Paper AS13

Building an Open-Source-First Statistical Computing Environment:

A Cloud-Native Approach with R, Python, and Kubernetes

Rafael Pereira, *Appsilon*
Amanda Lopuski, *Solid Biosciences*

ABSTRACT

Modernizing Statistical Computing Environments (SCEs) is essential for enabling faster, more flexible, and compliant clinical workflows. This paper presents a case study of how Appsilon partnered with Solid Biosciences to design and implement a validated, cloud-native clinical analysis environment on AWS, leveraging container technology and Kubernetes orchestration. The solution enables secure ingestion, analysis, and reporting of clinical data using open-source tools within a fully GxP-compliant framework.

What makes this implementation distinctive is Solid Biosciences' decision to build their clinical development infrastructure from the ground up with R and Python as primary languages, rather than following the traditional SAS-first approach. This paper examines the technical architecture, the advantages of an open-source-first strategy, and the practical lessons learned from building an inspection-ready, modular SCE that accelerates delivery while maintaining regulatory compliance.

INTRODUCTION

The Changing Landscape of Statistical Computing

Statistical Computing Environments have supported regulated clinical development for decades, but many of these systems are showing their age. Legacy setups, often built around SAS, were designed for a different era when studies were simpler, timelines longer, and open-source tools less common. Today, data volumes have exploded; teams expect tools like R, Python, and Git, and regulatory expectations are rising. These shifts demand a platform that is not only compliant but flexible, collaborative, and scalable.

R has been the major driving force for open-source adoption in pharmaceutical statistical computing. Since the first R-based FDA submissions, organizations have increasingly recognized that open-source languages offer capabilities that match or exceed proprietary alternatives. Python is now emerging as a complementary force, bringing data engineering patterns and machine learning capabilities that align with broader technology industry standards.

This convergence is bringing pharmaceutical computing closer to the data engineering standards of the technology industry overall. Modern SCEs are adopting practices like Infrastructure as Code, container orchestration, and GitOps that have been standard in tech companies for years. The result is a new generation of computing environments that can deliver both regulatory compliance and operational agility.

Open Source as the Gateway to AI

Perhaps the most compelling argument for open-source adoption today is its role as a key enabler for artificial intelligence. Organizations embracing modern open-source practices are

adopting AI capabilities significantly faster than their peers still reliant on legacy proprietary systems.

This correlation is not coincidental. The same infrastructure that supports open-source workflows—containerization, version control, cloud-native architecture, and programmatic interfaces—provides the foundation for AI integration. Organizations that have already invested in these capabilities can deploy AI tools with minimal additional infrastructure work. In contrast, those operating on legacy platforms often face months or years of modernization before AI becomes practically accessible.

For pharmaceutical organizations evaluating their SCE strategy, this creates a compounding advantage: faster open-source adoption leads to faster AI adoption. The organizations that modernize today will be positioned to leverage AI for clinical programming, automated QC, intelligent document generation, and advanced analytics—capabilities that are rapidly moving from experimental to essential.

The Case for Open-Source-First

Solid Biosciences made a strategic decision to build their clinical analytics infrastructure with R and Python as primary languages from day one. This open-source-first approach, rather than layering open-source capabilities onto an existing SAS infrastructure, offered several distinct advantages:

- **Design flexibility:** Building without legacy constraints allowed the system to be designed around modern workflows and organizational needs, rather than accommodating historical technical debt.
- **Tool ecosystem:** A wider array of tools and integrations become available when the platform is built around open-source from the start, including the pharmaverse ecosystem, modern IDEs, and cloud-native services.
- **Infrastructure transparency:** IT and platform engineering teams gain better visibility into system behavior when working with open-source components, enabling more effective troubleshooting and optimization.
- **Security options:** More comprehensive and configurable security controls are available through cloud-native and open-source security tooling.
- **Incremental updates:** Modular open-source components can be updated independently, avoiding the all-or-nothing upgrade cycles common with monolithic commercial platforms.

The cost structure also shifts with an open-source approach. While licensing costs decrease, investment moves toward setup, customization, and maintenance. This tradeoff provides more flexibility in how resources are allocated and enables the organization to build capabilities that directly address their specific needs.

ARCHITECTURE OVERVIEW

Containers: The Foundation for Reproducibility

At the core of our architecture is container technology. Containers provide the essential capability for reproducibility that regulatory compliance demands. A container encapsulates the exact R version, all package dependencies with their specific versions, operating system libraries, and environment configurations needed to reproduce an analysis.

This approach directly addresses one of the persistent challenges in validated environments: ensuring that analysis code produces identical results regardless of when or where it runs.

