PharmaSUG 2026 - Paper ET-292

Engineering Secure and Reproducible R based Clinical Programming Systems using Open Source DevSecOps Practices

Indraneel Chakraborty, Ephicacy Lifescience Analytics Pvt Ltd, Bengaluru, India

ABSTRACT

R is increasingly being adopted in clinical programming, evolving from isolated scripts into reusable functions, packages, and automated pipelines that generate analysis-ready datasets and TLF outputs. As reuse expands across studies and teams, risks emerge, including accidental exposure of credentials, result drift across machines, dependency changes that alter outputs or introduce vulnerabilities, automation failures in non-interactive environments, and unintended leakage of sensitive data through logs or artifacts.

This paper presents an engineering roadmap for moving from scripts to packages to pipelines while making security, auditability, and reproducibility inherent qualities of the system. We emphasize platform-neutral DevSecOps practices that support clinical traceability and consistent execution, including git-based version history, automated quality gates such as tests and style checks, and standardized run configurations that behave reliably across laptops, CI systems, and scheduling platforms. Reproducibility is framed as an end-to-end runtime property rather than only dependency management. We compare approaches for capturing and restoring project dependencies to maintain analytical stability, alongside Dockerfile-based containers that bundle OS libraries, R, and system components to ensure consistent behavior in controlled batch environments, enabling durable reruns months later. Security considerations span the full lifecycle, covering separation of secrets from code, least-privilege access, reduced network exposure in automation, dependency health monitoring, and generating outputs with evidence linking deliverables to exact code versions, configurations, and execution contexts. A brief discussion on LLM-related risks highlights potential leakage pathways. Readers will gain practical, transferable patterns for building secure, reproducible, and maintainable R-based clinical programming systems that teams can confidently rely on.

INTRODUCTION

The DevSecOps framework is organized around three core areas: development, security, and operations. It can be applied to R clinical programming systems with the help of open-source utilities. Each area corresponds to a distinct phase of the clinical programming lifecycle, with specific tools recommended by the pharmaverse, tidyverse, and R Validation Hub communities. The architecture flow in Figure 1 below illustrates how these three layers integrate and prove how every output is linked to version-controlled code and a reproducible runtime environment.

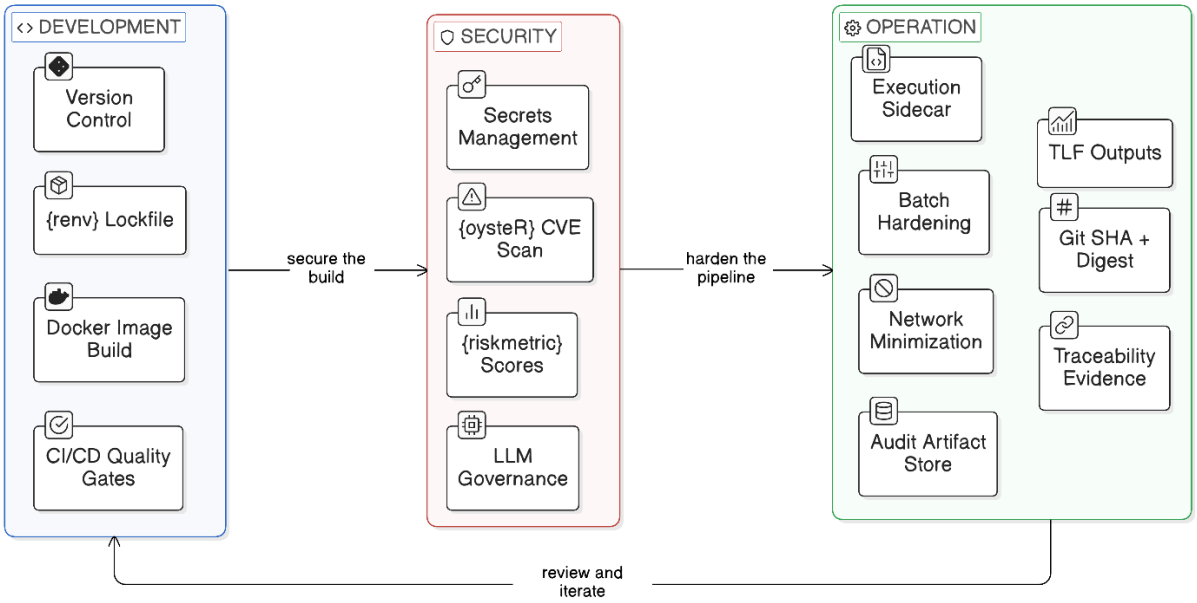

Figure 1: Three-layer DevSecOps Architecture

The sections below unpack each layer with concrete patterns, tool recommendations, and code examples. Together they make security, auditability, and reproducibility inherent system properties rather than afterthoughts.

COMMON RISKS IN EXPANDING R WORKFLOWS

Clinical R workflows that scale beyond a single study face a predictable set of failure modes. Understanding them is the first step toward building systems that avoid them.

