ExUnit run asynchronous tests?
 - ExUnit parallelizes across **modules** (`async: true`).
 - BUT tests *within a single module* run **serially** on one process!
- Phoenix had only **7 monolithic test files**:
 - i. `AppWithDefaultsTest` (12 tests)
 - ii. `UmbrellaAppWithDefaultsTest` (10 tests)
 - iii. `AppWithScopesTest` (8 tests)
 - iv. `AppWithMySQLAdapterTest` (6 tests)
 - v. `AppWithMSSQLAdapterTest` (6 tests)
 - vi. `AppWithSQLite3AdapterTest` (6 tests)
 - vii. `AppWithNoOptionsTest` (5 tests)

Famous Last Words...



Rodolfo Carvalho 31.08.26, 16:31

I see your point. With 50+ tests we could try running more than 8 concurrently. I'm not signing up for that right now :) (it takes a lot of time iterating on this, AI or not)



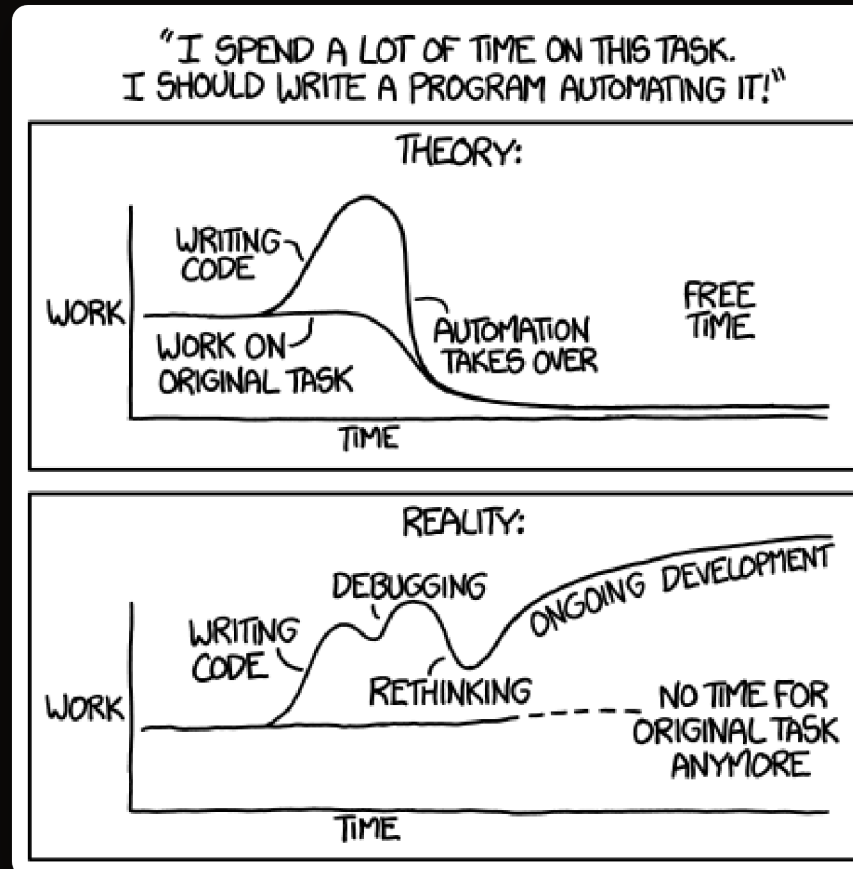
1



August 31: "I'm not signing up for that right now :)"

Spoiler: He went on to sign up for exactly that.

"I'll Just Automate It!"



"Theory: Work on original task permanently reduced. Reality: Rethinking, debugging, ongoing development..." (XKCD #1319)

Stage 2: Breaking the Monolith (PR #6826)

- Split **7 monolithic files** into **33 focused async modules**:
 - `app_with_postgres_adapter_auth_html_test.exs`
 - `app_with_postgres_adapter_auth_live_test.exs`
 - `app_with_postgres_adapter_html_test.exs`
 - `app_with_postgres_adapter_json_test.exs`
 - `app_with_postgres_adapter_live_test.exs`
 - ...and symmetrical counterparts for MySQL, MSSQL, SQLite, & Umbrella!
- **Guarantees:**
 - Preserved all **53 original tests**.
 - Every module designed to execute in **under 3 minutes**.
 - ExUnit now dynamically saturates all runner cores from start to finish!

