A tiered CI strategy for zzcollab research compendia

Ronald ‘Ryy’ G. Thomas

2026-05-06

Figure 1: A layered set of frosted glass panels stacked at increasing depth, each carrying a fainter ink mark, evoking the verification tiers a CI workflow stacks between a commit and a deploy.

Three independent verification tiers, each with its own failure mode and its own definition of ‘passing’.

⚠ Correction – May 2026

Two recommendations in this post were superseded by findings from multi-repo deployment in May 2026. Both are annotated inline below at the relevant sections.

`r-lib/actions/setup-renv@v2` fails on zzcollab research compendia. All zzcollab compendia include an `.Rprofile` gate that suppresses renv activation outside a Docker container (`ZZCOLLAB_CONTAINER=true` must be set). When `setup-renv@v2` calls `renv::restore()` on a host runner, the gate fires, renv declines to activate, and no packages are installed. The CI job either silently passes (nothing installed, nothing verified) or fails with misleading errors. The corrected approach for `r-package.yml` is `r-lib/actions/setup-r-dependencies@v2`, which reads

`DESCRIPTION` directly and never invokes renv.
`renv::status()$synchronized` is an unstable internal field. After a renv version downgrade during `renv::restore()` (for example, from 1.2.3 to 1.2.2), the field returns `FALSE` regardless of actual synchronisation state, producing spurious CI failures. The `quit(status = 1)` guard built on this field should be removed. Relying on `renv::restore()`'s exit code is sufficient.
The current working architecture (`r-package.yml` v2.7.0) is documented in Post 69: zzcollab GitHub Actions workflows.

Introduction

I did not realise how much my CI was lying to me until I added a single explicit failure check and watched five 'passing' projects turn honestly red. The framework's default workflow for the zzcollab ecosystem (a Docker-plus-renv pattern) had been showing green checkmarks across the projects I cared about, and I had been trusting those checkmarks the way one trusts the brake light on a car. The revelation was that some of the brake lights were on green because the bulb was burnt out, not because anything had been verified.

The discovery happened almost by accident. I was migrating one research compendium to a tighter CI pattern (`r-lib/actions/setup-renv@v2` plus an explicit `quit(status = 1)` when `renv::status()` reports inconsistency) and the first run failed. Reading the log I saw a table titled `The following package(s) are in an inconsistent state:` listing dozens of packages installed and used by source code but never recorded in the lockfile. The previous CI had not been catching this; the new check did. When I extended the migration to four more repositories the same shape of finding appeared in three of them, plus two distinct categories of project-side regression that had been similarly invisible.

This post walks through the migration: a tiered CI model with one template per zzcollab workspace type, an explicit synchronisation gate for lockfiles, and a strategy for getting the framework out of the business of reporting fictional health. The companion white paper (mirrored in the zzcollab repository) records the analytic detail; this post is the procedural distillation.

Motivations

- **Green checkmarks were structurally misleading.** A workflow that prints findings without erroring on them looks identical, in GitHub's UI, to a workflow that found nothing wrong.
- **The framework's single CI default conflated four workspace types.** Tool packages, LaTeX manuscript compendia, Quarto compendia, and Quarto blog posts have substantively different CI needs. Applying the same template to all four meant some projects were over-checked (blog posts forced through `R CMD check`) and some were under-checked (compendia whose render was never tested).
- **Lockfile drift was the dominant silent regression.** Three of five projects in the multi-repo verification had `renv.lock` pinning many fewer packages than their source actually used. The previous CI did not detect this; researchers were running in environments their lockfiles could not reproduce.

- **Posit binary identifiers age out of the registry.** Lockfiles that pin specific Posit Package Manager binary builds (the `-N` suffix) eventually fail when those builds are retired from the snapshot. This pattern recurred across two unrelated repos and two different packages (Rcpp 1.1.1-1, S7 0.2.1-1).
- **The default R version is not stable.** `setup-r@v2`'s `r-version: 'release'` resolves to whatever is current; lockfiles pin a specific older version; the mismatch surfaces as opaque compilation errors against the wrong header set.