Traditional SCEs often struggle with package management and environmental drift. With containers, the principle of "write once, run anywhere" ensures that the exact computational environment is preserved and can be reconstructed at any point.

From the Platform Engineering perspective: Containers simplify deployment, scaling, and maintenance. Each workload runs in isolation with precisely defined resource requirements. This isolation also enhances security by limiting the blast radius of any potential issues.

From the Biometrics perspective: Containers mean that the development environment exactly matches production. Teams can work locally with the same tooling and packages that will be used for validated analyses, eliminating the friction of environment mismatches.

Kubernetes: The Modern Operating System

Kubernetes serves as the orchestration layer for our containerized workloads. We position Kubernetes as the modern operating system for running these containers at scale. It provides the capabilities needed for enterprise-grade clinical computing:

- **Automated scaling:** Resources scale up during submission crunch times and scale down during quieter periods, optimizing cost without manual intervention.
- **Self-healing:** Failed workloads are automatically restarted, maintaining availability without operator intervention.
- **Declarative configuration:** The desired state of the system is defined in version-controlled manifests, enabling reproducible infrastructure.
- **AI/ML readiness:** Kubernetes is the standard platform for modern AI and machine learning workloads, positioning the SCE for future capabilities.

This architecture also creates a natural bridge to modern data engineering tools. While commercial platforms like Databricks or Snowflake represent a similar licensing-versus-operational cost tradeoff to SAS, Kubernetes enables integration with these tools within an open ecosystem, avoiding vendor lock-in while accessing advanced capabilities when needed.

Git and Infrastructure as Code: Compliance Game Changers

Git-based version control combined with Infrastructure as Code (IaC) fundamentally transforms how compliance is achieved. Rather than treating compliance as an overlay of documentation and manual controls, these technologies make compliance an inherent property of the system.

Git provides: A complete, immutable change record of every modification to code, configuration, and infrastructure. Every change is attributed to a specific person, timestamped, and preserves the full history of what was changed and why. This creates an audit trail that exceeds traditional documentation approaches.

Infrastructure as Code provides: The ability to define the entire computing environment in version-controlled configuration files. Tools like Terraform and Ansible enable automated environment provisioning, disaster recovery, and consistent deployments across development, testing, and production environments.

Together with cloud infrastructure, this combination becomes a game changer for fast iteration and adaptability. Changes that previously required weeks of manual configuration and documentation can be implemented, tested, and deployed in hours while maintaining complete traceability. The environment can rapidly evolve while remaining inspection-ready.

CI/CD: Fast, Clear, and Documented Deployments

Continuous Integration and Continuous Deployment (CI/CD) pipelines complete the automation picture. When code is committed, automated pipelines can run tests, validate configurations, build container images, and deploy to target environments—all without manual intervention. This automation reduces human error and accelerates the path from development to production.

Critically for regulated environments, CI/CD pipelines also generate documentation automatically. Each deployment produces a complete record of what was deployed, when, by whom (or which automated trigger), what tests passed, and what the state of the system was before and after. This generated documentation is more reliable than manual records because it is produced programmatically from the actual deployment process rather than documented after the fact.

The combination of Git history, IaC definitions, and CI/CD logs creates a comprehensive audit trail that satisfies regulatory requirements while reducing the documentation burden on teams. Compliance becomes a byproduct of good engineering practice rather than a separate workstream.

Storage and Data Integration

Modern cloud storage, particularly object storage like AWS S3, provides flexibility that was difficult to achieve with traditional file shares. Containers provide the flexibility to access this storage with comprehensive role-based access control while maintaining cost efficiency.

The power of R and Python comes with extensive integration capabilities. Libraries exist for connecting to virtually any data source, from CDISC-formatted clinical data to real-world evidence databases. This flexibility enables an architecture designed to support any data integration need enabled by the package ecosystem.

Key storage architecture principles include decoupling storage from compute, implementing granular access controls at the study level, maintaining versioning and immutability for audit purposes, and ensuring encryption both in transit and at rest.

R PACKAGE VALIDATION WITH RISK-BASED METHODOLOGY

A critical component of any R-based SCE is a systematic approach to package validation. We integrated the R Validation Hub's risk-based methodology to support compliant R package validation aligned with 21 CFR Part 11 requirements.

The approach moves away from treating each R package as a one-off validation project. Instead, packages are assessed based on quantifiable risk metrics including development best practices, code documentation, community engagement, and maintenance patterns. This assessment produces a risk score that guides the level of validation effort required.

Our implementation includes automated assessment of packages against organizational rules, dependency analysis to flag potential risks in package supply chains, historical tracking to monitor packages for changes that might affect validation status, and role-based workflows for package review and approval.