1. CREDENTIAL EXPOSURE

Scripts that embed database passwords or API tokens directly in source files function without issue in isolated development environments. Once a script is committed to a shared repository, the credential is effectively public. Git history is permanent within a repository where even a delete-and-commit makes the secret visible to anyone who checks out an earlier revision. In regulated environments where repositories are shared across teams, sponsors, or CROs, this is a significant compliance exposure.

2. REPRODUCIBILITY DRIFT

R packages update continuously on CRAN. A pipeline that produces valid results today may fail or produce different numeric outputs after a routine dependency upgrade. In a GxP context, result drifting across environments or time points is not just inconvenient, it can invalidate a submission. The discrepancy between a pipeline's behavior on a developer workstation and its behavior in a CI system months later is the most common source of reproducibility failure in regulated R environments.

3. AUTOMATION FAILURES IN NON-INTERACTIVE ENVIRONMENTS

Code written for interactive use routinely depends on environment variables, file paths, or session state that are absent in a headless batch job. Interactive prompts block indefinitely. Assumed home directories may be unavailable. Credentials loaded manually in an *.Rprofile* are absent. Pipelines that have not been confirmed outside an interactive development environment often fail during production execution, often at critical project milestones.

4. DATA LEAKAGE

Sensitive clinical data can escape through log files capturing patient identifiers, artifact directories inadvertently including raw datasets, and external tools receiving inputs without adequate access controls. The growing use of LLMs as coding assistants is an emerging and insufficiently governed leakage surface.

Patient data or proprietary analysis logic passed into a commercial LLM prompt exits the controlled environment entirely and may be kept for model training.

5. OPEN-SOURCE VULNERABILITIES

A common assumption holds that open-source ecosystems are inherently secure by virtue of their transparency. Recent events in the R ecosystem show that openness does not drop security risks. In April 2024, a high-severity deserialization vulnerability (CVE-2024-27322) was discovered in R, affecting versions 1.4.0 up to but not including 4.4.0. This flaw allows attackers to execute arbitrary code through malicious RDS, RDX, or package data. The official CRAN repository hosts over 23,000 packages and the submission process focuses on correctness and policy compliance, not deep security audits. Teams must employ supplementary dependency vulnerability scanning, such as the {***oysteR***} package, especially when deploying R in production or regulated environments.

The following flow-diagram summarizes the five risk categories described above, illustrating how each failure mode manifests as R workflows expand beyond a single study or team -

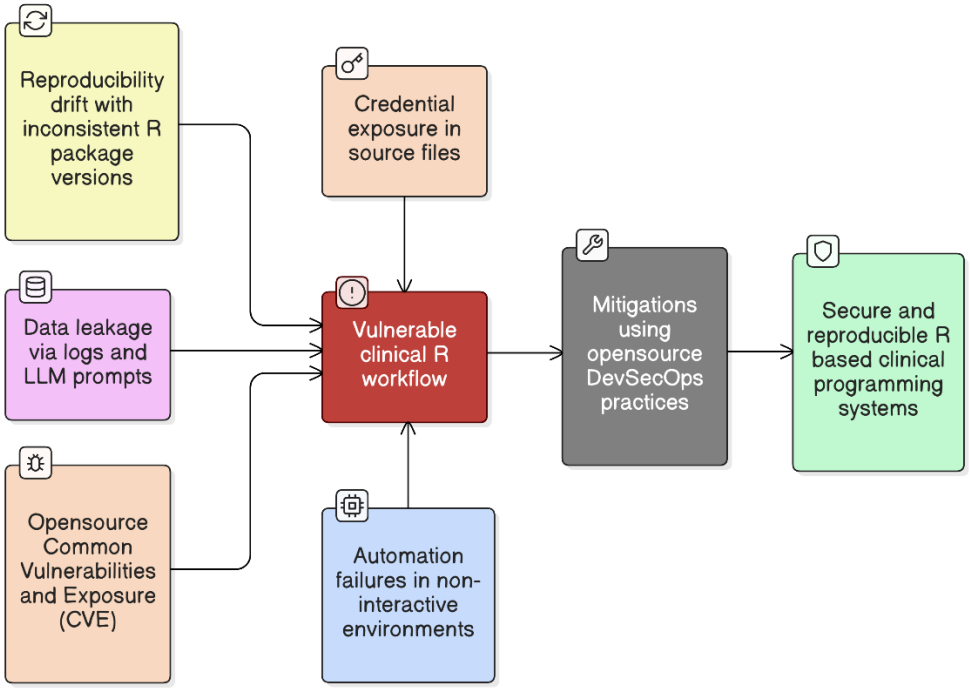

Figure 2: Common Risks in Expanding R Workflows

DEVELOPMENT: CONTROLLED ENVIRONMENT, REPRODUCIBLE BUILDS, AUTOMATED QUALITY GATES

The development layer covers version control, dependency locking, containerization, and automated quality enforcement. In a GxP context, these are not optional engineering choices. They are the foundation of an auditable system.

1. VERSION CONTROL AND TRACEABILITY

Every development action must be traceable to an exact code revision. Git is the foundation. The key discipline is not the tool itself but the workflow around it. Some key practices to achieve this include:

- Use a primary production branch for submission-ready code and a separate development branch for ongoing work.
- Create short-lived feature branches for specific tasks and merge back after peer review.
- Tag each regulatory submission with a unique identifier so the exact code revision used can always be recovered.