Objectives

By the end of this migration, the zzcollab project will have:

1. **A workspace-type-appropriate CI template** chosen from the four-row typology. Tool packages get one shape; the three compendium variants get tiered models with the right Tier 3 renderer.
2. **A Tier 1 lockfile-validation job that fails informatively** when `renv.lock` is out of sync with declared dependencies or actually-used source-code references.
3. **R-version pinning sourced from the lockfile** (`r-version: 'renv'`) so the CI environment matches what the project's developers run locally.
4. **Posit Package Manager URLs that resolve to current binaries**, either via date-pinned URLs in `renv.lock` or via the `RENV_CONFIG_REPOS_OVERRIDE` env var that `setup-renv@v2` sets automatically.

Figure 2: A coffee setting in a quiet workspace, the morning's first mug poured before the day's verification work begins. Placeholder imagery; topic-specific replacement pending.

What is the tiered CI model?

A workflow design in which CI is split into independent jobs by verification purpose, each gated on a different cadence and each catching a different class of regression. The architectural analogy is a building's life-safety inspections: the structural inspection fires once at construction, the fire-alarm test fires periodically, and the daily lock-up check fires at every closing. Each is a separate inspection with a separate sign-off.

In zzcollab CI specifically, three tiers cover most needs. Tier 1 (`validate`) verifies that the lockfile, the declared dependencies in `DESCRIPTION`, and the source code's actual imports agree. Tier 2 (`check`) is the conventional `R CMD check`. Tier 3 (`render`) renders the project's primary deliverable document (the manuscript or blog post). Workspace type determines which tiers apply: tool packages skip Tier 3, blog posts skip Tier 2, compendia run all three.

Prerequisites

- A zzcollab project (compendium or tool package) with a populated `renv.lock`, `DESCRIPTION`, and either an `analysis/report/` directory or a top-level `index.qmd`.

- A GitHub repository with Actions enabled.
- The `gh` CLI installed locally for triggering and reading runs.
- R 4.4 or later locally for `renv::snapshot()` and `renv::status()` invocations.
- About one hour of attention per repository for the first migration; subsequent migrations take 10-15 minutes once the patterns are in hand.

Migrating an existing zzcollab project

The migration is the same five steps for every workspace type; only the workflow contents differ. Pick one repository to start with and walk it end to end before applying the pattern to others.

Step 1: Identify the workspace type

Inspect the project's deliverable to choose one of four templates:

- `analysis/report/report.Rmd` exists with `output: pdf_document` in the YAML header: **LaTeX manuscript compendium**.
- `analysis/report/index.qmd` (or top-level `index.qmd`) has `document-type: "blog"` in the YAML header, or the project lives inside a Quarto blog tree: **Quarto blog post**.
- `analysis/report/index.qmd` (or `report.qmd`) without the blog marker, targeting HTML or PDF via Quarto: **Quarto analysis compendium**.
- None of the above; only `R/`, `tests/`, `man/` populated: **tool package**.

File presence alone is unreliable for this decision. zzcollab scaffolds the same R-package skeleton (DESCRIPTION, NAMESPACE, R/, tests/) under every paradigm, so the discriminating signal is the YAML header of the primary document, not the directory structure.

Step 2: Replace the workflow

The four templates are recorded in the white paper and reproduced in `analysis/configs/` in this companion compendium. The shape common to all four is:

```yaml
name: <Workflow name per workspace>

on:
  push:
    branches: [ main, master ]
  pull_request:
    branches: [ main, master ]

jobs:
  validate:
    runs-on: ubuntu-latest
    env:
      GITHUB_PAT: ${{ secrets.GITHUB_TOKEN }}
```

```
    steps:
      - uses: actions/checkout@v4
      - uses: r-lib/actions/setup-r@v2
        with:
          use-public-rspm: true
          r-version: 'renv'
      - uses: r-lib/actions/setup-renv@v2
      - name: Report renv status (fail on inconsistency)
        shell: Rscript {0}
        run: |
          status <- renv::status()
          if (!isTRUE(status$synchronized)) {
            cat("renv reports the project is not synchronized.\n",
                "Run renv::snapshot() locally and commit ",
                "the updated renv.lock.\n",
                sep = "")
            quit(status = 1)
          }
          cat("renv: lockfile, library, and source are synchronized.\n")
```

> ⚠ Correction (May 2026)
>
> `r-lib/actions/setup-renv@v2` does not work on zzcollab research compendia. The zzcollab `.Rprofile` gate suppresses renv outside a Docker container (`ZZCOLLAB_CONTAINER=true`), so `setup-renv@v2` cannot install the lockfile on a host runner. The corrected `r-package.yml` template uses `r-lib/actions/setup-r-dependencies@v2` (reads `DESCRIPTION`, no renv involvement) together with `r-lib/actions/check-r-package@v2`. See Post 69 for the complete v2.7.0 template.

