Neutrosophic Computing and Machine Learning, Vol. 44, 2026

University of New Mexico

Automatización de despliegue y mejora continua en la plataforma SECOED V3. Gestión de plan piloto

Deployment automation and continuous improvement in the SECOED V3 platform. Pilot plan management

Ángela Yanza Montalván[1], Flavio Franco Campos Inga[1], Keneth Bryan Riera Rizzo[1]

[1] Universidad de Guayaquil, Facultad de Ciencias Matemáticas y Físicas, Carrera de Software, Guayaquil, Ecuador

Correspondencia: angela.yanzam@ug.edu.ec · flavio.camposi@ug.edu.ec · keneth.rierar@ug.edu.ec

Resumen. La plataforma SECOED V3 de la Universidad de Guayaquil ejecutaba sus despliegues de forma manual y directa al entorno de producción, lo que generaba tiempos de entrega prolongados, alta dependencia operativa, mayor tasa de errores humanos y limitada capacidad de respuesta ante el cambio. El objetivo de este trabajo fue desarrollar un sistema automatizado de Integración y Despliegue Continuo (CI/CD) con herramientas de código abierto, que estandarizara el ciclo de vida del software e incorporara la seguridad y la calidad como compuertas del propio canal de entrega. La metodología fue de enfoque mixto y tecnológico, bajo el marco ágil Scrum. Se implementaron pipelines en Jenkins con agentes efímeros (checkout, construcción, pruebas, análisis de calidad y despliegue), inspección de código con SonarQube, escaneo de vulnerabilidades de imágenes con Trivy, contenedorización con Docker y un mecanismo de reversión (rollback). El sistema eliminó la intervención manual sobre el servidor de producción. La solución se validó mediante un plan piloto controlado con cuatro docentes, sin fallos críticos y con percepciones de satisfacción predominantemente positivas en usabilidad, estabilidad y velocidad; los seis criterios de aceptación definidos se cumplieron. Se concluye que la automatización del despliegue reduce los riesgos operativos del proceso manual y prepara la plataforma para un despliegue a mayor escala.

Palabras clave: *SECOED V3, integración continua, despliegue continuo, DevOps, Jenkins, Docker, SonarQube, Trivy, plan piloto, mejora continua.*

Abstract. The SECOED V3 platform performed manual deployments directly to production, causing long delivery times, operational dependency and human errors. The goal was to develop an automated CI/CD system using open-source tools, embedding security and quality as gates of the delivery pipeline, under a mixed, technological approach with Scrum. Jenkins pipelines with ephemeral agents, SonarQube code inspection, Trivy container scanning, Docker containerization and a rollback mechanism were implemented, removing manual server intervention. The solution was validated through a controlled pilot with four teachers, without critical failures and with predominantly positive

satisfaction; all six acceptance criteria were met. Deployment automation reduces the operational risks of the manual process and prepares the platform for larger-scale deployment.

Keywords: SECOED V3, continuous integration, continuous deployment, DevOps, Jenkins, Docker, SonarQube, Trivy, pilot project, continuous improvement.

1. Introducción

La capacidad de entregar software de forma rápida, confiable y repetible constituye hoy un factor determinante de la competitividad de las organizaciones [1], [3]. La cultura DevOps, al integrar el desarrollo y las operaciones mediante automatización, colaboración y medición continua, ha demostrado reducir los tiempos de entrega y elevar la estabilidad operativa [2], [3]. Su adopción exige estandarizar el ciclo de vida del software y desplazar las verificaciones de calidad y seguridad hacia etapas tempranas del flujo de entrega, algo que numerosas instituciones del sector público aún no han alcanzado.

La plataforma SECOED V3, de la Facultad de Ciencias Matemáticas y Físicas de la Universidad de Guayaquil, ilustra esta brecha. Sus despliegues se ejecutaban manualmente y de forma directa sobre el entorno de producción, lo que prolongaba los tiempos de entrega de nuevas funcionalidades y correcciones, multiplicaba el riesgo de errores humanos durante la integración, las pruebas y el despliegue, y limitaba la capacidad de respuesta de la institución ante los cambios. La ausencia de un canal automatizado de Integración y Despliegue Continuo (CI/CD) impedía además incorporar controles sistemáticos de calidad y seguridad antes de liberar cada versión, y concentraba el conocimiento operativo en pocas personas, generando un riesgo de continuidad.