- Record the commit secure hash algorithm (SHA) that produced a deliverable in output metadata.
- Use pull requests as the code review gate: they create the auditable record of who reviewed what, when, and what comments were made.

The pharmaverse guidelines across packages such as {***admiral***} show practical examples of PR review, labeling, and changelog conventions that balance speed with traceability.

2. DEPENDENCY LOCKING WITH {RENV}

The {***renv***} package records the exact versions of all installed packages in a "*renv.lock*" file, committed alongside the code. Key practices include:

- Run *renv::restore()* to recreate the exact package setup on any machine, regardless of CRAN updates.
- Combine the "*renv.lock*" file with a git tag to create a point-in-time specification of code and packages.
- Run *renv::snapshot()* after any package change and commit the updated lockfile at once.
- Audit for known vulnerabilities using {***oysteR***}, which queries the Sonatype OSS Index. {***rosv***} package is also often used which checks for known open source vulnerabilities against the free OSV database.

The following flowchart includes the key practices that can be followed in a pharma context while working with the {***renv***} package -

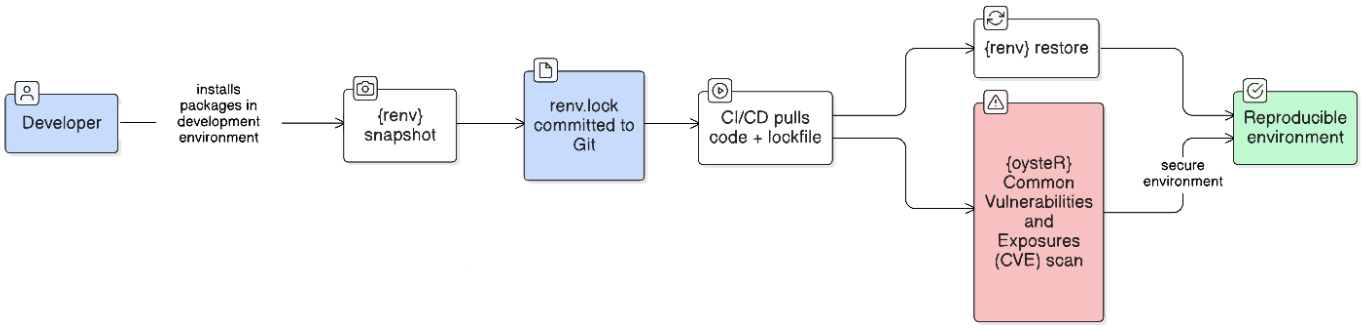

Figure 3: {*renv*} Dependency Locking Workflow

3. CONTAINERIZATION FOR DURABLE RERUNS

Lockfiles manage R packages well, but not the R interpreter, compiled system libraries such as *libssl* and *libxml2*, or OS-level locale settings. For regulatory reruns, Docker captures these into an immutable runtime. *Rocker* versioned base images (*rocker/r-ver*) pin both R and the underlying Debian release. The image tag plus "*renv.lock*" becomes a complete runtime specification.

To ensure reproducibility, a production-ready container image should:

- Start from a pinned base image specifying both the R version and OS release.
- Install all required system libraries explicitly to support compiled R packages.
- Restore the R package library from "*renv.lock*" during the image build, not at runtime.
- Copy the project code and run the pipeline as a non-root user with a single, deterministic entrypoint.
- Record the image digest alongside the Git SHA in all output metadata.

Build the image at study lock and push it to a private registry under change control. For reruns, pull the same tagged image and run against archived data. The OS, system libraries, R, and R packages are unchanged. For multi-stage pipelines, the {***targets***} package provides dependency tracking and incremental parallel execution inside the container.

The FDA does not mandate Docker specifically. It actively supports containerization as a pathway to reproducibility, and the FDA's openFDA codebase uses docker compose publicly, showing practical adoption.

4. AUTOMATED QUALITY GATES AND CI/CD

Automated quality enforcement with continuous integration and continuous deployment (CI/CD) ensures that every code change is rigorously checked before human review. These can be triggered to run on every push or pull request. Following are few R packages that can be used in these automated checks:

- {***testthat***}: unit tests covering derivation logic, edge cases, and output structure; snapshot testing captures expected results and flags deviations.
- {***lintr***}: style checks and static analysis enforcing coding standards across teams via shared configuration.
- {***covr***}: code coverage measurement; enforce a minimum threshold before any merge is allowed.
- {***rcmdcheck***} and {***goodpractice***}: comprehensive package checks and complexity hygiene.
- {***diffdf***}: data frame comparison to detect unexpected output changes between runs.

All quality gates must execute within the same Docker image used in production to prevent environment-driven mismatches. Platforms such as GitHub Actions, GitLab CI, and Jenkins automate these checks and block merges on failure. CI logs and artifacts from every run form a complete, time-stamped audit trail.