Three points warrant emphasis. First, `r-version: 'renv'` reads the R version from the lockfile rather than defaulting to whatever `release` happens to be. Second, the explicit `quit(status = 1)` on `!isTRUE(status$synchronized)` is what turns the silent print into a visible failure. Third, the workflow does not run a Docker container; `setup-renv@v2` handles renv installation, lockfile restore, and GitHub Actions caching itself. The Docker container remains useful for local development reproducibility but is not needed in CI.

Step 3: Push and observe failures

Commit and push the new workflow. The first run almost certainly fails. This is expected and informative; the failure mode indicates which class of project-side regression is present.

```
git add .github/workflows/
git commit -m "Migrate to new-generation tiered CI"
git push origin main
gh run list --limit 5
```

Examine the failed run with `gh run view <id> --log-failed`. The failures fall into a small number of categories.

Step 4: Address project-side findings

For each failure category, the fix is project-side, not CI-side. Section 'Things to Watch Out For' below catalogues the patterns observed across the multi-repo verification and the action each implies. Once the project-side issues are repaired, the same workflow will pass cleanly.

Step 5: Verify and propagate

Once one repository is green, apply the same workflow template (adjusted for workspace type) to the other repositories in the same project family. The mechanical work shrinks dramatically: copy the YAML, push, observe, fix project-side issues that surface.

Figure 3: A coffee scene mid-pour, the kind of small precision the migration above is asking of CI. Placeholder imagery; topic-specific replacement pending.

Things to Watch Out For

Six recurring patterns to expect during this migration, each with a specific symptom and a specific fix. The first three were observed directly in the five-repository verification; the remaining three are documented in the companion white paper from earlier diagnostic work. None of these are CI-infrastructure noise; all are project-side regressions that the previous CI pattern was masking.

Symptom 1: `renv::status()` lists packages in 'inconsistent state' but the workflow shows

green. This was the founding discovery. `renv::status()` prints findings and returns a list; it does not raise an error. Without an explicit `quit(status = 1)` on `!isTRUE(status$synchronized)`, the workflow exits 0 and GitHub marks the run successful regardless of how much drift the log records. **Fix:** the explicit guard shown in Step 2 above.

> ⚠ Correction (May 2026)
>
> The `renv::status()$synchronized` field is not a stable API. After a renv version downgrade during `renv::restore()` (observed when the lockfile pins renv 1.2.2 but the host runner installs 1.2.3 first, then downgrades), the field returns `FALSE` regardless of actual synchronisation state. The `quit(status = 1)` guard therefore produces spurious CI failures on an otherwise clean project. Relying on `renv::restore()`'s own exit code, and omitting the `renv::status()` check entirely, is the more robust pattern.

Symptom 2: 'package(s) are in an inconsistent state, installed: y, recorded: n, used: y'. The lockfile is incomplete. Source code references packages that the lockfile does not record; those packages are present in the project library because something (usually `setup-renv@v2`'s aggressive transitive resolution, or a prior `renv::install()`) put them there. **Fix:** locally, `renv::snapshot()` to capture the actual library, then commit the updated `renv.lock`. The CI should then go green honestly.

Symptom 3: `Error: failed to install '<package>'`, where the version has a `-N` suffix (`Rcpp 1.1.1-1`, `S7 0.2.1-1`). Posit Package Manager has retired the specific binary build the lockfile pins. **Fix:** locally, refresh the lockfile against the current Posit URL via `renv::snapshot()`, or pin `R$Repositories[0].URL` in `renv.lock` to a date-stamped Posit URL with the OS segment (`__linux__/noble/<date>`) so binaries are reproducible from a fixed snapshot.