De esta problemática se derivan dos preguntas de investigación: ¿cómo un sistema automatizado de CI/CD basado en Jenkins, Docker y SonarQube puede reducir los tiempos de entrega de nuevas funcionalidades y correcciones en SECOED V3?, y ¿cómo dicha automatización puede mitigar los errores humanos durante la integración, las pruebas y el despliegue, garantizando la estabilidad técnica requerida para la gestión de un plan piloto?

Para responderlas, el presente trabajo diseña e implementa un sistema de CI/CD sustentado en herramientas de código abierto y en la cultura DevOps, y lo valida mediante un plan piloto controlado con docentes. La contribución es un artefacto de software replicable que estandariza el ciclo de vida en un contexto universitario con recursos acotados, integrando la calidad y la seguridad como parte del propio proceso de entrega.

2. Trabajos relacionados y marco teórico

2.1 Entrega continua, DevOps y agilidad.

La entrega continua se define como la capacidad de poner en producción cambios de cualquier tipo de manera segura, rápida y sostenible [1]. El enfoque DevOps amplía esta noción articulando el flujo de trabajo, la retroalimentación y el aprendizaje continuo a lo largo del ciclo de vida [2], y se sintetiza en el modelo de las Tres Vías: maximizar el flujo de izquierda a derecha, habilitar la retroalimentación temprana y promover el aprendizaje continuo. La investigación empírica asociada demuestra que las

organizaciones de alto desempeño despliegan con mayor frecuencia y se recuperan más rápido de los incidentes [3]. La estandarización del proceso bajo marcos como ISO/IEC 29110 se ha propuesto como puente hacia la adopción de DevOps en organizaciones pequeñas [10], y la migración desde sistemas de control de versiones heredados constituye un habilitador frecuente de estas iniciativas [11]. La gestión del proyecto se organizó bajo el marco ágil Scrum [4], complementado con buenas prácticas de dirección de proyectos [5].

Orquestación y contenedorización. Un canal de CI/CD se compone de etapas automatizadas de integración, prueba, análisis y despliegue. Jenkins es un servidor de automatización de código abierto que permite versionar el pipeline como código (Jenkinsfile) y extender su funcionalidad mediante un amplio ecosistema de plugins, ofreciendo control total a costa de una mayor curva de aprendizaje [9]. Frente a alternativas SaaS como GitHub Actions —de integración nativa con el repositorio pero menor control sobre la infraestructura— se seleccionó Jenkins por su autonomía y su despliegue autoalojado en la infraestructura institucional [9]. La contenedorización con Docker garantiza la reproducibilidad del entorno y, mediante el etiquetado semántico de imágenes, un historial inmutable de cada versión [7].

Calidad y seguridad del software. La calidad del código se inspecciona con SonarQube, que reporta deuda técnica y vulnerabilidades y permite definir compuertas de calidad (Quality Gates) que condicionan el avance del pipeline [6]. La seguridad de la cadena de suministro se refuerza con Trivy, un escáner de vulnerabilidades especializado en imágenes de contenedores, bibliotecas y archivos de configuración [8]. La incorporación de ambas herramientas como compuertas del flujo materializa el principio de desplazar a la izquierda (shift-left) la verificación de calidad y seguridad, reduciendo el costo de corregir defectos y el riesgo de liberarlos a producción.

3. Materiales y métodos

a. **Diseño y enfoque.** La investigación adoptó un enfoque mixto y tecnológico, orientado a la construcción y validación de un artefacto de software, bajo el marco ágil Scrum [4]. El proceso se estructuró en fases de definición del problema, análisis de involucrados, diseño de la solución, implementación incremental por sprints y evaluación mediante criterios de aceptación y un plan piloto controlado. El problema central se definió como la falta de un proceso automatizado de despliegue y mejora continua, generador de ineficiencias operativas, mayor tasa de errores, retrasos en las entregas y baja adaptabilidad al cambio. El levantamiento del estado actual (AS-IS) frente al estado deseado (TO-BE) evidenció el contraste entre el despliegue manual y el canal automatizado propuesto (Figura 1).

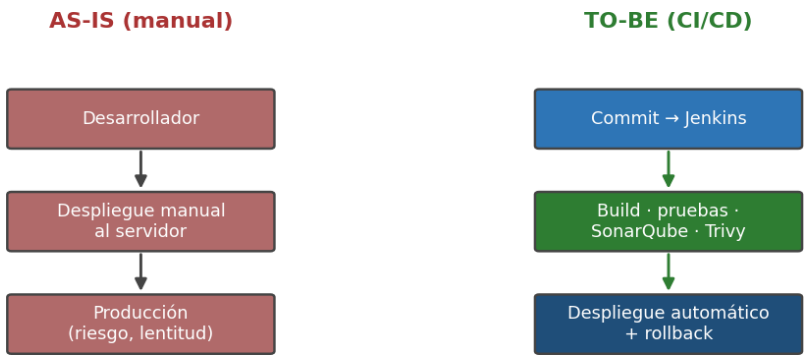

Figura 1. Comparación del proceso de despliegue AS-IS (manual) y TO-BE (CI/CD automatizado).