The following flow-diagram depicts the CI/CD quality gate pipeline, showing how automated checks are sequenced on each pull request before code reaches the production branch -

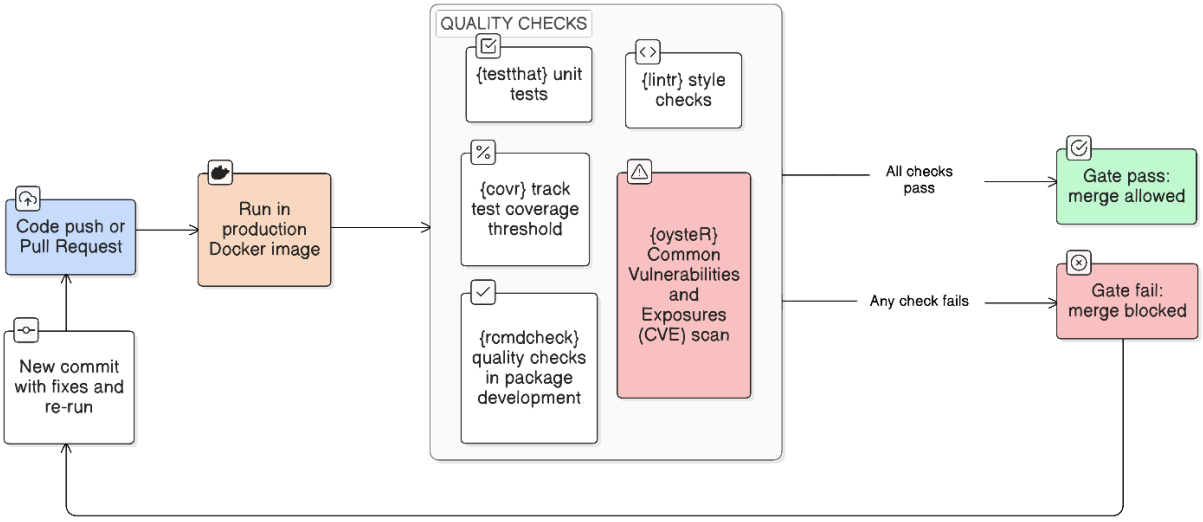

Figure 4: CI/CD Quality Gate Pipeline

SECURITY: OPEN-SOURCE VULNERABILITY MANAGEMENT AND SECRETS HYGIENE

The security layer protects credentials, assesses packages for vulnerabilities and quality, and addresses newer risks from LLM-assisted development.

1. SECRETS MANAGEMENT: NEVER EXPOSE IN CODE

Credentials, tokens, and keys must never appear in code or version-controlled configuration files. These practices can be implemented:

- Inject secrets at runtime via CI/CD platform secret stores such as GitHub Secrets, GitLab CI variables, or HashiCorp Vault.
- Grant credentials the minimum permissions needed and set short expiry windows.
- In R, store secrets in the "*.Renviron*" file and access them at runtime using the "*Sys.getenv()*" without needing to hardcode values in code.
- Use pre-commit hooks or tools such as git-secrets to block accidental credential commits.

2. OPEN-SOURCE VULNERABILITY SCANNING

Trusting CRAN acceptance as a security gate creates a false sense of assurance. The following controls can be layered in every regulated R environment:

- {***oysteR***} scans R packages against the Sonatype OSS Index CVE database; integrate directly into CI/CD so every code change triggers a vulnerability check.
- {***rosv***} fetches information from the OSV database, which is a free, community-backed vulnerability feed with an API that supports automated pipeline lookups.
- {***riskmetric***} and {***riskassessment***} evaluate package quality across testing practices, documentation completeness, maintenance activity, and community support; {***riskassessment***} also provides a centralized place to document package reviews and organizational approval decisions, supporting GxP software validation.
- Container image vulnerability scanning tools such as Trivy or Grype scan the full software stack including system libraries; pair with software bill of materials (SBOM) generation to list all components for regulators.
- Base image pinning use immutable digest references for container base images to ensure deployment consistency.

3. LLM LEAKAGE RISKS AND SAFE USAGE PATTERNS

LLMs are a newer and underappreciated leakage surface in clinical workflows. Patient data or proprietary analysis logic pasted into a commercial LLM prompt exits the controlled environment entirely. The risks, however, differ materially depending on deployment model. Commercial cloud-based APIs present the highest leakage exposure, as prompts may be logged or kept by the provider. Locally hosted open-weight models such as those deployed via *Ollama* or *vLLM* on organization-controlled infrastructure substantially reduce this exposure, as data does not leave the secure environment. The following practices apply across both deployment types but are especially critical for commercial API usage:

- Use synthetic or anonymized datasets for LLM-assisted development. The pharmaverse includes some R packages that provide test datasets for use.
- Manage LLM API keys via environment variables accessed with "*Sys.getenv()*"; never hardcode them in exposed codes or snippets.
- Log all LLM interactions using {***logger***} or {***logrx***}, ensuring that logs hold no protected data.
- Establish clear data processing agreements with LLM providers before organizational adoption.
- Prefer locally hosted open-weight models for tasks that require access to study-level data; reserve commercial APIs for tasks involving only synthetic or fully de-identified inputs.