This approach significantly reduces the validation burden while maintaining rigor where it matters most. Low-risk packages with strong community adoption and robust testing can move through validation quickly, while higher-risk packages receive appropriate scrutiny.

IMPLEMENTATION INSIGHTS

Dual Perspective: Platform Engineering and Biometrics

One of the key success factors in this implementation was maintaining close collaboration between platform engineering and Biometrics perspectives throughout the design and build process. Each architectural decision was evaluated from both viewpoints:

Component	Platform Engineering View	Biometrics View
Containers	Simplified deployment, isolated workloads, defined resource limits	Dev environment matches production, reproducible analyses
Kubernetes	Auto-scaling, self-healing, declarative management	Reliable compute, resources available when needed
Git/IaC	Reproducible infrastructure, automated provisioning, and controlled promotion between environments	Version control for code and configs, audit trail, controlled promotion between environments
Package Validation	Automated assessment, managed repositories	Access to validated tools, reduced friction

Key Outcomes

The implementation delivered a validated, scalable environment that supports both regulatory submissions and exploratory research. Key outcomes include:

- **Unified workflows:** Analysts can explore in R and submit in R, eliminating the "two-environment problem" where work must be repeated across different systems.
- **Validation zones:** The system supports both validated zones for regulatory work and flexible zones for innovation, using the same tools with different control levels.
- **Modular architecture:** Components can be updated independently without triggering full system revalidation.
- **Automated compliance:** Audit trails are generated automatically through Git workflows and infrastructure logging.
- **Scalability:** The system handles concurrent users and scales during peak submission periods.

LESSONS LEARNED AND GUIDANCE

For organizations considering similar modernization initiatives, we offer the following guidance based on our experience:

1. **Start with goals, not technology.** Design around purpose and organizational needs, not just compliance checkboxes. The best SCE is one that fits your current needs while scaling with future study pipelines.
2. **Invest in modularity.** Flexibility today prevents disruption tomorrow. Traditional monolithic SCEs become all-or-nothing systems where small improvements require major projects.
3. **Support open source deliberately.** Modern teams expect R, Python, and version control. SCEs that don't support these languages risk being sidelined as new talent enters already trained in these tools.

4. **Balance control and innovation.** Isolate validated work without limiting experimentation. Many organizations apply overly strict compliance requirements to exploratory platforms, stifling innovation.
5. **Partner where needed.** External support accelerates success and reduces risk. Building cloud infrastructure, validation requirements, and pharma workflows together requires specialized expertise.

CONCLUSION

Modernizing a Statistical Computing Environment is no longer optional for pharmaceutical organizations that want to stay competitive, compliant, and collaborative. The convergence of open-source analytics tools, container technology, and cloud-native infrastructure creates an opportunity to build environments that are both more capable and more adaptable than traditional approaches.

The Solid Biosciences case demonstrates that an open-source-first approach is viable for clinical development. By building from the ground up with R and Python rather than layering onto legacy SAS infrastructure, organizations can create systems that are designed for modern workflows while meeting all regulatory requirements.

The most successful implementations recognize that an SCE is not just infrastructure; it is the foundation that determines whether an organization can adapt to modern drug development or gets left behind. Building that foundation on containers, Kubernetes, Git, and Infrastructure as Code creates a platform that can evolve with the industry.

REFERENCES

1. R Validation Hub. "A Risk-based Approach for Assessing R Package Accuracy within a Validated Infrastructure." https://www.pharmar.org/
2. FDA. "Statistical Software Clarifying Statement." May 2015.
3. R Consortium. "R Consortium Submission Working Group." https://rconsortium.github.io/submissions-wg/
4. Pharmaverse. "Open-source Tools for Clinical Reporting." https://pharmaverse.org/
5. Appsilon. "The Anatomy of Modern Statistical Computing Environments in Pharma." 2025.
6. Appsilon. “Open Source Adoption as an Indicator of AI Readiness” 2026
7. 21 CFR Part 11. Electronic Records; Electronic Signatures.

CONTACT INFORMATION

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

Rafael Pereira, Platform Unit Lead

Appsilon

rafael.pereira@appsilon.com

Amanda Lopuski, Statistical Programming Director, Biometrics

Solid Biosciences

alopuski@solidbio.com

ACKNOWLEDGMENTS

The authors would like to thank the Solid Biosciences team for their collaboration and willingness to share their implementation journey. We also acknowledge the contributions of the R Validation Hub, the pharmaverse community, and the broader open-source ecosystem that makes this work possible.

DISCLAIMER

The contents of this paper are the work of the authors and do not necessarily represent the opinions, recommendations, or practices of their respective organizations.