Symptom 4: Compilation errors against R headers, e.g. `R_NamespaceRegistry was not declared` or `R_ext/PrtUtil.h: No such file or directory`. The CI is running a different R version than the lockfile pins. The default `r-version: 'release'` resolves to the current release at run time, which drifts forward as new R versions ship. **Fix:** add `r-version: 'renv'` to `setup-r@v2`'s `with:` block. This reads the lockfile and matches the pinned version exactly.

Symptom 5: `R CMD check found WARNINGs` for things like 'Undocumented code objects' or 'Consider adding importFrom'. The previous CI used `error_on = 'error'` in its hand-rolled `rcmdcheck` call, which let WARNINGs pass. `r-lib/actions/check-r-package@v2` errors on WARNINGs by default. **Fix:** address the warnings (documentation gaps, NAMESPACE imports, examples). These are real package-quality regressions; the fact that CI surfaced them is the point.

Symptom 6: A workflow with the right structure but fires on every push, including pushes that change only the manuscript. The render tier is expensive (5-15 minutes) and should not run on every push. **Fix:** path-filter the render workflow to trigger only on changes under `analysis/`, `R/`, `DESCRIPTION`, or `renv.lock`. The split between `r-package.yml` (always-on validation) and `render-report.yml` (path-filtered render) is the standard zzcollab pattern.

Symptom 7 (Docker-anchored compendia only): `renv.lock` after `renv::snapshot()` records macOS binary fingerprints instead of Linux binaries, and CI or a new collaborator's `renv::restore()` installs different package builds than the developer tested. The lockfile was snapshotted on the developer's macOS host. `renv::snapshot()` captures the installed library of the running session; if that session is macOS R, the lockfile records macOS binary hashes that a

Linux runner cannot reproduce. The lockfile then describes an environment that only the original developer on that specific machine can fully reproduce, which violates the core zzcollab reproducibility guarantee.

For Docker-anchored compendia the answer is unambiguous: snapshot inside the Docker container. The reasoning rests on a five-pillar coherence check:

- **Dockerfile** – defines the computational environment (R version, OS, system libraries).
- **`renv.lock`** – must record packages that install cleanly inside that environment, meaning Linux binaries against the container's system libraries.
- **`.Rprofile`** – bootstraps renv on every session start.
- **`R/` and `analysis/`** – the source code.
- **Raw data** – the fixed input.

All five must cohere around the single environment the Dockerfile defines. The Dockerfile is the anchor; the lockfile follows it. A lockfile snapshotted outside the container has slipped its anchor.

Fix: enter the container with `make r`, ensure all required packages are installed (`renv::restore()` if the library is not yet populated), then run `renv::snapshot()`. The lockfile will then record Linux binary fingerprints that match both the container environment and the Ubuntu CI runner.

Consequence for CI design: for a Docker-anchored compendium, running CI inside the container (as the old-generation `rocker/tidyverse`-based workflow does) is the more principled choice, not a legacy pattern to be discarded. The new-generation `setup-renv@v2` pattern trades the container's reproducibility guarantee for CI simplicity – a trade worth making for tool packages and pure-renv projects, but one to evaluate carefully for a compendium whose five pillars are anchored to a Dockerfile. The recommended next step when migrating a Docker-anchored compendium is therefore not to discard the container-based CI but to tighten it: add the explicit `renv::status()` synchronisation guard (Symptom 1) inside the container job rather than switching to a containerless workflow.

Symptom 8 (Docker-anchored compendia with mounted TinyTeX): `sh: 1: xelatex: Permission denied` even after `chmod -R a+rx $HOME/.TinyTeX` on the host runner. This one has two compounding causes that must be understood separately.

Cause A: `chmod -R` on Linux does not follow file symlinks. In a TinyTeX installation, `~/.TinyTeX/bin/x86_64-linux/xelatex` is a symlink pointing into `~/.TinyTeX/texmf-dist/bin/x86_64-linux/`. Running `chmod -R a+rx $HOME/.TinyTeX/bin` changes the mode of the symlinks themselves, not the targets. The actual ELF binaries in `texmf-dist/` may remain non-executable. Expanding the scope to `chmod -R a+rx $HOME/.TinyTeX` does recurse into `texmf-dist/` (because `chmod -R` does follow directory symlinks), so it addresses Cause A – but Cause B still blocks execution.