OPERATIONS: EXECUTION TRACEABILITY, BATCH HARDENING & GXP ALIGNMENT

The Operations layer covers how pipelines run in production, what evidence they generate, and how that evidence maps to GxP and regulatory expectations. A pipeline that produces correct outputs but cannot prove it produced them is not submission ready.

1. EXECUTION METADATA AND AUDIT ARTIFACTS

For each pipeline run in a clinical setting, automatically collect and save execution context as a sidecar file alongside outputs. Following information should be recorded to begin with:

- Git commit SHA of the code version executed.
- Hash of the configuration file used, to verify settings were unchanged.
- R version and complete "*renv.lock"* content.
- User, host, and UTC timestamp.
- Docker image digest when running inside a container.

The {***digest***} package supports file hashing, {***jsonlite***} enables portable JSON metadata, and {***sessioninfo***} provides structured session details. For clinical workflows, {***logrx***} from pharmaverse logs

session details, input and output files, and any errors or warnings in a structured file persisted alongside pipeline outputs. Figure 5 illustrates the recommended structure of the execution artifact store, showing how sidecar files are organized alongside pipeline outputs for each production run.

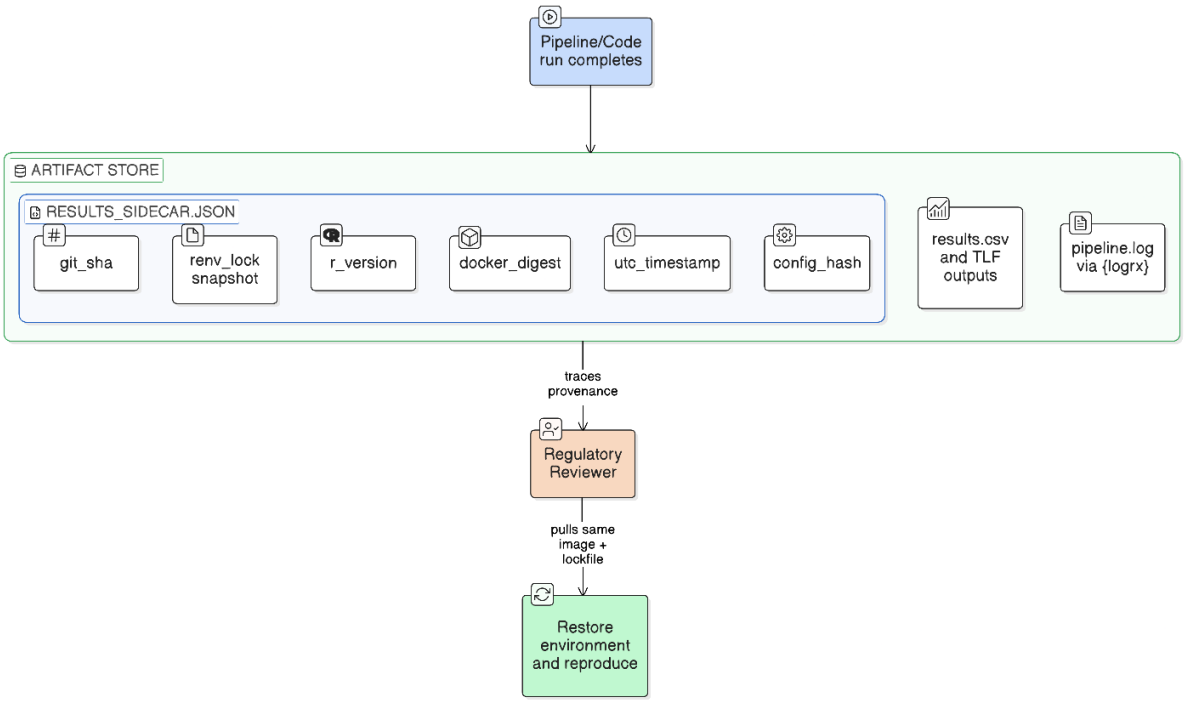

Figure 5: Execution Artifact Store Structure

2. CONFIGURATION AS CODE

Pipeline parameters, data paths, population flags, and output requirements must be separated from R source code using version-controlled configuration files in YAML or JSON format:

- Use the {***config***} package for a single YAML file with environment-specific overrides for development, testing, and production environments.
- Record the configuration file hash in pipeline metadata; any config change creates a new hash, ensuring full auditability.
- Use {***envsetup***} from pharmaverse to manage and document execution environments in GxP settings.

3. BATCH HARDENING: ELIMINATING INTERACTIVE DEPENDENCIES

Pipelines evaluated only in interactive sessions often fail in automated production environments. Harden pipelines by eliminating every interactive dependency:

- Set locale and time zone explicitly at the top of every script.
- Define a random seed for any analysis using randomness.
- Treat warnings as errors using "*options(warn = 2)"* so that small issues are not silently missed.
- Validate that all required environment variables exist before execution begins.
- Assert that all input files exist before the pipeline runs.
- Use {***callr***} to test code in a clean, separate R session before deployment; code that passes {***callr***} behaves reliably in CI.
- {***targets***} pipelines run headless by design and track which steps need re-execution, making every analysis consistent and reproducible.

- Scale to distributed execution using the {***crew***} package and cloud backends, removing the need to write custom scheduling code or manage cron jobs.