Trust but Verify: The AST Comparison Script

How do you refactor 53 critical tests across 33 files without missing assertions?

```
defmodule CompareSuites do
  def run do
    main_tests = Enum.flat_map(git_files("upstream/main"), &parse_git_file/1)
    head_tests = Enum.flat_map(git_files("HEAD"), &parse_git_file/1)

    # 1. Parse AST with Code.string_to_quoted!/1
    # 2. Extract describe, test names, and tags
    # 3. Normalize app names: "pg_auth_live" -> "__APP__"
    # 4. Assert byte-for-byte / AST equivalence of every test body!
    match_and_diff_tests(main_tests, head_tests)
  end
end
```

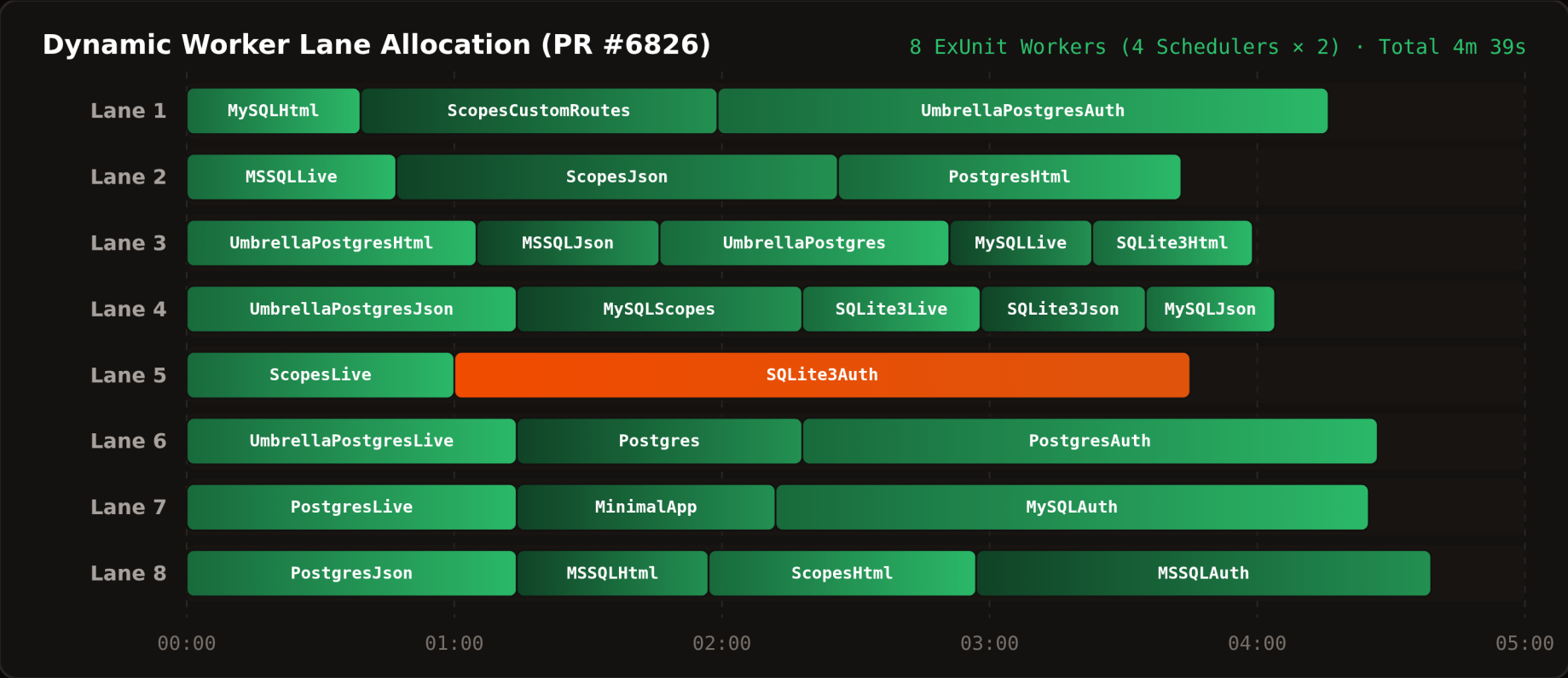
✓ Verified all 53 test bodies were 100% equivalent before opening the PR.

Greedy Interval Scheduling for Gantt Charts

- **Problem:** Rendering 33 individual rows in a Gantt chart produces an unreadable 33-line wall of text.
- **Solution in `SummaryFormatter`:**
 - Implemented **greedy interval scheduling** directly in Elixir!
 - Assigns non-overlapping test modules to compact virtual worker lanes (`Lane 1` to `Lane 8`).
 - Automatically tags the critical path module with `:crit`.

```
defp assign_to_lane([], item, acc), do: Enum.reverse([[item] | acc])
defp assign_to_lane([[{_mod, %{finish: f}} | _] = lane | rest], {_m, %{start: s}} = item, acc) do
  if f <= s, do: Enum.reverse(acc) ++ [[item | lane] | rest],
  else: assign_to_lane(rest, item, [lane | acc])
end
```

Compact Gantt Timeline (Worker Lanes)



All 8 ExUnit worker slots stay saturated across 4 vCPUs. No worker sits idle waiting for a monolithic file.