Cause B: `/root` itself is `700`. The strategy of mounting TinyTeX into the container as `-v $HOME/.TinyTeX:/root/.TinyTeX` places the tree under `/root` inside the container. A zzcollab Dockerfile ends with `USER analyst` (or equivalent non-root username), so the container process runs without root privileges. The Linux kernel checks directory execute-permission at each component of a path; `/root` being `700` means that even if every file inside `/root/.TinyTeX/` has world-execute permission, the non-root user cannot traverse into it. The error `sh: 1: xelatex: Permission denied` reflects the kernel blocking path traversal, not a missing execute bit on the binary.

Fix: mount TinyTeX to a world-accessible path rather than under `/root`. Replace the `-v` and `-e PATH` arguments in the `docker run` call:

```
# Before (fails for non-root container user)
-v $HOME/.TinyTeX:/root/.TinyTeX \
-e PATH="/root/.TinyTeX/bin/x86_64-linux:$PATH" \

# After (world-accessible mount point)
-v $HOME/.TinyTeX:/opt/tinytex \
-e PATH="/opt/tinytex/bin/x86_64-linux:$PATH" \
```

`/opt` is owned by root with permissions `755`, so all users can traverse it. The `chmod -R a+rx $HOME/.TinyTeX` step on the host runner is still needed to ensure the files themselves are world-readable and executable. The two lines together – chmod on the host, mount to `/opt/tinytex` in the container – resolve both causes.

Symptom 9 (cross-repo portability): A `render-report.yml` that works for one compendium fails for another because one uses `rocker/verse` (TinyTeX built-in, runs as root) and the other uses `rocker/tidyverse` (no LaTeX, non-root user). Projects in the same portfolio may have different Dockerfile base images. Applying a workflow that was fixed for one base to another without adjustment reproduces the original failure.

The `/opt/tinytex` host-mount pattern from Symptom 8 generalises to both cases and can serve as the single standard template across all compendium repos:

- *`rocker/verse`-based repos (root user):* The image already ships TinyTeX at `/root/.TinyTeX` with its bin directory on PATH. The host mount at `/opt/tinytex` is redundant but harmless: `xelatex` is found in the container's own PATH entry before the explicit `-e PATH` prefix is reached. Both LaTeX sources exist simultaneously; the container's built-in takes precedence.
- *`rocker/tidyverse`-based repos (non-root user):* The image has no LaTeX. The host mount at `/opt/tinytex` is the only source of `xelatex`. Because `/opt` is world-traversable (`755`), the non-root user can access the binaries.

The three-step block that achieves this generalisation:

```yaml
- name: Install TinyTeX on host runner
  uses: r-lib/actions/setup-tinytex@v2

- name: Make TinyTeX binaries accessible to container
  run: chmod -R a+rx $HOME/.TinyTeX

- name: Restore packages and render all manuscripts
  run: |
    docker run --rm -e CI=true \
      -v ${{ github.workspace }}:/project \
      -v $HOME/.TinyTeX:/opt/tinytex \
      -e PATH="/opt/tinytex/bin/x86_64-linux:$PATH" \
      -w /project compendium-env \
      Rscript -e '...'
```

Fix: adopt this three-step block as the standard `render-report.yml` template across all Docker-anchored compendium repos, regardless of Dockerfile base image. The `rocker/verse` case incurs

a small overhead (downloading TinyTeX on the host runner, ~30 seconds) that is acceptable in exchange for a single maintainable template.

Symptom 10 (ggplot2/patchwork version skew): `object 'is_ggplot' is not exported by 'namespace:ggplot2'` during patchwork installation, even when patchwork appears to be a current CRAN release. This is a compound failure that surfaces when the Docker base image ships ggplot2 3.5.x and the renv lockfile (or the PPM snapshot) pulls patchwork 1.3.2 or later.

`patchwork 1.3.2` added `@importFrom ggplot2 is_ggplot` to its NAMESPACE – it imports `is_ggplot` as an exported symbol from ggplot2. `ggplot2 3.5.1` (which ships with `rocker/verse:4.4.2`) does not export `is_ggplot`; the function was introduced as an export in ggplot2 4.0.0. Installing patchwork 1.3.2 into a container that has ggplot2 3.5.1 therefore fails at the 'prepare package for lazy loading' step with the export error.