Organización Scrum. El trabajo se organizó en un Product Backlog y sucesivos Sprint Backlogs. La Tabla 1 resume las fases y los entregables. La planificación abarcó el levantamiento de los modelos AS-IS y TO-BE y los diagramas de arquitectura y de flujo; la implementación comprendió el desarrollo de los pipelines CI/CD, el arranque del plan piloto, la configuración en producción y las encuestas de satisfacción; y el cierre incluyó pruebas de resiliencia, retrospectiva y la entrega de los manuales. Las historias de usuario guiaron cada incremento; por ejemplo, la automatización del análisis estático con SonarQube y la validación de un Quality Gate como condición del despliegue se planificaron como una historia del líder técnico con criterio de aceptación explícito.

Tabla 1. Fases y entregables del proceso Scrum.

Fase	**Entregables principales**
Planificación	Modelos AS-IS y TO-BE; diagramas de arquitectura y de flujo; Product Backlog
Implementación	Pipelines CI/CD; configuración en producción; arranque del plan piloto
Pruebas	Pruebas de resiliencia; encuestas de satisfacción del plan piloto
Cierre	Retrospectiva; manuales técnico, de configuración, de usuario y de gestión del piloto

Selección de herramientas. Para la orquestación se evaluaron Jenkins y GitHub Actions; se optó por Jenkins por su carácter de código abierto, su control total sobre el flujo y su integración con el ecosistema institucional [9]. La pila tecnológica se completó con Docker para la contenedorización [7], SonarQube para el análisis estático de calidad [6] y Trivy para el escaneo de vulnerabilidades de imágenes [8]. La Tabla 2 resume las herramientas y su función.

Tabla 2. Pila tecnológica del canal CI/CD y su función.

Herramienta	**Función**

Jenkins (autoalojado)	Orquestación del pipeline (pipeline como código)
Docker	Contenedorización y reproducibilidad del entorno
SonarQube	Análisis estático de calidad y Quality Gate
Trivy	Escaneo de vulnerabilidades de imágenes de contenedor
Docker Hub	Registro de imágenes con etiquetado semántico
Script de rollback	Reversión a la versión previa ante fallos

b. **Arquitectura en tres capas.** La arquitectura se organizó en tres capas (Figura 2). La capa de orquestación se sustenta en un servidor Jenkins autoalojado que, en lugar de ejecutar los trabajos en su sistema operativo base, instancia agentes efímeros (contenedores) por cada ejecución, garantizando entornos limpios y reproducibles. La capa de aseguramiento de calidad y seguridad integra el análisis estático con SonarQube y el escaneo de imágenes con Trivy, actuando como compuertas del flujo. La capa de despliegue y gestión de artefactos publica las imágenes Docker con etiquetado semántico — creando un historial inmutable— y despliega en el servidor de producción de forma automatizada, con un script de rollback que revierte a la versión anterior ante fallos. La gestión de credenciales se realiza de forma segura, evitando su exposición en el código fuente.

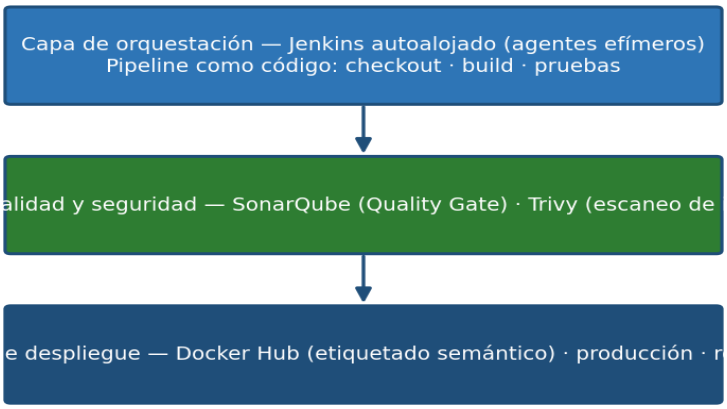

Figura 2. Arquitectura del canal CI/CD en tres capas.