4. NETWORK MINIMIZATION IN AUTOMATION

Batch pipelines that access the internet at runtime create two problems for clinical teams. Results may not be reproducible because external services can change. Sensitive clinical data may also leave the secure environment accidentally. Best practices include:

- Bundle all packages and reference data into the Docker image during the controlled build, not at runtime.
- Run production containers with outbound network access blocked or restricted to approved internal registries.
- Use a private package repository such as Posit Package Manager as the single approved source for R packages.

The following flow diagram presents a reference architecture integrating the practices described across the development, security, and operations layers into a cohesive, pharma-grade clinical R environment.

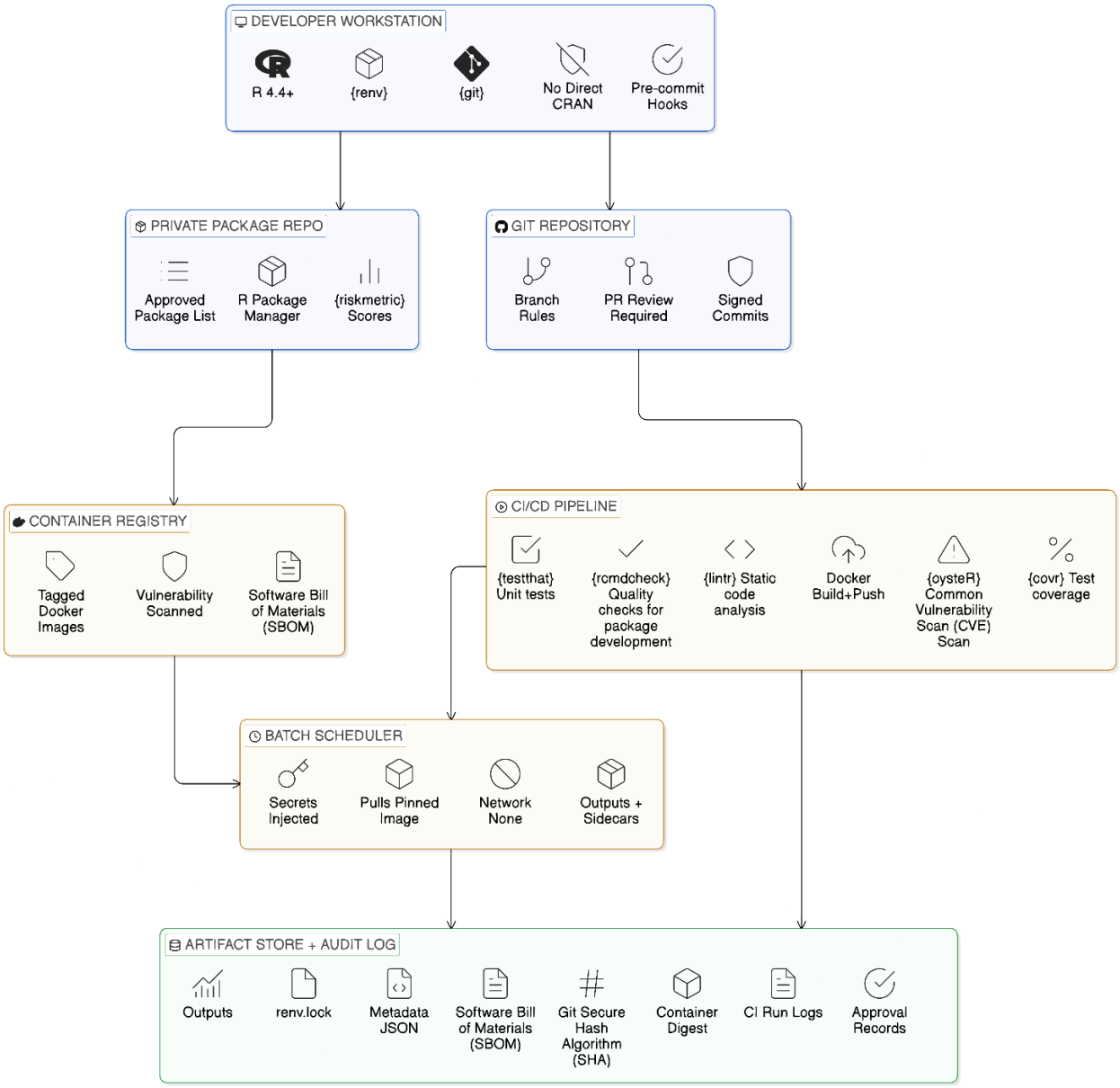

Figure 6: Reference Architecture for a Pharma-Grade Clinical R Environment

GXP ALIGNMENT AND FDA CONTEXT

The DEV, SEC, and OPS practices described in this paper map directly to requirements under the current and evolving regulatory landscape. The key frameworks and their relevance can be explored further –