Infrastructure: Host Runners & tmpfs (PR #6831)

- **Migrated from Alpine Container to Native Ubuntu Runner:**
 - Ran directly on `ubuntu-24.04` host using `erlef/setup-beam`.
 - Pre-installed system build tools (no more `apk add` latency).
 - Removed background `socat` TCP proxy bridges (ports bind directly to `localhost`).
- **Dependency & Build Caching:**
 - Enabled `actions/cache` for `deps/` and `_build/` keyed on `mix.lock`.
 - Saves **~50 seconds** of dependency re-compilation on warm runs!
- **Mounted `installer/tmp` on `tmpfs` (RAM Disk):**
 - Integration tests generate apps, copying dependencies & compiling code repeatedly.
 - Running file I/O entirely in RAM eliminated disk bottlenecks!

Sharding: `--test-partition` vs DB Sharding

Steffen Deusch and I explored two horizontal scaling strategies:

Generic Partitions (PR #6834)

```
mix test --test-partition 4
```

- Great for self-contained unit tests!
- **The flaw for integration tests:** ExUnit hashes test files across partitions.
- Every partition runner gets a random mix of Postgres, MySQL, & MSSQL tests.
- **Every runner must boot all 3 DBs!**

Database Sharding (PR #6836)

```
[postgresql], [mysql], [mssql],  
[none]
```

- Shard strictly by backing database service.
- Containers boot **on demand**:
 - `[sqlite3 + no-db]` boots **0** containers!
 - Postgres runner boots *only* Postgres.
- Mounts DB storage in `tmpfs`.
- Saturates 8 runners across 32 vCPUs!

Cross-Shard Summary Aggregation

- **The Problem:** 8 parallel CI jobs = 8 separate GitHub Step Summaries.
 - Fragmented, cluttered, and hard to spot the overall critical path.
- **The Solution (`aggregate_summary.exs`):**
 - i. Each shard writes a lightweight `summary.json` (a few KB).
 - ii. Uploaded as temporary workflow artifacts.
 - iii. A final aggregator job downloads the JSON files and compiles a unified dashboard:

Elixir/OTP	Job	Status	Tests	Wall Time	Slowest Test
1.20 / 29	Combined	Passed	53	3m 19s	UmbrellaPostgres : auth live (42s)
	PostgreSQL	Passed	24	3m 19s	UmbrellaPostgres : auth live (42s)
	SQLite3 + no-DB	Passed	14	2m 38s	UmbrellaDefaults : defaults (36s)
	MySQL	Passed	8	2m 04s	AppWithMySQL : auth live (38s)
	MSSQL	Passed	7	1m 32s	AppWithMSSQL : auth html (32s)

The Payoff: 10 Minutes → 3m 35s!

Real CI wall-clock metrics from GitHub Actions Run **33897992674**:

10m+

ORIGINAL WALL TIME

3m 35s

OPTIMIZED WALL TIME

~65%

TOTAL TIME SAVED

- Breakdown:

`npm test`: 17s

`installer test`: 36s

`mix test`: 56s

`integration test [mssql]`: 1m 32s

`integration test [mysql]`: 2m 04s

`integration test [sqlite3 + no-db]`: 2m 38s

`integration test [postgresql]`: **3m 19s (Critical Path)**

`integration test results` (aggregator): 12s

AI Pair Programming: Blessing & Curse

Using AI coding assistants (Gemini / Antigravity) throughout this journey:

The Blessing ⚡

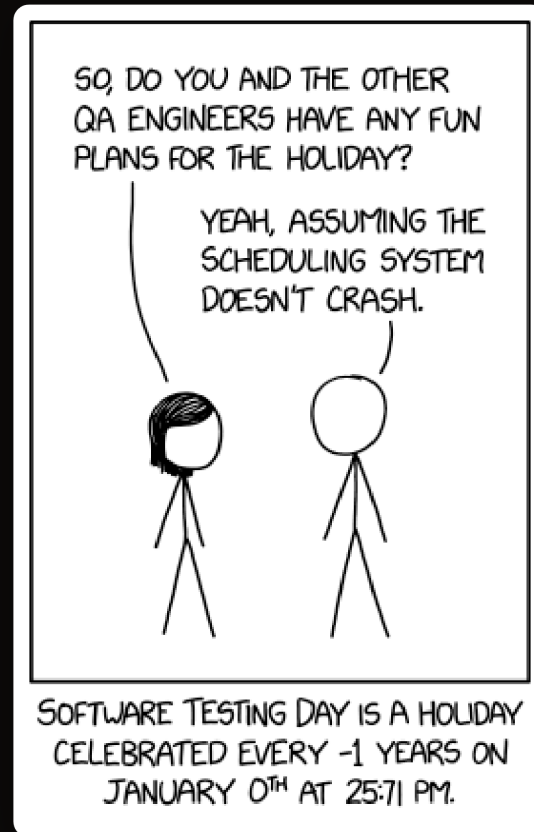
- **Rapid prototyping:** Testing out interval scheduling algorithms in minutes.
- **AST parser generation:** Generating the `CompareSuites` verification script.
- **Git archaeology:** Tracing historical directory leaks back to 2014 commits.