The failure looks like it belongs to zzlongplot or the project's own code because the error appears during zzlongplot's installation (which imports patchwork). But the root cause is one level deeper: patchwork itself cannot load against ggplot2 3.5.1.

Fix: update ggplot2 to 4.0.x before (or alongside) installing patchwork. In the lockfile-rebuild script:

```
pkgs <- c("ggplot2", "patchwork", ...)  # include ggplot2 explicitly
install.packages(pkgs,
  repos = "https://packagemanager.posit.co/cran/__linux__/noble/latest")
```

ggplot2 4.0.x pulls in `S7` as a new dependency – add `S7` to the lockfile as part of the same rebuild. After this update, patchwork 1.3.2 loads cleanly and any package that depends on patchwork (such as zzlongplot) can be installed.

Portfolio implication: any zzcollab compendium that uses patchwork and is based on `rocker/verse:4.4.2` (which ships ggplot2 3.5.1) will hit this failure when the lockfile is rebuilt against the current PPM snapshot. The fix – updating ggplot2 to 4.x – must be applied consistently across all affected repos during lockfile refresh.

Uninstall / Rollback

The migration is fully reversible. The previous workflow lives in git history; restoring it requires checking out the prior version of `.github/workflows/r-package.yml` (and any companion files) from the commit before the migration:

```
git log --oneline -- .github/workflows/r-package.yml
git checkout <prior-sha> -- .github/workflows/r-package.yml
git commit -m "Revert to prior CI workflow"
git push
```

If only the strict synchronisation check is the source of friction (for example, during a transitional period while several lockfiles are being repaired in parallel), the strict guard can be loosened to

informational reporting by removing the `quit(status = 1)` line. The rest of the new generation pattern remains in place.

Figure 4: An empty cup beside a closed notebook, the morning's verification work done and the cup ready for the next pour. Placeholder imagery; topic-specific replacement pending.

What did we learn?

Conceptual

- **CI checks for regressions, and different regressions need different checks.** A workflow that runs `R CMD check` is necessary for a tool package and useless for a Quarto blog post whose deliverable is rendered HTML. The framework's purpose determines the appropriate verification, not the framework's default.
- **A green dashboard with red findings in logs is operationally indistinguishable from an absent CI check.** The visible signal is the job exit status. A check that prints evidence of regression but exits 0 is a false sense of safety, not a check.
- **File presence is a noisy signal for workspace-type detection.** zzcollab scaffolds the same R-package skeleton across all paradigms; the discriminating signal lives in the document YAML header (`document-type: "blog"`, `output: pdf_document`).
- **The two-workflow split is an existing zzcollab pattern that should be propagated uniformly.** The framework already ships `r-package.yml` and `render-report.yml` templates; three of eighteen surveyed compendium repos have both deployed. The contribution of this migration is uniform deployment, not a new design.

Technical

- **`r-lib/actions/setup-renv@v2`** handles renv installation, lockfile restore, GitHub Actions caching, and the `RENV_CONFIG_REPOS_OVERRIDE` env var in roughly twenty lines of YAML. It is the canonical R-package CI bootstrap; the Docker-in-CI pattern adds complexity that this action already resolves.
- **`r-version: 'renv'`** on `setup-r@v2` reads the R version from the lockfile. Without it, the workflow's R version drifts forward with each new R release while the project's pinned version stays put.
- **`renv::status()` returns a list with `synchronized = TRUE/FALSE`.** Capture it explicitly and `quit(status = 1)` on inconsistency. The job's exit status is what GitHub reads; nothing else matters.
- **`r-lib/actions/check-r-package@v2`** errors on WARNINGs by default. This is stricter than a hand-rolled `rcmdcheck::rcmdcheck(error_on = 'error')`. The strictness is a feature: it surfaces real package-quality regressions.

Gotchas

- **The default `r-version` drifts.** `setup-r@v2` resolves `release` at run time, so a workflow that worked yesterday can fail tomorrow when R 4.x.y is released. Pin to the lockfile.
- **Posit binary identifiers are not stable.** A lockfile pinning `Rcpp 1.1.1-1` will eventually fail when Posit retires that build. Prefer date-pinned URLs.
- **The `.Rprofile` auto-restore can race the workflow.** The zzcollab template fires `renv::restore()` on R session start; the workflow also calls restore. Set `ZZCOLLAB_AUTO_RESTORE=false` in the workflow `env:` to give control to the workflow.