- **FDA 21 CFR Part 11 (Electronic Records and Signatures):** requires audit trails, access control, and validation of computerized systems. The October 2024 final Q&A guidance on electronic systems in clinical investigations clarifies applicability to sponsors, CROs, and IRBs and reinforces risk-based validation approaches (FDA 2024b).
- **FDA Computer Software Assurance (CSA) Guidance (September 2025, updated February 2026):** replaces traditional Computer System Validation (CSV) with a risk-based, least-burdensome approach. CI/CD pipelines, container image validation, and automated test evidence align directly with the CSA framework (FDA 2025).
- **EU GMP Annex 11 (2025 draft, final expected mid-2026):** the revised draft significantly expands requirements for lifecycle management, data integrity, audit trails, cybersecurity, and cloud-based system oversight. The new Annex 22 on Artificial Intelligence also establishes governance requirements for ML models in regulated environments (EMA/PIC/S 2025).
- **CDISC Analysis Results Standard (ARS) v1.0 (2024):** requires analysis results to link directly to source data, programs, and metadata. The OPS layer sidecar artifact fulfills this requirement automatically.
- **TransCelerate Modernized Statistical Analytics (MSA) Framework:** principles of accuracy, traceability, and reproducibility are operationalized by the three-layer architecture in this paper.
- **FDA R Submissions Pilot program (Pilots 3, 4, and 5):** demonstrates that R-based submissions using containerized environments and {***riskmetric***} assessments are accepted. FDA reviewers increasingly expect structured, documented, and traceable pipeline artifacts.
- **FDA Agentic AI deployment (2025):** with the FDA now using AI to review submissions (PDA 2025), machine-readable metadata in sidecars and structured audit trails become especially critical for review efficiency.

DISCUSSION AND FUTURE DIRECTIONS

The DevSecOps practices outlined in this paper address risks inherent in scaling clinical R environments. However, their adoption varies significantly across the pharmaceutical industry. In most pharmaceutical organizations, version control systems and basic testing frameworks have been implemented to some extent. However, containerized reproducible builds are less commonly established, along with vulnerability scanning and formal governance policies for large language models (LLMs). These gaps highlight critical areas for near-term improvement and targeted investment -

1. REGULATORY CONVERGENCE IS CREATING A NARROW ADOPTION WINDOW

The regulatory environment is evolving rapidly, often outpacing many pharmaceutical R teams. Guidance from the FDA CSA, released in 2025, the EU GMP Annex 11 draft published in 2025, and the introduction of Annex 22 relating to AI governance all demonstrate the ongoing shift toward risk-based, evidence-driven software assurance as a standard expectation. Teams that proactively implement CI/CD pipelines, automated quality controls, and supplementary audit artifacts will be well-positioned to comply with these requirements now in effect. In contrast, teams that postponed adaptation may face substantially greater remediation challenges.

2. LLM GOVERNANCE NEEDS ORGANIZATIONAL POLICY BEFORE INDIVIDUAL ADOPTION

LLMs are already embedded in development workflows through IDE plugins, code completion tools, and code review assistants. In most current practices within pharmaceutical companies, individual developers independently figure out which data to share with specific LLM services on an ad hoc basis. Organizations need explicit governance policies that classify data types, define logging requirements, and establish contractual obligations with LLM providers. Standardized regulatory guidance on LLM use in clinical programming does not exist and this gap should be proactively filled.

3. COMMUNITY TOOLING AND SHARED INFRASTRUCTURE CAN REDUCE ADOPTION BARRIERS

The pharmaverse ecosystem is converging around shared standards. Packages such as {***admiral***}, {***teal***}, and {***riskassessment***} are becoming de-facto baselines. The next step is shared container base images, {***renv***} lockfile templates, and cross-organization pre-validated package lists maintained through the R Validation Hub. These investments reduce duplicated effort and make DevSecOps adoption accessible to smaller teams that cannot build infrastructure from scratch.

4. NUMERIC AND CROSS-LANGUAGE REPRODUCIBILITY REMAIN OPEN PROBLEMS

Current approaches address software reproducibility well. Achieving exact numeric reproducibility across hardware, compilers, and basic linear algebra subprograms (BLAS) libraries is still difficult for certain statistical and machine learning algorithms. As AI approaches appear in regulatory submissions, the community needs formal guidance on acceptable numeric tolerances. Additionally, many pharmaceutical teams run R and Python in the same pipeline. Extending {***renv***} and {***oysteR***} principles to Python environments using tools such as "*poetry.lock*" and "*pip-audit"* is technically straightforward but organizationally underinvested.

CONCLUSION

R-based clinical programming has matured into a dependable and submission-ready discipline. Sustaining this standard requires that organizations embed security, auditability, and reproducibility as core engineering principles from the outset, rather than treating them as compliance requirements to be addressed retroactively.

Drawing from DevSecOps discussions, the three-layer architecture provides a robust framework that incorporates several best practices with:

- ***Development layer:*** traceable code, reproducible environments, and automated quality gates.
- ***Security layer:*** secrets management, package vulnerability assessment, and LLM governance.
- ***Operations layer:*** execution audit trails, batch hardening, and GxP-compliant traceability artifacts.