c. **Pipeline y plan piloto.** El canal se diseñó en cinco etapas secuenciales con falla rápida: (i) checkout del código fuente; (ii) construcción de la imagen; (iii) ejecución de pruebas; (iv) análisis de calidad con SonarQube y compuertas de deuda técnica y vulnerabilidades; y (v) despliegue. La validación final se realizó con un plan piloto controlado con cuatro docentes, diseñado para verificar la estabilidad y el rendimiento de SECOED V3 bajo uso real y para medir la percepción de los usuarios mediante una encuesta de satisfacción (escala tipo Likert de 1 a 5) sobre usabilidad, estabilidad y velocidad. El cumplimiento de la solución se evaluó además mediante una matriz de criterios de aceptación.

d. **Justificación y alcance.** La justificación de la propuesta radica en su pertinencia técnica y económica: al emplear exclusivamente herramientas de código abierto y reutilizar la infraestructura de la institución, el sistema se implanta sin costos de licenciamiento y resulta sostenible para una facultad pública. El alcance del trabajo abarca el diseño y la implementación del canal CI/CD y la habilitación y ejecución de un plan piloto controlado con docentes; la operación a escala completa del entorno de producción se delimita como fase posterior, sujeta a una validación ampliada.

4. Resultados

a. **Sistema CI/CD implementado.** Se implementó un pipeline CI/CD completo en Jenkins que automatiza la construcción, las pruebas, el análisis de calidad con SonarQube, el escaneo de seguridad con Trivy y el despliegue, eliminando por completo la intervención manual sobre el servidor de producción. Cada ejecución se realiza en un agente efímero, lo que asegura entornos limpios y reproducibles; las imágenes resultantes se publican con etiquetado semántico, proporcionando trazabilidad de cada versión liberada.

i. **Etapas del pipeline.** El flujo opera en cinco etapas con falla rápida. En el checkout, Jenkins obtiene el código fuente y registra el commit para la trazabilidad. En la construcción se genera la imagen de la aplicación en un agente efímero. La etapa de pruebas ejecuta las verificaciones

automatizadas. El análisis de calidad envía el código a SonarQube y evalúa el Quality Gate: el avance del pipeline queda condicionado al resultado, de modo que un incumplimiento de los umbrales de deuda técnica o de vulnerabilidades detiene la promoción. Finalmente, la etapa de despliegue publica la imagen verificada en el registro y la libera al entorno de producción.

b. **Calidad y seguridad integradas.** El análisis estático con SonarQube genera reportes de deuda técnica y vulnerabilidades y habilita compuertas de calidad que condicionan el avance del pipeline; el escaneo con Trivy detecta vulnerabilidades en las imágenes antes de su publicación. En conjunto, ambas herramientas desplazan la verificación de calidad y seguridad hacia etapas tempranas del flujo, reduciendo el riesgo de liberar defectos a producción.

c. **Resiliencia y rollback.** Se realizaron pruebas de resiliencia para verificar la recuperación del sistema ante fallos en el despliegue. El mecanismo de rollback restauró correctamente la versión inmediatamente anterior a partir del historial inmutable de imágenes, garantizando la continuidad del servicio sin intervención manual y validando la robustez del canal frente a errores de promoción.

d. **Plan piloto y satisfacción.** El plan piloto se ejecutó con cuatro docentes sin presentar fallos críticos. La encuesta de satisfacción mostró percepciones predominantemente positivas: el acceso al sistema se valoró como sencillo y rápido, la interfaz como intuitiva y fácil de entender, y la velocidad de carga de las páginas como adecuada, confirmando la viabilidad operativa de la solución en condiciones de uso real (Tabla 3). Se entregó, asimismo, la documentación técnica, de configuración, de usuario y la guía de gestión del plan piloto.

Tabla 3. Percepción de los docentes en el plan piloto (escala Likert 1–5).

Dimensión evaluada	**Percepción**
Acceso sencillo y rápido	Positiva
Interfaz intuitiva y fácil de entender	Positiva
Velocidad de carga adecuada	Positiva
Estabilidad del sistema	Positiva
Satisfacción general	Satisfecho / Muy satisfecho

El análisis por ítems confirmó la tendencia positiva. La facilidad y rapidez de acceso, la claridad e intuición de la interfaz y la adecuación de la velocidad de carga obtuvieron las valoraciones más altas, mientras que la estabilidad percibida durante la sesión piloto reforzó la confianza en la solución. En conjunto, los docentes manifestaron estar satisfechos o muy satisfechos con la plataforma, lo que respalda la decisión de avanzar hacia una validación de mayor escala.

e. **Criterios de aceptación.** La Tabla 4 presenta la matriz de criterios de aceptación del proyecto, todos satisfechos.