- **Stale `render-report.yml` variants exist in deployed repos.** The `templates/.github/` tree (since deleted from canonical zzcollab) shipped a hardcoded `report.Rmd` matcher; projects that received that variant fail silently for any project using `manuscript.Rmd`. Re-template via `zzc doctor`.

Limitations

- **Project-side fixes are still required.** The new CI surfaces drift, missing documentation, and aged-out binaries; it does not repair them. Each surfaced issue requires a developer to address it.
- **Posit snapshot dates expire.** Date-pinned URLs eventually age out (Posit retains snapshots for roughly three to five years). Long-lived projects need periodic snapshot refreshes.
- **Detection at scaffolding time is not yet automated.** The decision among the four workspace templates currently requires a developer to inspect the project and pick. A future enhancement to `zzc analysis` or `zzc doctor` could automate this.
- **The `renv::status()` synchronisation check assumes `renv.lock` is the source of truth.** Projects that intentionally maintain a smaller lockfile (for example, a package that lists only its declared `Imports` and trusts the user's environment for everything else) will fail the synchronisation check by design. The check is appropriate for research compendia; it may not be for some other patterns.
- **The Docker container is no longer exercised by CI.** Projects that rely on the Dockerfile for production deployment should consider adding a separate Docker-build verification job.

Opportunities for Improvement

1. **Auto-detect workspace type at `zzc analysis` time.** Inspect the YAML headers and emit the appropriate workflow template in `.github/workflows/` automatically. This removes the manual decision in Step 1 of the migration.
2. **Date-pin the lockfile URL at init time.** When `zzc` initialises a project, write `R$Repositories[0].URL` as `https://packagemanager.posit.co/cran/__linux__/noble/<today>` rather than the unqualified source URL the framework currently emits.
3. **`zzc doctor` propagation.** Extend the doctor command's full-content workflow replacement (introduced earlier in this work) to detect workspace type and replace stale `render-report.yml` variants with the canonical `.qmd`-aware version.
4. **Output verification.** Hash the rendered manuscript and fail Tier 3 if the hash differs from a recorded baseline. Catches regressions in figures and tables that pass `R CMD check` but produce different rendered output.
5. **Schedule-driven freshness checks.** Add a weekly cron job that re-runs Tier 1 against the current Posit URL, surfacing binary aging-out before it blocks an active push.
6. **Documentation pass.** The four workflow templates and the migration procedure deserve to be embedded in the canonical zzcollab user guide alongside the existing `CI_WORKFLOW_ARCHITECTURE.md` and `ci-workflows-whitepaper.md` documents.

Wrapping Up

The migration's core contribution is making CI honest. The previous green checkmarks across these projects were structurally misleading; they reflected the absence of an error rather than the presence of verification. The new pattern's first response on most projects is to turn red, but the red is informative and actionable, and the resulting fixes are real improvements to the projects' reproducibility.

The pattern is not a new design. The two-workflow split (`r-package.yml` plus `render-report.yml`), the use of `r-lib/actions/setup-renv@v2`, and the path-filtering on the render workflow all already exist in zzcollab's template set, applied inconsistently across deployed projects. The migration described here is the explicit deployment of those existing patterns uniformly, with the addition of a single explicit synchronisation guard that the framework's templates had been missing.

The companion white paper records the analytic detail behind the choices made here, including the four-workspace typology, the verification of the synchronisation check at scale, and the classification of failure modes observed during the multi-repo deployment. Readers who want to understand why the templates have the shapes they do should consult that document; readers who want to apply the templates should follow this post.

In conclusion, four points merit emphasis. First, a CI check that prints findings without erroring is operationally invisible; the job's exit status is the only signal GitHub shows. Second, four zzcollab workspace types map to four distinct CI templates (tool package, LaTeX compendium, Quarto compendium, Quarto blog post), and the document YAML header is the discriminating signal, not file presence. Third, `r-lib/actions/setup-renv@v2` plus `r-version: 'renv'` plus a one-line `quit(status = 1)` guard on `renv::status()$synchronized` covers the bulk of what an always-on validation tier needs. Fourth, failures from the new pattern point to real project-side regressions (lockfile drift, aged-out Posit binaries, R-version mismatch, missing package documentation); none are CI-infrastructure noise.