The Curse ⚠

- **Subtle bugs:** Generated app names exceeded default 98-char formatter limits!
- **Phantom edits:** Expanding/contracting words in test descriptions (`it'd` $\rightarrow$ `it would`).
- **Domain socket pitfalls:** Container healthchecks probing Unix sockets instead of TCP.

High speed requires even higher vigilance. "That's the times we live in."

Software Testing Day



"Celebrated every -1 years on January 0th at 25:71 PM." (XKCD #2928)

Key Takeaways

1. Measure Before Optimizing:

Beware of built-in flags that alter runtime semantics (like `mix test --slowest` forcing `--trace`).

2. ExUnit Concurrency Model:

`async: true` parallelizes modules, not tests. Break up monolithic files to keep all CPU cores hot.

3. Sharding with Services:

Generic test partitioning is great for unit tests; service-aware sharding is king when containers are involved.

4. I/O Matters:

Mounting ephemeral test databases and temporary output on `tmpfs` is basically free speed.

5. Observability as Code:

Custom ExUnit formatters and `$GITHUB_STEP_SUMMARY` can turn raw CI dots into transparent engineering dashboards.

Thank You! Questions?

Rodolfo Carvalho

 praialabs.com

GitHub: [@rhcarvalho](https://github.com/rhcarvalho)



praialabs.github.io/optimizing-phoenix-ci

💖 Special thanks to **Steffen Deutsch** for maintaining Phoenix, exploring partition schemes, and reviewing PRs with patience and great insights!

Evolution from 2020 Scaffolding to 2024 Reality

The Origin (2020 - 2021)

- **Late 2020:** Aaron Renner introduces `/integration_test` (PRs #3970, #4027).
- **Jan 2021:** Blog post celebrates **Earthly** for reproducible local & CI builds.
- Containerized build matrix with Docker Compose backing services.

The Drift (2021 - 2024)

- **May 2024:** Earthly dropped by Steffen Deusch (PR #5817) due to local developer friction.
- Replaced with simple `docker.sh` and Docker Compose.
- **Leftover CI debt:** Tests still executed via a monolithic `test.sh` inside an Alpine Docker container.

Local Workflow Ergonomics

CI improvements should **never** make local development painful!

- **Apple Silicon (`arm64`) Auto-Detection in `docker.sh` :**
 - Official Microsoft SQL Server image only supports `amd64`.
 - Detects architecture via `uname -m`; automatically skips MSSQL on ARM64 macs instead of crashing under emulation.
- **Warm Database Containers:**
 - `docker.sh` keeps database containers alive between runs.
 - Re-running tests has **zero container boot latency**!
 - Run `./docker.sh down` when you're done for the day.
- **Non-Root & Port Isolation:**
 - Container runs as non-root user (no root-owned file pollution).
 - Named volumes for native NIFs (`bcrypt` , `argon2`).
 - Database ports bind to loopback (`127.0.0.1`), preventing host port collisions.

Micro-Wins (AI-Induced Tangents)

Eliminating Blind Sleeps (PR #6830)

- Migration versions use 1-second timestamps (`YYYYMMDDHHMMSS`).
- Chained generators (`phx.gen.auth` then `phx.gen.live`) called `Process.sleep(1500)` 8 times to prevent collisions.
- Added `adjust_migration_timestamps/1` :
 - Backdates migrations on disk deterministically into the past.
 - Subsequent generators create fresh timestamps instantly with **zero sleep latency!**

10-Year Directory Leak (PR #6833)

- Consolidated generated app output into `installer/tmp/`.
- **Git archaeology:** Found that `with_installer_tmp/3` had been leaking empty random directories since **2014!**
- Fixed the directory leak, leaving clean disk space for in-memory tmpfs mounts.

Upstream Pull Requests

- **PR #6817**: Add `SummaryFormatter` for integration test visibility in CI
- **PR #6825**: Decompose CI workflow steps & improve container reliability
- **PR #6826**: Split integration test suites into 33 async modules & lane scheduling
- **PR #6830**: Eliminate `Process.sleep` via backdated migration timestamps
- **PR #6831**: Run integration tests on host runner in CI with `actions/cache` & `tmpfs`
- **PR #6833**: Consolidate temporary app generation to `installer/tmp` and fix directory leak
- **PR #6836**: Shard integration tests by database with on-demand container setup in CI
- **PR #6873**: Improve local integration test workflow