Best practices can be introduced beginning with version control and basic unit testing, introducing lockfiles when reproducibility concerns arise, containerizing environments to support regulatory reruns, and incorporating vulnerability scanning to enable prompt responses to security inquiries. Adopting this approach requires a cultural transition in which these standards become shared within team practices, reinforced through rigorous peer review, comprehensive test coverage, and transparent development workflows. With the FDA increasingly leveraging AI in submission reviews, teams that provide well-structured, version-controlled, and fully documented pipeline artifacts can navigate regulatory scrutiny with increased confidence. This positions pharmaceutical organizations, CROs and consultants to accelerate innovation, streamline submissions building lasting trust with regulators and stakeholders alike. By prioritizing robust and innovative engineering strategies using open-source utilities, the industry can meaningfully shape clinical outcomes and advance the evolution of data-driven medicine for years to come.

REFERENCES

eCFR. 2026. *21 CFR Part 11: Electronic Records; Electronic Signatures*. https://ecfr.gov/current/title-21/chapter-I/subchapter-A/part-11

EMA/PIC/S. 2025. *Draft Annex 11 (Computerised Systems) & Draft Annex 22 (Artificial Intelligence)*. EudraLex Vol. 4 GMP (public consultation). https://health.ec.europa.eu/consultations/stakeholders-consultation-eudralex-volume-4-good-manufacturing-practice-guidelines-chapter-4-annex_en

FDA. 2003. *Part 11, Electronic Records; Electronic Signatures: Scope and Application*. https://fda.gov/regulatory-information/search-fda-guidance-documents/part-11-electronic-records-electronic-signatures-scope-and-application

FDA. 2018. *Computerized Systems Used in Clinical Trials* (Guidance for Industry). https://fda.gov/inspections-compliance-enforcement-and-criminal-investigations/fda-bioresearch-monitoring-information/guidance-industry-computerized-systems-used-clinical-trials

FDA. 2024a. *Study Data Technical Conformance Guide* (v5.9). https://fda.gov/industry/fda-data-standards-advisory-board/study-data-standards-resources

FDA. 2024b. *Electronic Systems, Electronic Records, and Electronic Signatures in Clinical Investigations: Q&A* (Final guidance, Oct 2024). https://www.federalregister.gov/documents/2024/10/02/2024-22562/electronic-systems-electronic-records-and-electronic-signatures-in-clinical-investigations-questions

FDA. 2025. *Computer Software Assurance for Production and Quality Management System Software* (Final guidance; updated Feb 2026). https://www.fda.gov/regulatory-information/search-fda-guidance-documents/computer-software-assurance-production-and-quality-management-system-software

FDA. 2026. *openFDA* (GitHub repository). https://github.com/FDA/openfda/

HiddenLayer. 2024. *R-bitrary Code Execution (CVE-2024-27322)*. https://hiddenlayer.com/research/r-bitrary-code-execution

Jumping Rivers. 2025. *Detecting Security Vulnerabilities in R Packages Using {oysteR}*. https://jumpingrivers.com/blog/r-package-vulnerabilities-security/

Landau WM. 2021. *The targets R package*. JOSS 6(57):2959. https://doi.org/10.21105/joss.02959

OSV. 2026. *Open Source Vulnerabilities (OSV) Database*. https://osv.dev/

PDA. 2025. *News Brief: FDA Expands AI with Agentic Deployment*. https://pda.org/pda-letter-portal/home/full-article/news-brief-fda-expands-ai-with-agentic-deployment

pharmaR. 2020. *A Risk-Based Approach for Assessing R Package Accuracy within a Validated Infrastructure*. R Validation Hub white paper. https://pharmar.org/white-paper/

pharmaverse. 2024. *A Connected Network of R Packages for Clinical Reporting*. https://pharmaverse.org

R Consortium. 2025. *R Submissions Working Group: 2026 Plans and 2025 Success*. https://r-consortium.org/posts/submissions-wg-2026/

R Validation Hub. 2025. *riskmetric* (v0.2.5). https://pharmar.github.io/riskmetric/

Rocker Project. 2025. *Versioned Docker Images for R*. https://rocker-project.org

Sonatype. 2025. *OSS Index*. https://ossindex.sonatype.org

TransCelerate. 2022. *Modernization of Statistical Analytics (MSA) Framework*. https://transceleratebiopharmainc.com/modernization-of-statistical-analytics/

Ushey K, Wickham H. 2026. *renv: Project Environments* (v1.1.6). https://rstudio.github.io/renv/

Wickham H. 2011. *testthat: Get Started with Testing*. The R Journal 3:5–10. https://journal.r-project.org/articles/RJ-2011-002/

ACKNOWLEDGEMENT

I thank the leadership and my colleagues at Ephicacy. I am especially grateful to Mrityunjay Kumar, Akshata J Salian, Tyagrajan Swaminathan, Trupti Hemant Bal and Harsha Dyavappa for their valuable guidance and mentorship. I am deeply indebted to my family for their support, encouragement and motivation throughout. I also thank the PharmaSUG 2026 organizers for this opportunity and the global open source software communities for continuously advancing innovation with collaboration.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Indraneel Chakraborty
Technical Lead – Statistical Programming
Ephicacy Lifescience Analytics Pvt Ltd, Bengaluru, India
Email: indraneel.chakraborty@ephicacy.com
LinkedIn: https://www.linkedin.com/in/indraneelchakraborty/