Tabla 4. Cumplimiento de los criterios de aceptación.

Criterio	**Estado**
Funcionalidad del pipeline (todas las etapas)	Cumple
Calidad de código (SonarQube)	Cumple

Seguridad (accesos, credenciales, escaneo)	Cumple
Rollback ante fallos	Cumple
Plan piloto con 4 docentes sin fallos críticos	Cumple
Documentación completa	Cumple

5. Discusión

La automatización del despliegue suprimió la dependencia del procedimiento manual y estandarizó el ciclo de vida del software, incorporando la calidad y la seguridad como compuertas del propio pipeline. Estos resultados son coherentes con la evidencia que asocia la madurez del CI/CD con menores tiempos de entrega y mayor estabilidad [1], [3], y con las propuestas que vinculan la estandarización del proceso a la adopción de DevOps en organizaciones de menor tamaño [10]. El uso de Jenkins, Docker, SonarQube y Trivy demuestra que es posible construir un canal robusto y auditable con herramientas de código abierto, sin licencias, en una institución pública con recursos limitados.

a. **Impacto y replicabilidad.** En términos de impacto, la solución beneficia directamente al equipo técnico de la plataforma —que deja de depender de despliegues manuales y gana trazabilidad y capacidad de reversión— e, indirectamente, a los docentes y usuarios de SECOED V3, que reciben versiones más estables y entregadas con mayor rapidez. Al sustentarse íntegramente en software libre y reutilizar la infraestructura institucional, la propuesta es replicable en otras plataformas de la Universidad con un costo de licencias nulo, y constituye una referencia para la adopción de prácticas DevOps en el sector público universitario ecuatoriano.

b. **Limitaciones y trabajo futuro.** Entre las limitaciones se cuentan el tamaño del piloto (cuatro docentes) y la ejecución acotada, que restringen la generalización de los resultados. Como trabajo futuro se propone ampliar la muestra de usuarios, ejecutar el despliegue a escala real y consolidar métricas cuantitativas de rendimiento DevOps —frecuencia de despliegue, tiempo de entrega de cambios, tasa de fallo de los cambios y tiempo medio de recuperación—, que permitirán cuantificar de forma objetiva la mejora respecto del proceso manual.

6. Conclusiones

Se desarrolló e implementó un sistema de Integración y Despliegue Continuo para la plataforma SECOED V3 con herramientas de código abierto, que automatiza el ciclo de vida del software, integra análisis de calidad y seguridad y elimina la intervención manual en producción. El plan piloto con docentes validó la funcionalidad y la estabilidad de la solución sin fallos críticos, con percepciones de satisfacción positivas en usabilidad, estabilidad y velocidad, y se cumplió la totalidad de los criterios de aceptación. Las pruebas de resiliencia confirmaron la capacidad de rollback del canal. Se recomienda escalar progresivamente la solución al despliegue real, ampliando la muestra de usuarios y consolidando las métricas de rendimiento antes de la generalización.

Referencias

[1] J. Humble and D. Farley, Continuous Delivery: Reliable Software Releases through Build, Test, and

Deployment Automation. Addison-Wesley, 2010.

[2] G. Kim, J. Humble, P. Debois, and J. Willis, The DevOps Handbook. IT Revolution Press, 2016.

[3] N. Forsgren, J. Humble, and G. Kim, Accelerate: The Science of Lean Software and DevOps. IT Revolution Press, 2018.

[4] K. Schwaber and J. Sutherland, "The Scrum Guide," Scrum.org, 2020.

[5] Project Management Institute, A Guide to the Project Management Body of Knowledge (PMBOK Guide), 7th ed. PMI, 2021.

[6] SonarSource, "SonarQube documentation," 2024. [Online]. Available: https://docs.sonarsource.com

[7] D. Merkel, "Docker: Lightweight Linux containers for consistent development and deployment," Linux Journal, vol. 2014, no. 239, 2014.

[8] Pardo Minaya, "Análisis de seguridad de contenedores con Trivy," 2024.

[9] Jenkins Project, "Jenkins declarative pipeline syntax," 2024. [Online]. Available: https://www.jenkins.io/doc

[10] G.-C. Sergio, M. Mirna, A. Daniela, and M. A. Pastrana, "Estandarización y continuidad: el puente entre ISO/IEC 29110 y DevOps," RISTI, pp. 5–22, 2024.

[11] V. Singh, A. Singh, A. Aggarwal, and S. Aggarwal, "DevOps based migration from legacy version control to advanced distributed VCS," 2021.