> ⚠ Correction to Point 3 (May 2026)
>
> Point 3 above reflects the intent of this post at time of writing, but both mechanisms named have since proved unreliable for zzcollab compendia. `setup-renv@v2` cannot install packages on a host runner because the zzcollab `.Rprofile` gate suppresses renv outside the Docker container. `renv::status()$synchronized` misfires after renv version downgrades, producing spurious failures. The corrected `r-package.yml` (v2.7.0) uses `setup-r-dependencies@v2` to install from `DESCRIPTION` and `check-r-package@v2` for validation; no renv involvement on the host runner at all. Full template and rationale in Post 69. The four-workspace typology (Point 2) and the exit-status principle (Point 1) remain sound.

See Also

- The companion white paper in this compendium, which records the analytic detail behind the choices in this post.
- The same white paper mirrored in zzcollab itself at `~/prj/sfw/07-zzcollab/zzcollab/docs/ci-strategy-tiered-model.md`, for ongoing co-evolution.

- The renv documentation on continuous integration, which describes the canonical `r-lib/actions/setup-renv@v2` pattern this work builds on.
- The r-lib/actions repository, which documents `setup-r`, `setup-renv`, `setup-pandoc`, `setup-tinytex`, and `check-r-package`.
- Related blog posts in this series: Post 14: penguins1zzcollab (the worked example whose CI verification produced Section 9.7 of the white paper); Post 61: zzcollab analysis checklist (the closest-in-time companion post on zzcollab procedure).

Reproducibility

Tested on the configurations below.

Component	Version	Notes
OS (CI runner)	Ubuntu 24.04 (noble)	GitHub Actions ubuntu-latest as of May 2026
OS (local)	macOS 15 (Darwin 25)	for `gh`, `git`, local renv operations
R	4.4.2	matches lockfile pins across surveyed projects
`r-lib/actions/setup-r`	v2	with `r-version: 'renv'`
`r-lib/actions/setup-renv`	v2	handles caching automatically
`r-lib/actions/setup-pandoc`	v2	needed for Tier 2 and Tier 3
`r-lib/actions/setup-tinytex`	v2	needed for LaTeX render tier
`r-lib/actions/check-r-package`	v2	errors on WARNINGs by default
`quarto-dev/quarto-actions/setup`	v2	needed for Quarto render tier
Date of last verification	2026-05-06	five-repo deployment, all surfacing project-side findings

The companion compendium ships `analysis/configs/r-package.yml`, `analysis/configs/render-report.yml`, and `analysis/configs/blog-render.yml` as the four templates referenced in Step 2. Copy the appropriate one to `.github/workflows/` in the project.

Let's Connect

Comments and questions on this post are welcome via the giscus panel below. Corrections, alternative patterns, and reports of edge cases that the four-workspace typology does not cover are particularly useful: the typology was extracted from a sample of roughly twenty projects in two trees, and the natural next step is to test it against external projects whose patterns differ.

The companion white paper at `zzcollab/docs/ci-strategy-tiered-model.md` is the live document; it will be updated with new findings as they accumulate.

Rendered on 2026-05-22 at 09:22 PDT. Source: ~/Dropbox/prj/qblog/posts/63-zzcollab-ci-strategy/zzcollab-ci-strategy/analysis/report/index.qmd

Related posts in this cluster

This post is part of the *ZZCOLLAB Reproducible Compendia* series. Recommended reading order:

1. Post 01: Reproducible Blog Posts with ZZCOLLAB
2. Post 02: Constructing a reproducible blog post using zzcollab tools
3. Post 03: From Markdown to Blog Post: A ZZCOLLAB workflow
4. Post 04: Sharing R Code via Docker: R Markdown Reports
5. Post 05: A 55-Item Initiation Checklist for zzcollab Data Analyses
6. Post 06: Seven Required Elements for a zzc Manuscript report.Rmd
7. **Post 07: A tiered CI strategy for zzcollab research compendia** (this post)
8. Post 08: GitHub Actions workflows for zzcollab research compendia